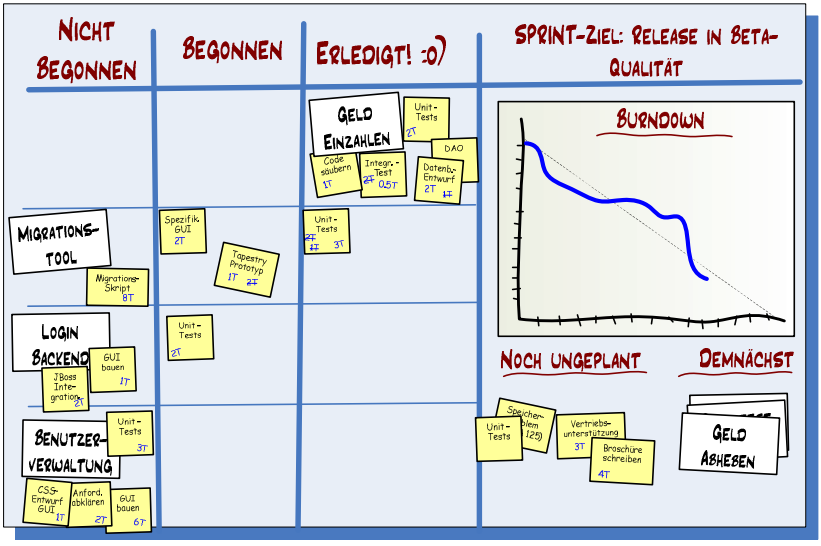
Ein agiler Praxisbericht

Scrum und XP im harten Projektalltag

So wird Scrum bei uns gemacht



Henrik Kniberg

Vorworte von Jeff Sutherland und Mike Cohn

InfoQ Enterprise Software Development Series

KOSTENLOSE ONLINE-AUSGABE

(nicht druckbare, kostenlose Online-Ausgabe)

Falls Ihnen das Buch gefällt,
unterstützen Sie bitte den Autor und InfoQ und
bestellen eine (englische) gedruckte Ausgabe:
http://www.lulu.com/content/899349
(nur $22.95)

Präsentiert und
mit freundlicher Genehmigung von

Das Buch ist auf InfoQ.com kostenlos erhältlich.
Falls Sie es auf anderem Weg erhalten haben,
unterstützen Sie Autor und Verleger und
registrieren Sie sich bitte auf InfoQ.com.

Besuchen Sie auch die Website zu diesem Buch unter:

http://infoq.com/minibooks/scrum-xp-from-the-trenches

Scrum und XP im harten Projektalltag

So wird Scrum bei uns gemacht

von:
Henrik Kniberg

C4Media, Verleger von InfoQ.com.

Dieses Buch ist Teil der "InfoQ Enterprise Software Development" Buchreihe.

Bestellinformation zu diesem oder anderen InfoQ Büchern erhalten Sie unter books@c4media.com.

Geschäftsführende Herausgeberin: Diana Plesa
Titelbild. Dixie Press
Satz: Dixie Press

Deutsche Übersetzung von:
Robert Sösemann – robert@soesemann.com
Andreas Schliep - mail@andreas-schliep.de

Library of Congress Katalogisierungsinformation:
ISBN: 978-1-4303-2264-1

Gedruckt in den Vereinigten Staaten von Amerika

Danksagung

Den ersten Entwurf dieses Dokuments habe ich an nur einem Wochenende fertig gestellt – sicherlich ein intensives Wochenende mit einem Fokus-Faktor von 150% :o)!

Ich danke meiner Frau Sophia und unseren Kindern Dave und Jenny, die ein Wochenende meine Ungeselligkeit erduldet haben. Außerdem möchte ich mich bei Sophias Eltern Eva und Jörgen bedanken, die spontan vorbeigekommen sind, um sich um die Familie zu kümmern.

Für das Korrekturlesen und die Verbesserungsvorschläge danke ich meinen Kollegen bei Crisp in Stockholm und den Mitgliedern der *Yahoo Scrum Development* Gruppe.

Und schließlich danke ich all meinen Lesern für ihr zahl- und hilfreiches Feedback. Es freut mich außerordentlich, dass dieses Buch für so viele von ihnen Auslöser war, Agile Softwareentwicklung auszuprobieren!

Inhaltsverzeichnis

VORWORT VON JEFF SUTHERLAND **i**

VORWORT VON MIKE COHN **iii**

EINLEITUNG **7**
Hinweis **8**
Meine Motivation, ein Buch zu schreiben **8**
Was, bitte, ist Scrum?! **8**

SO SIEHT UNSER PRODUCT BACKLOG AUS **9**
Zusätzliche Story-Felder **11**
Wie wir das Product Backlog in Kundensprache halten **11**

SO BEREITEN WIR DIE SPRINT-PLANUNG VOR **13**

SO LAUFEN SPRINT-PLANUNGSMEETING AB **15**
Warum der Product Owner teilnehmen muss **15**
Warum Qualität nicht verhandelbar ist **17**
Wenn das Meeting kein Ende findet… **18**
Ablaufplan für das Sprint-Planungsmeeting **19**
Festlegen der Sprint-Länge **20**
Einigung auf ein Sprint-Ziel **21**
Auswahl der Stories für den Sprint **21**
Wie beeinflusst der Product Owner die Auswahl der Stories für den Sprint **22**
Und wie entscheidet das Team? **24**
Warum wir Karteikarten verwenden **29**
Die Bedeutung von "Erledigt" festlegen **32**
Aufwandschätzungen mit Planungspoker **33**
Unklarheiten zu Stories ausräumen **35**
Stories in mehrere Stories zerlegen **36**
Stories in Aufgaben zerlegen **36**
Termin für tägliches Scrum-Meeting finden **38**
Wann man aufhören sollte **38**
Technik-Stories **39**
Bugtracker oder Product Backlog **41**
Das Sprint-Planungsmeeting ist endlich vorbei **42**

SO MACHEN WIR SPRINTS BEKANNT **43**

SO SIEHT UNSER SPRINT BACKLOG AUS **45**
Sprint Backlog Formate **45**
So funktioniert die Aufgabenwand **47**
Beispiel 1 - nach dem ersten Scrum-Meeting **48**
Beispiel 2 – nach ein paar Tagen **49**
So funktioniert ein Burndown-Diagramm **51**
Alarmsignale auf der Aufgabenwand **52**
Wie sieht's mit der Rückverfolgbarkeit aus?! **54**
In Tagen oder Stunden schätzen **54**

SO RICHTEN WIR DEN TEAMRAUM EIN **55**

DIE ENTWURFS-ECKE **55**
Setzen Sie das Team zusammen! **57**
Halten Sie den Product Owner auf Abstand **58**
Halten Sie Manager und Berater auf Abstand **58**

SO LÄUFT DAS TÄGLICHE SCRUM-MEETING AB **61**
So aktualisieren wir die Aufgabenwand **61**
Wie gehen wir mit Nachzüglern um? **62**
Hilfe bei "Und was soll ich heute machen?" **62**

SO PRÄSENTIEREN WIR SPRINT-ERGEBNISSE **65**
Bei uns gilt: Kein Sprint ohne Vorführung **65**
Checkliste für Sprint-Vorführungen **66**
Unvorführbares vorführen **66**

WIE WIR SPRINT-RETROSPEKTIVEN DURCHFÜHREN **67**
Warum wir auf Retrospektiven bei allen Teams bestehen **67**
So organisieren wir unsere Retrospektiven **67**
Erfahrungsaustausch zwischen Teams **69**
Ändern oder nicht ändern? **70**
Beispielhafte Ergebnisse einer Retrospektive **71**

LEERLAUF ZWISCHEN SPRINTS **73**

WIE WIR RELEASES UND FESTPREISPROJEKTE PLANEN **77**
Definieren Sie Akzeptanzschwellen **77**

Wie man die wichtigsten Einträge schätzt **78**
Entwicklungsgeschwindigkeit schätzen **80**
Alles zu einem Releaseplan zusammenfügen **81**
Den Releaseplan anpassen **82**

WIE WIR SCRUM MIT XP KOMBINIEREN **83**
Programmierung in Paaren **83**
Testgetriebene Entwicklung **84**
Inkrementelle Architektur **86**
Fortlaufende Integration **87**
Gemeinsames Code- Eigentum **87**
Informativer Arbeitsbereich **88**
Programmierstandards **88**
Nachhaltiges, schwungvolles Arbeitstempo **89**

WIE WIR TESTEN **91**
Um eine Phase für Abnahmetests werden Sie wahrscheinlich nicht herumkommen **91**
Minimieren Sie die Abnahmetestphase **92**
Tester im Scrum-Team verbessern die Qualität **93**
Die Qualität verbessern, indem sie sich pro Sprint weniger vornehmen **95**
Sind Abnahmetests Teil des Sprints? **95**
Sprint-Zyklen und Abnahmetest-Phasen **96**
Überfordern Sie nicht ihr schwächstes Kettenglied **100**
Wieder zurück zur Realität **101**

WIE WIR MIT MEHREREN SCRUM-TEAMS UMGEHEN **103**
Wie viele Teams soll man aufstellen? **103**
Warum wir eine Teamleiter-Rolle eingeführt haben **107**
Wie wir Mitarbeiter in Teams aufteilen **108**
Spezialisierte Teams - oder nicht? **110**
Teams zwischen Sprints umordnen - oder nicht? **112**
Teilzeit-Teammitglieder **113**
Wie wir Scrum-of-Scrums- Meetings durchführen **114**
Tägliche Scrum-Meetings aufeinander abstimmen **116**
"Feuerwehr-Teams" **117**
Das Product Backlog aufteilen - oder nicht? **118**
Code Branching **122**
Retrospektiven mit mehreren Teams **123**

WIE WIR MIT VERTEILTEN TEAMS UMGEHEN **125**
Offshoring **126**

Heimarbeit von Teammitgliedern 127

CHECKLISTE FÜR SCRUMMASTER 129
Zu Beginn des Sprints 129
Täglich 129
Am Sprint-Ende 130

SCHLUSSWORT 131

LESE-EMPFEHLUNGEN 133

DER AUTOR 135

DIE ÜBERSETZER DER DEUTSCHEN AUSGABE 136

Vorwort von Jeff Sutherland

Teams sollten über die Grundlagen von Scrum Bescheid wissen. Beispielsweise, wie man ein Product Backlog erstellt und daraus ein Sprint Backlog ableitet. Oder wie ein Burndown-Diagramm funktioniert und man die durchschnittliche Entwicklungsgeschwindigkeit (engl. Velocity) seines Teams ausrechnet. Henriks Buch ist eine gute Starthilfe für Teams, die über die ersten Gehversuche hinaus einen professionellen Einsatz vom Scrum erreichen wollen.

Der richtige Einsatz von Scrum ist auch bei der Vergabe von Risikokapital immer wichtiger. Als Berater bei einer Risikokapitalfirma sorge ich dafür, dass nur in die Unternehmen investiert wird, die Agile Praktiken auch richtig anwenden. Unser Senior Partner befragt jede Firma in unserem Portfolio nach der Entwicklungsgeschwindigkeit ihrer Teams. Die meisten tun sich momentan mit der Antwort noch schwer. Bei der Vergabe zukünftiger Investitionen ist sie allerdings Grundvoraussetzung.

Warum aber ist das so wichtig? Wenn ein Team seine Entwicklungsgeschwindigkeit nicht kennt, kann der Product Owner auch keinen Releaseplan mit glaubwürdigen Endterminen für das Produkt erstellen. Ohne verlässliche Termine wiederum ist es sehr wahrscheinlich, dass die Firma scheitert und der Investor sein Geld verliert!

Dieser Herausforderung sehen sich große und kleine, junge und etablierte, sowie eigen- und fremdfinanzierte Unternehmen gegenüber. Kürzlich wurde auf einer Konferenz in London der Einsatz von Scrum bei Google diskutiert. Ich befragte das Publikum, wer Scrum einsetzt. Von den 135 Teilnehmern meldeten sich 30. Anschließend fragte ich, wer iterative Entwicklung nach Nokia Standard betreibt. Iterative Entwicklung ist laut dem Agilen Manifest eine Grundvoraussetzung, um schnell und häufig funktionierende Software auszuliefern.
Über Jahre hinweg, leitete Nokia aus den Ergebnissen von Scrum-Retrospektiven hunderter Teams die folgenden Regeln für iterative Entwicklung ab:

- Eine Iteration hat immer eine feste Länge und sollte stets kürzer als sechs Wochen sein.
- Am Ende jeder Iteration steht funktionierender und getesteter Code.

Lediglich die Hälfte der zuvor erwähnten 30 Personen befolgte die erste der beiden Regeln. Und davon wiederum nur drei erfüllten die Nokia Scrum-Regeln. Diese sind:

- Jedes Scrum-Team hat einen Product Owner
- Der Product Owner führt ein Product Backlog mit Schätzungen vom Team
- Das Team hat ein Burndown-Diagramm und kennt seine Entwicklungsgeschwindigkeit
- Während eines Sprints hat kein Außenstehender störenden Einfluss auf das Team

Meine Risikokapital-Partner werden in Zukunft nur in solche Teams investieren.

Teams, welche die Praxistipps aus Henriks Buch beherzigen, haben hingegen alle ein Product Backlog mit Aufwandschätzungen und ein Burndown-Diagramm. Sie kennen ihre Entwicklungsgeschwindigkeit und halten sich an andere wichtige Regeln für effektives Scrum. Außerdem befolgen sie die Scrum-Regeln von Nokia und sind damit für Investoren prinzipiell interessant. Als Startup sind sie für Risikokapitalgeber eine attraktive Investition.

Vielleicht haben diese Teams sogar das Zeug, die Zukunft der Softwarebranche mitzugestalten und die kommende Generation führender Softwareprodukte zu entwickeln.

Jeff Sutherland,
Ph.D., Mitbegründer von Scrum

Vorwort von Mike Cohn

Bei Scrum und auch beim Extreme Programming (XP) soll am Ende jeder Iteration vom Team ein konkretes, potentiell auslieferbares Stück Software produziert werden. Iterationen sind mit Absicht kurz und zeitlich begrenzt, denn das oberste Ziel ist, innerhalb kürzester Zeit funktionsfähigen Code fertig zu stellen. Scrum- und XP-Teams bleibt somit wenig Zeit fürs Theoretisieren und Feilen am ausgefeilten UML-Modell oder Pflichtenheft. Sie konzentrieren sich darauf, ihre Arbeitspakete zügig abzuarbeiten, anstatt Code zu schreiben, der alle späteren Eventualitäten abdeckt. Sie wissen nicht nur, dass dabei Fehler passieren können, sondern auch, dass derjenige bei der Fehlersuche besser vorankommt, der sich am konkreten Problem die "Hände schmutzig macht" anstatt sich in Theorien und Analysen zu verlieren.

Diese Beschränkung aufs Praktische und der Verzicht auf Theorie ist es, der dieses Buch auszeichnet. Von der ersten Seite an merkt man, wie wichtig Henrik das ist. Anstelle einer langatmigen Einführung in Scrum, belässt er es bei ein paar Links auf relevante Websites. Am Beispiel des Product Backlogs steigt er direkt in die Praxis ein und stellt alle wichtigen Artefakte und Praktiken eines gut laufenden agilen Projekts vor. Er verzichtet auf dröge Theorien, Literaturhinweise oder Fußnoten. Sein Buch gibt auch keine tiefschürfende Begründung für den Erfolg von einzelnen Praktiken oder Scrum im Allgemeinen. Gerade weil er sich darauf konzentriert von gut funktionierenden Teams zu erzählen, ist der Untertitel "So wird Scrum bei uns gemacht" so passend.

Auch wenn Scrum bei Ihnen vielleicht anders verwendet wird, können Sie viel von den Erfahrungen anderer Teams lernen. Erst recht, wenn diese zum Erfolg geführt haben.

Die Liste mit Faustregeln für Scrum gibt es nicht und wird es wohl auch nicht geben. Einfach, weil der konkrete Projektrahmen wichtiger ist als alles andere. Wir sollten uns also lieber fragen, in welchem Kontext etwas erfolgreich war, bevor wir allgemeingültige Erfolgsrezepte fordern.

Je mehr Erfahrungsberichte erfolgreicher Teams Sie lesen, desto besser sind Sie auf die Hürden vorbereitet, die Ihnen bei Scrum und XP bevorstehen.

Henrik hilft uns mit dieser guten Mischung aus praktischen Erfahrungen und Hintergrundinformationen den eigenen Projektalltag durch Scrum und XP besser zu meistern.

Mike Cohn

Autor der Bücher *Agile Estimating and Planning* und *User Stories Applied for Agile Software Development.*

Vorwort - Wow, Scrum funktioniert!

Scrum hat funktioniert! Zumindest bei uns, oder besser gesagt bei meinem aktuellen Kunden in Stockholm, dessen Name hier nichts zur Sache tut. Vielleicht hilft dieses Buch ja dabei, dass es auch bei Ihnen klappt.

Für mich war es das erste Mal, dass eine Entwicklungsmethodik (Entschuldige, Ken, ich meine natürlich ein Framework) sich praktisch vom Buch weg in die Praxis umsetzen ließ. Sozusagen echtes Plug ‘n Play. Jeder bei uns ist zufrieden damit, egal ob Entwickler, Tester oder Manager. Dank Scrum haben wir eine schwere Krise überwunden und konnten trotz wirtschaftlicher Turbulenzen und Personaleinsparungen konzentriert und beherzt weiterarbeiten.

Ich muss zugeben, dass mich das überrascht hat. Nachdem ich zuvor einige Bücher zum Thema Scrum überflogen hatte, war mein Eindruck, dass das alles zu gut klingt, um wahr zu sein. Dementsprechend skeptisch war ich am Anfang. Nach einem Jahr mit Scrum sind ich und viele meiner Kollegen außerordentlich beeindruckt. Wann immer also nichts dagegen spricht, wird Scrum in jedem neuen Projekt eingesetzt.

1

Einleitung

Sie wollen in Ihrem Unternehmen Scrum einsetzen oder tun das bereits seit einiger Zeit? Sie kennen die Grundlagen, haben Bücher zum Thema gelesen und vielleicht sogar eine Scrum-Zertifizierung absolviert? Dann kann ich Sie nur beglückwünschen.

So richtig sicher fühlen Sie sich aber trotzdem noch nicht?

Um es mit den Worten von Ken Schwaber zu sagen: Scrum ist keine Methodik, sondern nur ein Framework. Auf Deutsch heißt das in etwa, dass Scrum nicht genau vorgibt, was Sie zu tun oder zu lassen haben.

Aber keine Sorge. Von mir erfahren Sie bis ins letzte Detail, wie Scrum bei uns eingesetzt wird. Natürlich müssen Sie es nicht genauso machen. Selbst wir würden es unter anderen Umständen vermutlich anders machen.

Dass, man Scrum den eigenen Bedürfnissen anpassen kann und auch muss, ist zugleich Stärke und Crux von Scrum.

Unsere aktuelle Arbeitsweise ist das Ergebnis eines Jahres voller Experimente in einem 40-köpfigen Team. Damals ging es der Firma nicht gut. Obwohl alle im Akkord arbeiteten, hatten wir massive Qualitätsprobleme, hetzten von einem Feuerwehreinsatz zum nächsten und verfehlten ständig unsere Abgabetermine. Der Einsatz von Scrum wurde beschlossen und mein Job war es, dafür zu sorgen, dass das auch passierte. Damals war Scrum für die meisten Kollegen in der Entwicklungsabteilung nicht mehr als ein abgehobenes Modewort ohne jeden Alltagsbezug.

Es verging ein Jahr, bis Scrum in allen Unternehmensbereichen etabliert war. Wir haben mit Teamgrößen von drei bis zwölf Leuten, Sprintlängen von zwei bis sechs Wochen und den verschiedensten Definitionen des Wortes "Erledigt" herumexperimentiert. Wir haben verschieden Formate und Tools (z.B. Excel, JIRA, oder Karteikarten) für Product und Sprint Backlog sowie Strategien für Tests, Sprint-Vorführungen und die Zusammenarbeit verteilter Teams evaluiert.

Sogar die Praktiken des Extreme Programming, wie z.B. die Fortlaufende Integration, das Programmieren in Paaren, oder Testgetriebene Entwicklung haben wir auf Ihre Anwendbarkeit und Kombinierbarkeit mit Scrum hin überprüft.

Dieser Lernprozess ist längst nicht abgeschlossen. Ich bin sicher, dass wir mit jeder Sprint-Retrospektive (sofern wir sie weiterführen) hinzulernen und besser darin werden, wie man Scrum in verschiedenen Situationen gewinnbringend einsetzt.

Hinweis

Es ist nicht die Absicht dieses Buches, die einzig richtige Art, Scrum zu praktizieren, zu beschreiben,. Es will zeigen, zu welchem Ergebnis wir nach einem Jahr des Herumexperimentierens gekommen sind. Es kann also durchaus sein, dass Sie zu einem ganz anderen Ergebnis kommen.

Außerdem spiegelt es nur meine ganz persönliche Meinung wider und nicht die von Crisp oder meinen Kunden. Aus diesem Grund habe ich es vermieden, Produkt- oder Personennamen zu nennen.

Meine Motivation, ein Buch zu schreiben

Um Scrum zu lernen, habe ich die relevanten Bücher zu Scrum und agiler Softwareentwicklung gelesen, Websites und Formen durchstöbert, Ken Schwabers Zertifizierung absolviert, ihn mit Fragen bombardiert und ausführlich mit Kollegen diskutiert. Die ergiebigsten Informationsquellen sind trotz allem aber Berichte aus der Praxis, da nur sie erklären, wie man die ganzen Regeln und Verfahren im Arbeitsalltag einsetzt. Außerdem helfen Sie dabei, typische Anfängerfehler zu erkennen und zu vermeiden.

Jetzt ist es an mir, mit einem eigenen Erfahrungsbericht etwas zurückzugeben. Ich hoffe auf umfangreiches und erhellendes Feedback von all jenen, die in einer ähnlichen Situation sind.

Was, bitte, ist Scrum?!

Ach, Sie wissen noch gar nichts über Scrum? Dann werfen Sie doch zuvor einen Blick auf die folgenden Websites:

- http://agilemanifesto.org/
- http://www.mountaingoatsoftware.com/scrum
- http://www.xprogramming.com/xpmag/whatisxp.htm

Falls Sie ungeduldig sind, können Sie natürlich auch gerne sofort weiter lesen. Da die meisten Scrum-Fachbegriffe im Text erläutert werden, sollte dieses Buch auch für Sie verständlich sein.

2

So sieht unser Product Backlog aus

Das Product Backlog ist zugleich Herzstück und Ausgangspunkt jedes Scrum-Projekts. Letztlich ist es nicht mehr als eine priorisierte Liste aller Kundenanforderungen, formuliert in der Sprache des Kunden.

Wir nennen sie Stories oder Backlog-Einträge.

Jede unserer Stories hat die folgenden Attribute:

- **ID** – Eindeutiger Bezeichner, meist eine aufsteigende Zahl. So behalten wir den Überblick, auch wenn Stories später umbenannt werden.
- **Name** – Kurzer und aussagekräftiger Storytitel aus zwei bis zehn Worten. Ein gutes Beispiel ist "Überblick über eigene Transaktionen". Das ist klar genug, damit Entwickler und Product Owner wissen, worum es geht und um die Story von anderen zu unterscheiden.
- **Wichtigkeit** – Zahl, die besagt, wie wichtig dem Product Owner diese Story ist. Je höher, desto wichtiger.
  - Ich vermeide den Begriff "Priorität", weil der Wert 1 dort meist für die höchste Wichtigkeit steht. Spätestens wenn eine Story mit noch höherer Wichtigkeit auftaucht und Werte wie 0 oder -1 eingeführt werden müssen, wird es hässlich.
- **Schätzung** – ursprüngliche Schätzung des Teams, wie aufwändig die Implementierung der Story im Vergleich zu anderen Stories ist. Gemessen wird in sog. Story-Punkten, die in etwa einem idealisierten Arbeitstag entsprechen.
  - Fragen Sie das Team: "Stellt Euch vor, Ihr hättet die optimale Anzahl Entwickler (weder zu viele noch zu wenige, am besten zwei) und könntet völlig ungestört arbeiten. Nach wie vielen Tagen könnt Ihr getestete und auslieferbare Software präsentieren?". Die Antwort "Zu dritt würden wir ungefähr vier Tage benötigen." entspräche einer Schätzung von 12 Story-Punkten.

- Viel wichtiger als exakte Schätzungen einzelner Stories (z.B. alle 2-Punkte Stories dauern 2 Tage), sind die Relationen zwischen den Stories. Eine Story mit 4 Punkten sollte in ungefähr doppelt so lang dauern, wie eine mit 2 Punkten.

§ **Vorführung** – eine grobe Beschreibung, wie diese Story bei der Sprint Vorführung demonstriert werden soll. Dies entspricht einer einfachen Testspezifikation in der Art "Tu X und Y, dann sollte Z passieren."
  - Falls Sie Testgetriebene Entwicklung nutzen, kann das Feld auch für den Pseudo-Code Ihrer Abnahmetests verwendet werden.

§ **Notizen** – Feld für Zusatzinfos, Anmerkungen oder Verweise auf andere Dokumente.

Beispielhaftes Product Backlog					
ID	**Name**	**Wichtigkeit**	**Schätzung**	**Vorführung**	**Notizen**
1	Geld einzahlen	30	5	Einloggen > "Einzahlung" aufrufen> 10 € einzahlen> "Umsatz" aufrufen > Prüfen, ob um 10 € erhöht.	Sequenz-Diagramm fehlt noch. Verschlüsselung kommt erst später.
2	Überweisungs-Historie prüfen	10	8	Einloggen >, "Umsatzhistorie" aufrufen > Einzahlung vornehmen > zurück zur "Umsatzhistorie" > Auf neuen Eintrag prüfen.	Brauchen Paginierung analog zu Userlist-View zwecks DB-Performance.

Wir haben die unterschiedlichsten Felder ausprobiert, aber nur diese sechs haben sich wirklich im Alltag bewährt.

Damit mehrere Personen am Backlog arbeiten können, verwenden wir ein Excel-Dokument mit der Einstellung "Arbeitsmappe freigeben". Auch wenn der Verantwortliche der Product Owner ist, kann so jeder das

Dokument mitnutzen. Egal, ob um nur etwas nachzulesen, oder Schätzungen nachzubessern.

Aus diesem Grund speichern wir es auch nicht in der Versionskontrolle sondern auf einem Netzwerklaufwerk. So hat jeder Zugriff, ohne dass wir uns mit gesperrten Dokumenten oder Konflikten beim Zusammenführen herumärgern müssen.

Alle anderen Dokumente kommen selbstverständlich ins Repository.

Zusätzliche Story-Felder

Gelegentlich verwenden wir zusätzliche Felder, um dem Product Owner das Priorisieren der Stories zu erleichtern.

- **Bereich** – Grobe Kategorisierung der Stories, z.B. "Adminbereich" oder "Optimierung". Mit ihr kann der Product Owner auf einen Schlag alle Einträge zum Thema "Optimierung" herunter priorisieren.
- **Komponente** – Meistens sind das Checkboxen im Excel-Dokument, z.B. "Datenbank", "Server" oder "Client" mit denen das Team die im Rahmen der Story anzupassenden technischen Komponenten identifiziert. So können Stories spezialisierten Teams (z.B. dem Datenbank-Team) leichter zugeordnet werden.
- **Urheber** – So erinnert der Product Owner sich daran, wer das Feature angefragt hat und kann ihn über den Status auf dem Laufenden halten.
- **Bugtracker-ID** – Wer ähnlich wie wir ein Bugerfassungssystem verwendet, kann so den Zusammenhang von Story und Bugs dokumentieren.

Wie wir das Product Backlog in Kundensprache halten

Product Owner mit technischem Hintergrund neigen dazu, Stories technische Namen wie z.B. "EVENT-Tabelle um Datenbankindex erweitern" zu geben. Die Frage ist aber: Was interessieren Ihn Datenbankindizes? Geht es nicht um das Thema "Performance der Veranstaltungs-Suche verbessern"?

Alle sind besser damit beraten, wenn das Team entscheidet, wie es fachlichen Anforderungen des Product Owners konkret umsetzt. Denn oft sind es gerade nicht die fehlenden Datenbankindizes, die zu schlechten Antwortzeiten führen.

Meist genügen ein paar Rückfragen beim Product Owner, um seine tatsächlichen Wünsche herauszufinden. Diese fließen dann in den Namen der Story ein (hier: "Performance der Veranstaltungssuche verbessern"). Für den Realisierungsvorschlag (hier: "EVENT Tabelle um Datenbankindex erweitern") ist im Notizfeld Platz.

3

So bereiten wir die Sprint-Planung vor

Der Tag des Sprint-Planungsmeetings rückt schnell näher. Und eine Lektion haben wir in diesem Zusammenhang gelernt:

Sie lautet: Bringen Sie zuvor das Product Backlog unbedingt in eine verwertbare Form.

Keine Sorge! Das heißt nicht, dass all Ihre Schätzungen perfekt oder die Prioritäten final sein müssen. Stellen Sie aber unbedingt sicher, dass die folgenden Mindestanforderungen erfüllt sind:

- Es gibt überhaupt so etwas wie ein Product Backlog.
- Es gibt nur *ein* Product Backlog und pro Produkt auch nur *einen* Product Owner.
- Wichtige Einträge haben einen Wert im Feld "Wichtigkeit".
  - Es ist in Ordnung, wenn weniger wichtige Einträge identische Werte haben. Für sie ist während des Sprint-Planungsmeetings vermutlich sowieso keine Zeit.
  - Jede Story, die nach Meinung des Product Owners Teil des Sprints werden könnte, sollte einen eigenen, eindeutigen Wichtigkeitswert haben.
  - Die Werte dienen nur der Sortierbarkeit. Der Wert 100 bei Story A besagt also nicht, dass diese fünfmal wichtiger ist als Story B mit dem Wert 20. Bei insgesamt nur zwei Stories reicht demnach bei Story A auch der Wert 21.
  - Es ist hilfreich, zwischen den Werten Abstand zu lassen. So ist später zwischen Story A und B Platz für Story C, deren Wichtigkeit irgendwo dazwischen liegt. Und das alles, ohne unschöne Kommazahlen wie 20,5 einzuführen!
- Der Product Owner versteht jede Story. Erst recht solche, die von anderen Kollegen hinzugefügt und von ihm nur priorisiert wurden. Er muss nicht im Detail wissen, was für die Realisierung nötig ist, aber er sollte die Beweggründe für die Story kennen.

Hinweis: Es ist erlaubt, dass Außenstehende das Product Backlog um neue Stories erweitern. Die Beurteilung der Wichtigkeit ist allein die Aufgabe des Product Owners. Aufwände werden ausschließlich vom Team geschätzt.

Darüber hinaus haben wir folgende Dinge ausprobiert:

- Pflege des Product Backlogs in unserem Bug Tracker Jira: Die meisten Product Owner fanden es aber zu umständlich. Excel ist intuitiv bedienbar und so praktisch, weil man Elemente einfärben und umsortieren, schnell Spalten und Notizen hinzufügen oder Daten leicht importieren und exportieren kann.
- Verwendung von Tools für Agiles Projektmanagement, wie z.B. VersionOne, ScrumWorks, XPlanner. Eine Evaluation dieser Tools steht noch aus.

4

So laufen Sprint-Planungsmeetings ab

Das Sprint-Planungsmeeting ist ein wichtiges Meeting – wie ich finde, vielleicht sogar das wichtigste bei Scrum. Ein schlecht laufendes Sprint-Planungsmeeting kann zum Misserfolg des ganzen Sprints führen.

Sinn und Zweck des Sprint-Planungsmeetings ist es, nicht nur dem Team genug Information mitzugeben, sondern auch dem Product Owner genug Vertrauen zu geben, dass er das Team ein paar Wochen ungestört arbeiten lässt.

Um es auf den Punkt zu bringen, hier die konkreten Ziele des Meetings:

- Ein Sprint-Ziel
- Die Liste der Teammitglieder und evtl. Angaben, wie viel Ihrer Zeit sie ins Projekt einbringen können
- Ein Sprint Backlog (= Liste der im Sprint zu implementierenden Stories)
- Zeit und Treffpunkt für das tägliche Scrum-Meeting

Warum der Product Owner teilnehmen muss

Es ist ein echtes Problem, wenn der Product Owner sich sträubt, ein paar Stunden für das Sprint-Planungsmeeting mit dem Team zu opfern. Die Ausrede ist oft: "Ich kann zwar nicht an Eurem Meeting teilnehmen, aber ich habe ja bereits dokumentiert, was ich benötige."

Er und das gesamte Team müssen anwesend sein, weil jede Story aus drei Größen besteht, die stark voneinander abhängen.

Umfang und Wichtigkeit einzelner Anforderungen werden vom Product Owner festgelegt. Aufwandschätzungen hingegen stammen vom Team. Im Sprint-Planungsmeeting werden diese drei Größen in der Diskussion zwischen Team und Product Owner unaufhörlich aufeinander abgestimmt.

Normalerweise fängt das Meeting damit an, dass der Product Owner einen Überblick über das Sprint-Ziel und die wichtigsten Stories gibt. Angefangen bei der höchsten Wichtigkeit geht das Team anschließend alle Stories durch, und schätzt deren Aufwände. Dabei treten immer wieder wichtige Fragen zum Umfang einzelner Anforderungen auf. Zum Beispiel: Sollen bei der Story „Benutzeraccount löschen“ auch offene Transaktionen des Benutzers rückgängig gemacht werden?" Die Antwort wird das Team oft so überraschen, dass es seine Schätzung anpassen muss.

Ein anderes Mal fällt vielleicht der Product Owner wegen der Aufwandschätzung aus allen Wolken. Nicht selten priorisiert er die Stories dann um oder reduziert den Umfang so weit, dass die Schätzung verringert werden kann.

Diese direkte Zusammenarbeit ist elementar bei Scrum und der Agilen Softwareentwicklung im Allgemeinen.

Besteht der Product Owner weiter darauf, für das Meeting keine Zeit opfern zu wollen, wende ich meist nacheinander die folgenden drei Strategien an:

- Ich versuche ihn zu überzeugen, wie wichtig seine Teilnahme für den Erfolg ist und hoffe, dass das seine Meinung ändert.
- Ich suche einen Freiwilligen, der stellvertretend am Meeting teilnimmt. Dem Product Owner sage ich: "Nachdem Sie nicht teilnehmen können, wird Jeff Sie vertreten. Im Meeting darf er für Sie Prioritäten oder den Umfang einzelner Stories anpassen. Bitte stimmen Sie sich deshalb vorher so gut wie möglich mit ihm ab. Gerne können Sie auch selbst einen anderen Stellvertreter vorschlagen, der am gesamten Meeting teilnehmen kann.
- Ich versuche das Management zu überzeugen, einen anderen Product Owner zu bestimmen.
- Ich verschiebe das Sprint-Planungsmeeting auf einen Termin, an dem der Product Owner Zeit hat. In der Zwischenzeit gebe ich weder Zusagen zu Lieferinhalten noch zu Terminen. Das Team kann derweil tun, wozu es Lust hat.

Warum Qualität nicht verhandelbar ist

Das gezeigte Dreieck enthält aus gutem Grund nicht auch die Größe *Qualität.*

Zunächst möchte ich den Unterschiede zwischen *interner* und *externer Qualität* erklären.

- *Externe Qualität* wird vom Benutzer eines Systems wahrgenommen. Eine langsame, unintuitive Benutzeroberfläche ist ein gutes Beispiel für mangelhafte externe Qualität.
- Die *interne Qualität* wird vom Benutzer zwar meist nicht wahrgenommen, beeinflusst aber die Wartbarkeit des System enorm. Beispiele sind eine ausgewogene Systemarchitektur, gute Testabdeckung oder die Code-Lesbarkeit.

Systeme mit hoher interner Qualität können durchaus eine geringe externe Qualität haben. Systeme mit geringer interner Qualität haben aber nur selten eine hohe externe Qualität. Wohl einfach deshalb, weil es schwierig ist, auf einem brüchigen Fundament etwas wirklich Gutes zu bauen.

Externe Qualität ist bei uns Anforderungsgegenstand. Aus betriebswirtschaftlicher Sicht ist es nämlich durchaus sinnvoll, zunächst eine langsame, rudimentäre Version des Systems auszuliefern und diese später zu verbessern. Da der Product Owner den Anforderungsumfang verantwortet, sollte er auch diese Entscheidung treffen.

Die interne Qualität andererseits steht nicht zur Debatte. Nur das Team kann und muss für Wartbarkeit und Qualität des Systems gerade stehen. Insofern ist das nicht verhandelbar. Niemals.

(Gut! Zumindest *fast* nie)

Und wie unterscheidet man zwischen interner und externer Qualität?

Angenommen der Product Owner sagt: "Ich respektiere Eure Schätzung natürlich. Aber denkt Ihr nicht auch, dass man eine schnelle Lösung in der Hälfte der Zeit hinbekommt?"

Aha! Hier versucht jemand, die interne Qualität zu verhandeln. Und er tut dies, weil er will, dass Sie Ihre Schätzung reduzieren, ohne dass er dafür Funktionalität opfern muss. Das Wort "schnelle Lösung" sollte bei Ihnen die Alarmglocken klingen lassen.

Warum sind wir hier so restriktiv?

Meine Erfahrung zeigt mir, dass es eigentlich immer eine schlechte Idee ist, die interne Qualität zu opfern. Die langfristigen Kosten heben die anfängliche Zeitersparnis mehr als auf. Toleriert man erst einmal eine schlechtere Codequalität, ist diese auch später nur schwer wieder herstellbar.

Besser ist es, den Anforderungsumfang zu diskutieren und zu sagen "Wenn es wichtig ist, das Feature schnell auszuliefern, wie wäre es dann mit einer Reduzierung des Umfang, damit wir schneller fertig sind? Eine Möglichkeit wäre z.B. eine vereinfachte Fehlerbehandlung. Die Story "Erweitere Fehlerbehandlung" liefern wir später nach. Oder wir priorisieren andere Stories herunter, um uns auf diese konzentrieren zu können?"

Merkt der Product Owner erst einmal, dass die interne Qualität nicht verhandelbar ist, lernt er oft, recht geschickt mit den anderen Größen zu jonglieren.

Wenn das Meeting kein Ende findet...

Am schwierigsten sind Sprint-Planungsmeetings, bei denen zwar
1) alle glauben, das Meeting schnell abschließen zu können
2) ... und es dann doch nicht tun!

Alles bei Scrum hat feste Anfangs- und Endzeiten, so genannte Zeitboxen. Wir versuchen diese – wie ich finde – wunderbare und einfache Regel auch zu beherzigen.

Aber was tun Sie, wenn sich das Sprint-Planungsmeeting seinem Ende zuneigt und man sich weder auf ein Sprint-Ziel noch auf ein Sprint Backlog geeinigt hat? Hört man dann einfach auf? Verlängert man um eine weitere Stunde oder macht am nächsten Tag ein Folgemeeting?

Gerade bei neuen Teams passiert das gar nicht so selten. Ich weiß nicht, was Sie machen, aber wir brechen das Meeting abrupt ab. Schluss! Aus! Dann leidet eben der Sprint! Meist sage ich dem Team: "Das Meeting ist in zehn Minuten vorbei und wir haben nicht wirklich viel geplant. Lassen wir es damit bewenden oder machen wir morgen früh zwischen acht und zwölf noch mal ein Folgemeeting?" Die Antwort ist nicht schwer zu erraten, oder?

Ich habe auch ausprobiert, in solchen Fällen das Meeting einfach weiterlaufen zu lassen. Weil die Teilnehmer aber schon müde sind, führt das nur selten zum Ergebnis.

Wer in zwei bis acht Stunden keinen halbwegs passablen Plan produziert, schafft das auch nicht in einer zusätzlichen Stunde. Am Folgetag ein weiteres Meeting einzuberufen, ist eigentlich keine schlechte Lösung. Nur will das Team oft endlich loslegen und ist einfach zu ungeduldig für ein weiteres stundenlanges Planungsmeeting.

Aus diesem Grund breche ich solche Meetings einfach ab. Selbstverständlich leidet der Sprint darunter. Das Team hat seine Lektion aber gelernt und wird beim nächsten Mal deutlich effizienter arbeiten. All jenen, denen das erste Meeting schon nicht kurz genug sein konnte, werden sich bei der nächsten Einladung sicher weniger wehren.

Lernen Sie, sich an Ihre Zeitvorgaben zu halten, aber auch Zeitvorgaben realistisch zu setzen. Das gilt für die Länge von Meetings genauso wie für die Länge von Sprints.

Ablaufplan für das Sprint-Planungsmeeting

Das Risiko über die eingeplante Zeit hinauszuschießen, lässt sich durch einen groben Ablaufplan deutlich minimieren.

Hier ein beispielhafter Ablaufplan von uns:

Sprint-Planungsmeeting: 13:00 – 17:00 (Pro Stunde 10 Minuten Pause)

- **13:00 – 13:30.** Vorstellung des Sprint-Ziels und Zusammenfassung des Product Backlogs durch den Product Owner. Einigung auf Ort und Termin für die Sprint-Vorführung.
- **13:30 – 15:00**. Aufwandschätzung und eventuelle Zerlegung der Stories. Eventuelle Anpassungen der Prioritäten durch den Product Owner. Besprechung unklarer Stories. Einigung, welche Stories vorgeführt werden sollen.
- **15:00 – 16:00**. Auswahl der Stories für den Sprint durch das Team. Anhand der berechneten Entwicklungsgeschwindigkeit die Machbarkeit prüfen.
- **16:00 – 17:00**. Einigung auf Uhrzeit und Treffpunkt für das tägliche Scrum-Meeting (falls abweichend vom letzten Sprint). Ableiten der Arbeitspakete aus den Stories.

Die Zeiten sind lediglich als Vorschlag zu verstehen. Der ScrumMaster kann die einzelnen Abschnitte während des Meetings je nach Bedarf verlängern oder verkürzen.

Festlegen der Sprint-Länge

Da ein Ergebnis des Sprint-Planungsmeetings der Termin für die Sprint-Vorführung ist, muss man sich ebenfalls auf eine Sprint-Länge festlegen.

Und wie sollte diese idealerweise gewählt werden?

Kurze Sprints sind gut, weil sie dem Unternehmen erlauben "agil" zu sein und flexibel seine Ausrichtung zu ändern. Denn je kürzer die einzelnen Sprints sind, desto schneller kann man ausliefern und erhält damit schneller Feedback vom Kunden. Anstatt lange in die falsche Richtung zu laufen, lernt man früh dazu und kann entsprechen nachbessern.

Auf der anderen Seite sind auch lange Sprints gut. Sie lassen dem Team mehr Zeit sich einzuspielen oder um Verzögerungen wettzumachen - und das, ohne dabei gleich das Sprint-Ziel zu gefährden. Außerdem verbringt man pro Sprint prozentual weniger Zeit mit Meetings und Vorführungen.

Generell bevorzugen Product Owner kurze und Entwickler lange Sprints. Die Länge ist also ein Kompromiss. Nach einigen Experimenten bevorzugen wir mittlerweile eine Länge von drei Wochen. Die meisten Teams arbeiten demnach in dreiwöchigen Sprints. Das gibt uns einerseits ausreichend unternehmerische Agilität und andererseits Zeit „in Fahrt“ zu kommen und uns von Problemen zu erholen.

Wir sind zum Schluss gekommen, dass es am Besten ist, einfach verschiedene Sprint-Längen auszuprobieren. Opfern Sie keine Zeit für großartige Analysen. Wählen Sie stattdessen einfach eine halbwegs passable Länge, verwenden sie für ein bis zwei Sprints und ändern sie sie dann wieder ab.

Haben Sie so eine gut Länge ermittelt, bleiben Sie eine Weile dabei. Bei uns ist so die Länge von 3 Wochen herausgekommen, an die wir uns fast immer halten. Manchmal ist das auch zu lang, ein anderes Mal etwas zu kurz. Dadurch, dass es immer drei Wochen sind, ist so etwas wie ein unternehmensweiter Rhythmus entstanden, an den sich alle gewöhnt haben. Weil jeder weiß, dass wir alle drei Wochen ein Release machen, gibt es auch keine Termindiskussionen mehr.

Einigung auf ein Sprint-Ziel

Im Sprint-Planungsmeeting kommt es immer wieder vor, dass ich mit der Frage nach dem Sprint-Ziel nur fragende Blicke vom Team und einem etwas ratlosen Product Owner ernte.

Aus unerfindlichen Gründen ist es schwierig, so ein Ziel zu benennen. Aber es lohnt sich wirklich. Lieber ein schlechtes Ziel, als gar kein Ziel. Ein Ziel könnte z. B. sein, den Umsatz zu steigern, die drei dringlichsten Stories fertig zu stellen, den Geschäftsführer zu beeindrucken, das System für einen Beta-Test fertig zu machen oder rudimentäre Backend-Funktionalität einzubauen. Damit das Ziel auch für Außenstehende nachvollziehbar wird, sollten Sie auf technische Begriffe verzichten und in der Business-Sprache bleiben.

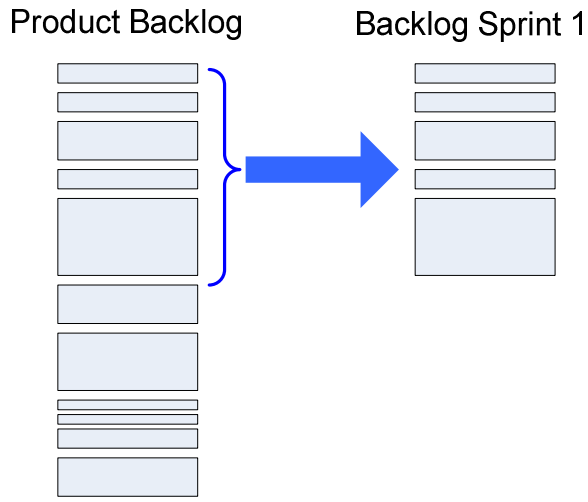
Letztlich muss das Sprint-Ziel eine Begründung liefern, warum ein Sprint überhaupt gemacht wird und nicht lieber jeder in den Urlaub geht. Oft ist es genau diese Frage, mit der man dem Product Owner ein Sprint-Ziel entlockt.

Es sollte etwas sein, was noch nicht erreicht ist. Das Ziel, den Geschäftsführer zu beeindrucken, eignet sich nur, wenn dieser nicht schon vom aktuellen System begeistert ist. Andernfalls ist das Ziel bereits erreicht und alle können nach Hause gehen.
Selbst wenn allen das Sprint-Ziel während der Planung albern und aus der Luft gegriffen vorkommt, ist es bei späteren Richtungsentscheidungen sehr hilfreich.

Wenn auch bei Ihnen mehrere Scrum-Teams an verschiedenen Produkten arbeiten, ist es hilfreich, die Sprintziele der Teams an einem für allgemein zugänglichen Platz (z.B. dem Unternehmens-Wiki) zu veröffentlichen. So wissen nicht nur das Management, sondern alle woran gerade gearbeitet wird.

Auswahl der Stories für den Sprint

Eine der Hauptaufgaben des Sprint-Planungsmeetings ist zu entscheiden, welche Stories im Sprint umgesetzt oder, genauer gesagt, aus dem Product Backlog in das Sprint Backlog übernommen werden sollen.



Die Rechtecke im oben stehenden Bild symbolisieren nach Wichtigkeit sortierte Stories. Die wichtigste Story findet sich ganz oben. Die Größe eines Rechtecks zeigt den Umfang der Story (z.B. in Story-Punkten). Die blaue Klammer steht für die geschätzte Velocity des Teams, also der Anzahl an Story-Punkten, die das Team innerhalb des Sprints glaubt, abarbeiten zu können.

Das Sprint Backlog ist eine Art Momentaufnahme des Product Backlog, enthält aber nur Stories, für die das Team zugesagt hat, sie im Rahmen des Sprints zu bearbeiten.

Nur das Team kann entscheiden, welche Stories Teil eines Sprints werden, weder der Product Owner noch sonst jemand anderes.

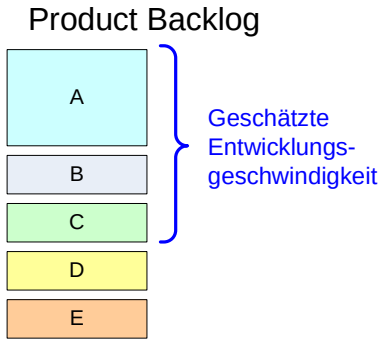
Das wirft zwei Fragen auf:

1. Nach welchen Kriterien trifft das Team seine Entscheidung?
2. Wie kann der Product Owner die Auswahl beeinflussen?

Ich möchte zunächst die zweite Frage beantworten.

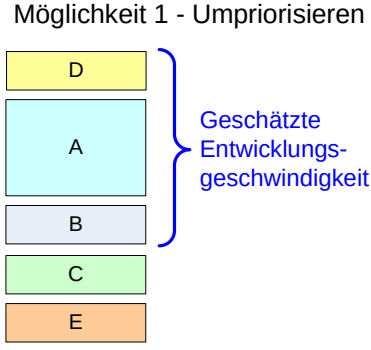
Wie beeinflusst der Product Owner die Auswahl der Stories für den Sprint

Nehmen wir an, während des Sprint-Planungsmeetings ergibt sich die folgende Situation:

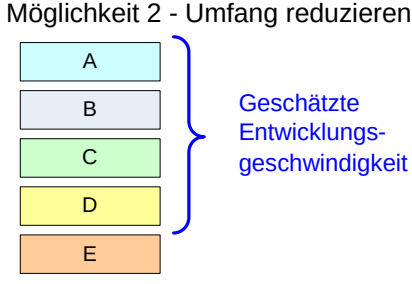


Der Product Owner ist enttäuscht, dass Story D nicht in den Sprint Backlog aufgenommen wurde. Wie kann er diese Entscheidung noch beeinflussen?

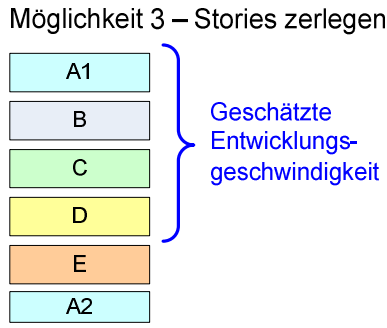
Eine Möglichkeit ist, die Priorisierung zu ändern und Story D die höchste Wichtigkeit zu geben. In diesem Fall müsste das Team sie in den Sprint aufnehmen und stattdessen Story C entfernen.



Alternativ lässt sich auch der Anforderungsumfang reduzieren. Im Beispiel ließe sich Story A so vereinfachen, bis das Team meint, auch sie noch unterzubekommen.



Eine dritte Möglichkeit ist es, Stories zu zerlegen. Wenn der Product Owner einige Aspekte der Story A für weniger wichtig hält als andere, kann er sie in Story A1 und A2 zerlegen und beiden verschiedene Wichtigkeitswerte geben.



Wie man sieht, hat der Product Owner bei der Frage, welche Stories Teil des Sprints werden, zwar keine direkte Entscheidungsgewalt, aber durchaus Einflussmöglichkeiten.

Und wie entscheidet das Team?

Wir entscheiden entweder:

1. nach Bauchgefühl oder
2. durch Abschätzung unserer Entwicklungsgeschwindigkeit

Aufwandschätzungen nach Bauchgefühl

- **ScrumMaster (zeigt auf den wichtigsten Eintrag im Product Backlog):** „Also Jungs, können wir Story A im kommenden Sprint abschließen?“
- **Lisa**: „Ja, na klar! Wir haben drei Wochen und die Story beschreibt ein ziemlich triviales Feature.“
- **ScrumMaster (zeigt auf die zweitwichtigste Story):** “Gut, wie wäre es, wenn wir Story B dazunehmen?“
- **Tom und Lisa gemeinsam**: “Immer noch Null Problemo!”
- **ScrumMaster**: “Prima. Wie sieht's aus, wenn noch Story C dazu kommt?”
- **Sam (zum Product Owner)**: “Ist hier eine ausgefeilte Fehlerbehandlung nötig?”
- **Product Owner**: “Nein, jetzt noch nicht. Momentan reicht mir rudimentäre Fehlerbehandlung.”
- **Sam**: “Dann sollte C auch machbar sein.”
- **ScrumMaster**: “OK, passt auch Story D rein??”
- **Lisa**: “Hmm....”

- **Tom**: “Ich glaube ehrlich gesagt ja!”
- **ScrumMaster**: “Mit einer Sicherheit von 90% oder eher 50%?”
- **Lisa und Tom**: “Nahe an 90%”.
- **ScrumMaster**: “Dann kommt D noch rein. Wie sieht's mit Story E aus?”
- **Sam**: “Vielleicht.”
- **ScrumMaster**: “90%? 50%?”
- **Sam**: “Ich denke, eher 50%”.
- **Lisa**: “Ich habe da meine Zweifel.”
- **ScrumMaster**: “Na gut, dann lassen wir E raus. Wir legen uns verbindlich auf A, B, C und D fest. Selbst wenn wir E schaffen könnten, soll sich niemand darauf verlassen. Insofern gehört es auch nicht in unseren Sprint-Plan. Einverstanden?”
- **Alle zusammen**: “Absolut!”

Vor allem bei kleinen Teams und kurzen Sprints funktionieren Aufwandschätzungen nach Bauchgefühl gut.

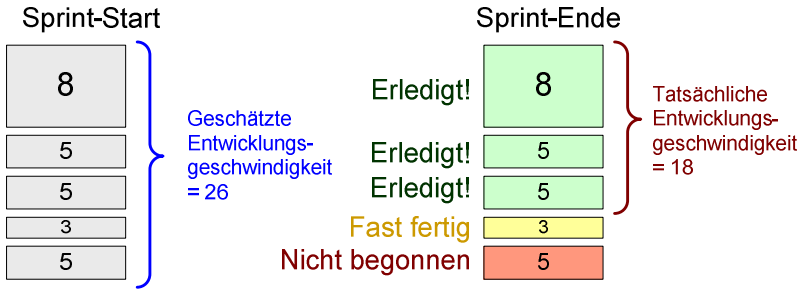
Schätzungen über die Entwicklungsgeschwindigkeit

Diese Technik hat zwei Schritte:

1. Wählen Sie eine geeignete Entwicklungsgeschwindigkeit
2. Errechnen Sie, wie viele Stories mit ihr möglich sind.

Die Entwicklungsgeschwindigkeit ist ein Maß für die Menge an erledigter Arbeit. Dabei wird von der ursprünglichen Aufwandschätzung jedes Arbeitspakets ausgegangen.

Das folgende Beispiel, zeigt wie zu Beginn und Ende des Sprints jeweils die angenommene und tatsächliche Entwicklungsgeschwindigkeit berechnet wird. Jedes Rechteck stellt eine Story dar, wobei die Zahl den ursprünglich geschätzten Aufwand angibt.



Interessant ist dabei, dass sich die tatsächliche Entwicklungsgeschwindigkeit nur aus den anfangs geschätzten Story-Aufwänden errechnet. Spätere Korrekturen an der Schätzung werden nicht miteinbezogen.

Ich höre Sie schon sagen: "Was soll das denn? Ob man eine hohe oder niedrige Entwicklungsgeschwindigkeit hat, hängt dann ja von hunderten Faktoren ab: von dusseligen Entwicklern, falschen Schätzungen, nachträglichen Anforderungen, Störungen während des Sprints."

Sie haben Recht. Es ist nur eine vage Zahl, aber die ist besser als gar keine Zahl. Außerdem liefert sie harte Fakten und zeigt, wie viel Sie, im Gegensatz zu dem was Sie sich vorgenommen haben, wirklich geschafft haben. Gründe für die Abweichung werden nicht betrachtet.

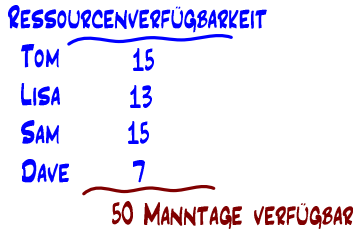
Und was ist mit Stories, die nur fast abgeschlossen sind? Kann man nicht einen Teil Ihrer Punkte in die Berechnung der Entwicklungsgeschwindigkeit mit einfließen lassen? Nein, kann man nicht! Scrum sowie andere Vorgehensweise aus dem Umfeld der Agilen Softwareentwicklung und des so genannten Lean Manufacturing legen großen Wert darauf, Dinge so zu beenden, so dass sie auslieferbar und wirklich erledigt sind. Der Wert halbfertiger Arbeitspakete liegt bei Null; wenn nicht gar im negativen Bereich. Wenn Sie mehr zum Thema wissen wollen, empfehle ich *Managing the Design Factory* von Donald Reinertsen oder eines der Bücher der Poppendiecks zu lesen.

Mit welchen magischen Formeln schätzen wir aber unsere Entwicklungsgeschwindigkeit?

Zu ihrer Abschätzung hilft ein Blick in die nähere Projektvergangenheit. Lag sie dort immer ungefähr bei X, gilt das vermutlich auch für den nächsten Sprint.

Dieses Prinzip nennt man "das Wetter von gestern". Es funktioniert nur dann, wenn das Team schon einige Sprints zusammen gearbeitet hat und sich Arbeitsbedingungen wie die Teamgröße nicht ständig ändern. Das ist natürlich nicht immer der Fall.

Im Detail läuft die Berechnung über die Ressourcenverfügbarkeit. Angenommen wir planen einen dreiwöchigen Sprint (entspricht 15 Arbeitstagen) für ein vierköpfiges Team. Wenn Lisa davon zwei Tage und Dave (generell nur zu 50% verfügbar) einen Tag Urlaub hat, ergibt das...



…für den Sprint 50 verfügbare Manntage.

Das ist aber noch nicht die geschätzte Entwicklungsgeschwindigkeit, weil wir den Aufwand in Story-Punkten geschätzt haben. Diese gleichen aber idealisierten, optimal effektiven Arbeitstagen ohne jede Störung. Weil solche Arbeitstage äußerst selten sind, und wir außerdem unerwartete Zusatzarbeit, Ausfälle wegen Krankheit und ähnliches berücksichtigen müssen, ist die geschätzte Entwicklungsgeschwindigkeit niedriger als 50.

Um wie viel niedriger, besagt der sog. Fokus-Faktor.

Dieser gibt an, wie konzentriert das Team arbeiten kann. Rechnen Sie mit häufigen Unterbrechungen oder damit, dass ihre Schätzungen zu optimistisch waren, ist der Wert entsprechend niedriger.

Anhand der Entwicklungsgeschwindigkeit des letzten Sprints - oder dem Durchschnitt der letzten Sprints - lässt sich leicht ein sinnvoller Wert für die kommenden Sprints ableiten.

Die tatsächliche Entwicklungsgeschwindigkeit erhält man, indem man die ursprünglichen Schätzungen aller Stories zusammenzählt, die im letzten Sprint erledigt wurden.

Angenommen das Team von Tom, Lisa und Sam hat innerhalb von drei Wochen (=45 Manntagen) 18 Story-Punkte abgeschlossen. Um es etwas schwieriger zu machen, erweitern wir das Team im nächsten Sprint um

einen weiteren Kollegen namens Dave. Unter Berücksichtigung der Urlaube erhalten wir 50 Manntage für den kommenden Sprint.

Mit einer geschätzten Entwicklungsgeschwindigkeit von 20 Story-Punkten für den kommenden Sprint kann das Team Stories im Wert von ungefähr 20 Story-Punkten hinzufügen.

Das Team kann sich hier entweder für die ersten vier (=19 Story-Punkte) oder ersten fünf Stories (=25 Story-Punkte) entscheiden.

Angenommen sie entscheiden sich für die ersten vier, weil das der 20 am nächsten kommt, errechnet sich die geschätzte Entwicklungs-Geschwindigkeit aus der Summe der vier Stories, also 19 Story-Punkte.

So praktisch der Trick mit dem "Wetter von gestern" auch ist, er ersetzt nicht den gesunden Menschenverstand. Lief der letzte Sprint etwa ungewöhnlich schlecht weil das halbe Team Tage krank war, gehen Sie von einem Einzelfall aus und wählen einen höheren Fokus-Faktor. Hat das Team zusätzlich ein neues blitzschnelles Build-System installiert, erhöhen Sie den Fokus-Faktor weiter. Verringern sollten Sie den Fokus-Faktor dann, wenn beispielsweise für das Einarbeiten neuer Mitarbeiter Zeit investiert werden muss.

Wann immer es möglich ist, sollten Sie ihre Schätzungen anhand früherer Erfahrungen absichern.

Nutzen Sie Erfahrungen anderer Teams, wenn Sie selbst noch nicht auf eine gemeinsame Projekthistorie zurückblicken können.

Wenn es keine anderen Teams zum „Abschauen“ gibt, hilft nur raten. Spätestens nach dem ersten Sprint haben sie ausreichend Erfahrungswerte und können Fokus-Faktor und Entwicklungsgeschwindigkeit kontinuierlich prüfen und nachbessern.

Weil sich über die Zeit in unseren Teams ein Fokus-Faktor von 70% herauskristallisiert hat, verwenden wir diesen auch bei neuen Teams.

So machen wir Aufwandschätzungen

Ich habe bereits verschiedene Techniken genannt: Schätzungen nach Bauchgefühl, auf Basis von Erfahrungswerten oder solche auf Basis von Annahmen der verfügbaren Manntage und des Fokus-Faktors.

Und welches Verfahren verwenden wir nun?

Normalerweise kombinieren wir sie, was nicht mal lange dauert.
Wir prüfen Fokus-Faktor und Entwicklungsgeschwindigkeit des vergangenen Sprints und zusätzlich die Ressourcenverfügbarkeit und den Fokus-Faktor des kommenden Sprints. Relevante Unterschiede werden diskutiert und die Werte entsprechend angepasst.

Ist die Liste mit den Stories für den Sprint halbwegs final, führe ich meist einen "Bauchgefühl-Test" durch. Ich bitte das Team, für einen Moment die Schätzungen außer Acht zu lassen und sich zu fragen, ob das alles in diesem Sprint geschafft werden kann. Je nach Antwort nehmen wir ein bis zwei Stories von der Liste, oder fügen weitere hinzu.

Fokus-Faktor, Ressourcenverfügbarkeit und die geschätzte Entwicklungs-Geschwindigkeit sind letztlich aber nur Mittel, um zu entscheiden, welche Stories Teil des Sprints werden.

Warum wir Karteikarten verwenden

Der größte Teil des Sprint-Planungsmeetings befasst sich mit den Stories im Product Backlog. Sie werden geschätzt, priorisiert, detailliert und in weitere Stories zerlegt.

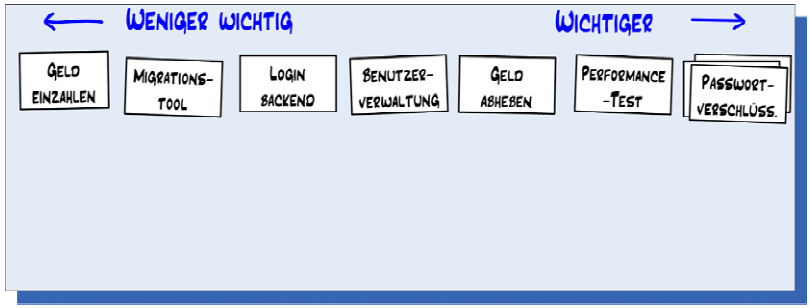
Wie läuft das konkret ab?

Nun, normalerweise wirft das Team den Beamer an und projiziert die Excel-Tabelle mit dem Backlog an die Wand. Ein Teilnehmer (meistens der Product Owner oder der ScrumMaster) schnappt sich die Tastatur, liest die einzelnen Stories vor und lädt zur Diskussion ein.

Sobald das Team sich über Details und Wichtigkeit verständigt hat, überträgt er die Werte in die Exceltabelle.

Klingt gut, oder? Nein, nicht wirklich! Wie schlecht diese Arbeitsweise eigentlich ist, merkt das Team oft erst am Ende des Meetings, dann wenn die Liste immer noch nicht abgeschlossen ist.

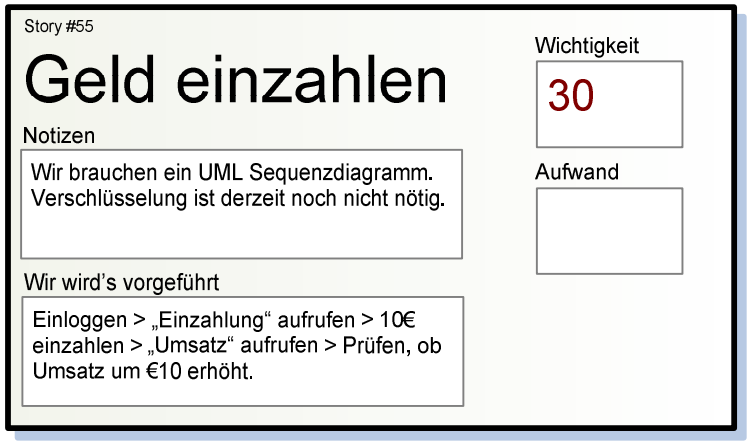
Wesentlich praktischer ist es, mit Karteikarten an einer Wand oder auf einem großen Tisch zu arbeiten.



Das ist der Arbeit mit Computer und Beamer weit überlegen, weil:

- alle stehen und herumlaufen. Das hält munter und ist gut für die Aufmerksamkeit.
- jeder persönlich involviert ist - nicht nur derjenige an der Tastatur.
- gleichzeitig mehrere Stories bearbeitet werden können.
- es zum Ändern der Priorität reicht, Karten zu verschieben.
- die Karteikarten nach dem Meeting auf der Aufgabenwand weitergenutzt werden können (siehe dazu Abschnitt “So sieht unser Sprint Backlog aus“).

Die Karten beschreiben Sie entweder mit der Hand oder erzeugen sie sich automatisch mit einem kleinen Skript aus der Excel-Tabelle.



PS: Auf meinem Blog unter http://blog.crisp.se/henrikkniberg kann man so ein Skript herunterladen.

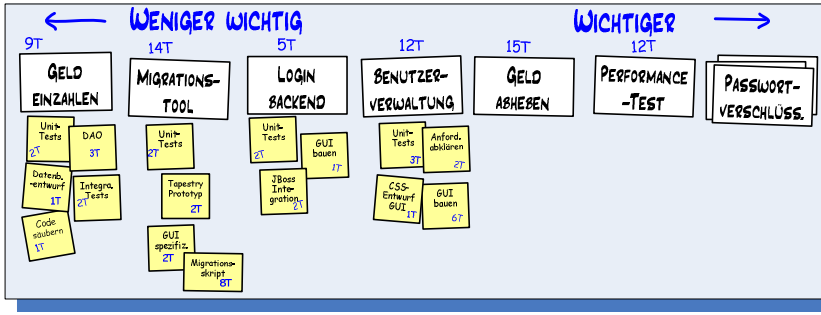
Wichtig: Nach dem Sprint-Planungsmeeting überträgt unser ScrumMaster die Änderungen vom Papier in die Excel-Tabelle. Selbstverständlich ist das ein Zusatzaufwand. Wenn man bedenkt, wie viel effizienter ein Meeting mit physischen Karteikarten abläuft, nimmt man das aber gern in Kauf.

Noch ein Hinweis: Wir übertragen den Wichtigkeitswert nur deshalb von Excel auf die Karten, weil es das Aufhängen der Karten an der Wand einfacher macht. Hängen die Karten aber erst einmal, verliert er seine Bedeutung und wir arbeiten nur noch mit der Reihenfolge der Karten. Die wichtigsten Karten hängen links, die weniger wichtigen. Wenn der Product Owner die Reihenfolge später ändert, passen wir die Zahl auf der Karte nicht an.

Wichtig ist es aber, die im Meeting geänderten Prioritäten in Excel nachzuziehen.

Der Aufwand einer Story lässt sich oft leichter und genauer schätzen, wenn man sie zunächst in Ihre Teilaufgaben zerlegt. (Wir sagen "Aufgabe", weil das englische Wort für "Tätigkeit" (task) auf Schwedisch eine völlig andere Bedeutung hat :0).)

Auch das Zerlegen lässt sich mit Karteikarten gut erledigen, indem sich das Team in Kleingruppen aufteilt und mehrere Stories parallel abarbeitet. Die so gewonnenen Aktivitäten schreiben wir auf gelbe Klebezettel und kleben sie unter die jeweilige Storykarte.



Wir übertragen die Aufgaben aus zwei Gründen nicht wieder nach Excel:

- Es handelt sich nur um eine vorläufige Zerlegung, die sich während des Sprints häufig ändern kann. Sie immer aktuell zu halten, ist schlichtweg zu aufwändig.
- Der Product Owner hat kein Interesse an dieser Detailtiefe.

Genau wie die Story-Karteikarten können auch die Aufgaben-Zettel für das Sprint Backlog weitergenutzt werden (siehe dazu den Abschnitt "So sieht unser Sprint Backlog aus.“).

Die Bedeutung von "Erledigt" festlegen

Product Owner und Teams sollten sich unbedingt auf eine klare Definition von "Erledigt" einigen. Ist eine Story bereits erledigt, wenn der Code eingecheckt ist, oder erst dann, wenn er auf der Testumgebung installiert und geprüft wurde? Wann immer möglich bedeutet „Erledigt“ bei uns, dass etwas bereit für die Übernahme in die Produktivumgebung ist. In Ausnahmen bedeutet es aber auch nur, dass etwas auf der Testumgebung installiert wurde und bereit für Abnahmetest ist.

Während wir früher für die Frage, ob etwas erledigt ist, eine detaillierte Checkliste hatten, genügt uns heute eine Freigabe durch den Tester. Er muss dafür sorgen, dass das Team die Vorstellung des Produkt Owner versteht und die Arbeit in dem Grad abschließt, wie es unsere Definition von "erledigt" vorsieht.

Uns ist klar geworden, dass dabei nicht alle Stories gleich behandelt werden können. Gut sieht man den Unterschied an den Stories "Formular für Benutzersuche" und "Betriebshandbuch erstellen". Letztere ist schon erledigt, wenn die Betriebsabteilung Ihr OK gibt.

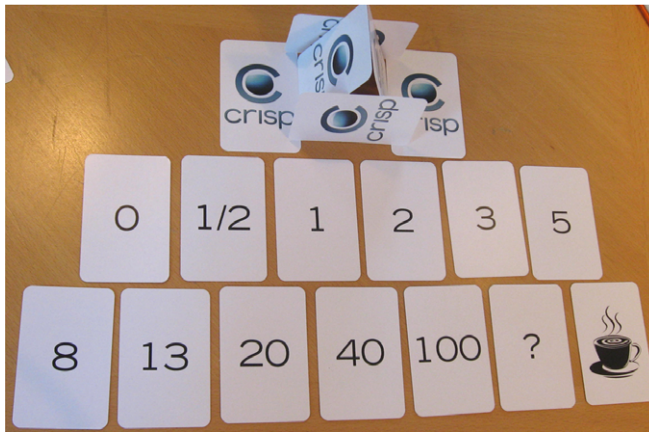
Sie sollten Ihre Stories um das Feld "Erledigt, sobald..." erweitern, wenn es bei der Frage, wann etwas erledigt ist, öfter Missverständnisse gibt.

Aufwandschätzungen mit Planungspoker

Aufwände schätzen ist eine Gruppenanstrengung des ganzen Teams. Jeder ist beteiligt, weil:

- zum Zeitpunkt der Planung oft nicht feststeht, wer genau bestimmte Stories umsetzt.
- mehrere Personen und Fähigkeiten (Oberflächendesign, Programmierung, Test usw.) zur Fertigstellung nötig sind.
- für die Abgabe einer Schätzung jeder in etwa verstehen muss, worum es sich bei einer Story handelt. Da alle an der Schätzung jeder Story beteiligt sind, haben auch alle ein Grundverständnis von ihr. Dass das Team dabei früher wichtige Fragen klärt und sich so im Sprint besser gegenseitig helfen kann, ist ein angenehmer Nebeneffekt.
- Manchmal weichen die Schätzungen einzelner Personen deutlich voneinander ab. Je eher man das bemerkt, desto besser.

Das Teammitglied mit dem besten Verständnis einer Story platzt meist auch zuerst mit einer Schätzung heraus und beeinflusst damit leider auch die der Anderen. Um das zu verhindern, gibt es Planungspoker - eine Idee, die, wie ich glaube, von Mike Cohn stammt.



Beim Planungspoker bekommt jeder Kollege die 13 oben gezeigten Karten. Soll eine Story geschätzt werden, wählt jeder die Karte aus, die der eigenen Aufwandschätzung am nächsten kommt und legt sie verdeckt auf den Tisch. Haben das alle getan, werden die Karten gleichzeitig aufgedeckt. So kann niemand die Schätzungen der anderen übernehmen, sondern muss sich selbst etwas überlegen.

Das Team diskutiert größere Unterschiede zwischen Schätzungen und überlegt gegebenenfalls, was für die Umsetzung im Detail zu tun ist. Dazu wird eine Story eventuell in Aktivitäten zerlegt und erneut geschätzt. Das wird so lange wiederholt, bis die Einzelschätzungen annähernd gleich groß sind.

Machen Sie dem Team klar, dass jede Schätzung nicht nur die eigenen, sondern alle Aufwände enthalten muss, die für die Story nötig sind. Ein Tester darf also nicht nur seine Test-Aufwände schätzen.

Sie haben sich vielleicht gefragt, warum die Zahlen auf den Karten so unregelmäßig sind und es beispielsweise zwischen 40 und 100 keinen Wert gibt Das ist so, um bei hohen Schätzungen kein trügerisches Gefühl von Genauigkeit zu vermitteln. Es ist einfach unsinnig zu diskutieren, ob eine Story 20, 18 oder 21 Tage benötigt. Jeder weiß, dass bei so großen Stories Schätzungen nur äußerst ungenau sind.

Sie wollen die Genauigkeit noch verbessern? Dann zerlegen Sie große Stories in mehrere kleine und schätzen dann diese.

Übrigens: Es ist nicht erlaubt, Karten zu kombinieren. Der Aufwand 7 ist auch nicht durch Kombination der Karten 5 und 2 möglich. Entweder Sie wählen 5 oder 8.

Einige der Karten sind erklärungsbedürftig:

- 0 bedeutet, dass eine Story bereits erledigt ist oder minimalen oder keinen Aufwand erfordert.
- ? = “ Ich habe keine blassen Schimmer!”
- Kaffeetasse = “Ich kann nicht mehr denken. Lasst uns eine Pause machen!"

Unklarheiten zu Stories ausräumen

Schlimm ist es, wenn das Team bei der Vorführung am Sprint-Ende stolz ein neues Feature vorstellt und der Product Owner sagt: "Sehr schön. Aber leider ist das nicht das, was ich gefordert habe!"
Wie stellt man sicher, dass Team und Product Owner bzw. das Team untereinander das gleiche Verständnis vom Umfang einer Story haben? Oder, dass alle im Team untereinander dasselbe Verständnis haben?
Auch wenn es dafür keine Garantie gibt, lassen sich mit ein paar Tricks doch die größten Missverständnisse auflösen. Zum Beispiel, indem man dafür sorgt, dass die Felder aller Sprint-relevanten Stories ausgefüllt sind.

Beispiel 1:
Gerade als das Team und der Product Owner zufrieden mit ihrem Plan das Meeting beenden wollen, meldet sich der ScrumMaster zu Wort: "Moment mal! Die Story 'Benutzer anlegen' hat noch keine Aufwandschätzung. Lasst sie uns bitte noch schätzen.” Nach ein paar Runden Planungspoker schockiert das Team den Product Owner mit der Schätzung von 20 Story-Punkten. Nach kurzer, hitziger Diskussion stellt sich heraus, dass das Team zum Umfang der Story eine hübsche Oberfläche zum Pflege von Benutzerkonten gezählt hat. Der Product Owner aber wollte lediglich per SQL Benutzer in die Datenbank schreiben. Die neue Schätzung beträgt nur noch 5 Punkte.

Beispiel 2:
Wieder wollen Team und Product Owner das Meeting beenden, als der ScrumMaster einwirft: "Eine Sekunde, wie wird eigentlich die Story "Benutzer anlegen" vorgeführt?" Als sich nach kurzen Gemurmel jemand zu Wort meldet und vorschlägt "Also, zuerst loggt man sich auf der Website ein, dann..." unterbricht ihn der Product Owner: "Auf der Website einloggen?! Nein, bloß nicht! Diese Funktion ist nicht Bestandteil der Website. Ein simples SQL-Skript für Administratoren genügt völlig."

Um das Sprint-Planungsmeeting pünktlich beenden zu können, fassen Sie sich mit der Beschreibung der Vorführung kurz.

Beschreiben Sie mit einfachen Worten, wie ein typisches Testszenario aussieht (so in der Art: "Tue dies, dann das und prüfe ob das passiert.")

Meiner Erfahrung nach sind es oft gerade die einfachen Beschreibungen, die all jene Missverständnisse offen legen, die man lieber früh als spät entdeckt.

Stories in mehrere Stories zerlegen

Eine Story, oder besser gesagt ihr Aufwand, sollte weder zu klein, noch zu groß sein. Ein Stapel von 0,5-Punkte Stories führt nicht selten zu Mikromanagement. 40-Punkte Stories laufen Gefahr, bis zum Sprint-Ende nicht abgeschlossen zu werden und für das Unternehmen nur Kosten statt Wert zugenerieren. Auch die Planung wird schwierig, wenn man bei einer Entwicklungsgeschwindigkeit von z.B. 70 zwei 40-Punkte Stories im Sprint unterbringen will. Mit einer Story unterfordern Sie das Team, nehmen Sie beide, überfordern Sie es.

Ich habe die Erfahrung gemacht, dass sich eigentlich jede Story weiter zerlegen lässt. Wichtig ist nur, dass auch diese kleineren Stories von Wert für das Unternehmen sind.

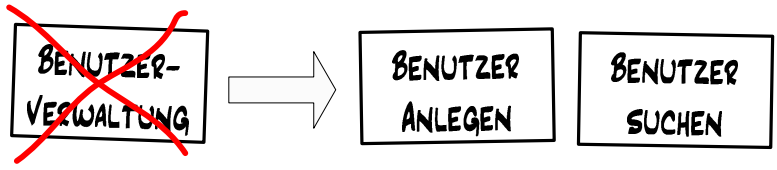
Normalerweise bemühen wir uns, Stories zwischen zwei und acht Tagen zu halten. Bei einer durchschnittlichen Entwicklungsgeschwindigkeit pro Team von 40 bis 60, können wir je Sprint ungefähr zehn Stories abarbeiten. Manchmal sind es nur fünf, ein anderes Mal 15. Mit so vielen Karteikarten kann man immer noch problemlos hantieren.

Stories in Aufgaben zerlegen

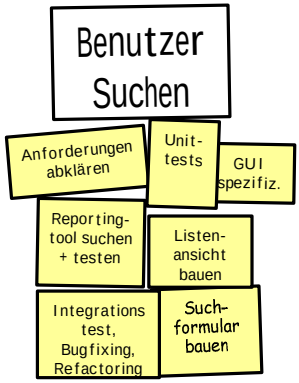
Was, bitte, ist denn der Unterschied zwischen Aufgaben und Stories?

Gute Frage; der Unterschied ist aber leicht erklärt. Während die Fertigstellung einer Story für den Product Owner von großem Belang ist, gilt das für die Fertigstellung einzelner Aufgaben nur selten.

Beispielhafte Zerlegung einer großen Story in zwei kleinere Stories:



Beispielhafte Zerlegung einer Story in Aufgaben:



Uns ist Folgendes aufgefallen:

- Scrum-unerfahrene Teams neigen dazu, zu viele Stories zu früh in Aufgaben zu zerlegen. Manche halten das für einen Rückfall ins Wasserfall-Modell.
- Stories, die man gut verstanden hat, lassen sich zu jedem Zeitpunkt leicht zerlegen.
- Eine Zerlegung bringt meist zusätzliche Aufgaben und Mehraufwände ans Licht. Der so gewonnene Sprint-Plan ist also realistischer.
- Die täglichen Scrum Meetings laufen dank der Zerlegung effizienter ab (siehe dazu Abschnitt "So läuft unser tägliches Scrum-Meeting ab").
- Diese Vorteile gelten auch, wenn die anfängliche Zerlegung vorläufig ist und später abgeändert wird.

Aus diesen Gründen reservieren wir uns in Sprint-Planungsmeetings für die Zerlegung etwas Zeit. Läuft uns dann doch die Zeit davon, verzichten wir einfach darauf (siehe dazu auch den Abschnitt "Wo zieht man die Grenze" weiter unten).

Hinweis: Wegen des Einsatzes Testgetriebener Entwicklung (TDD), ist die erste Aufgabe bei uns eigentlich immer das Schreiben von Unit-Tests und die letzte Aufgabe das Refaktorieren, um die Lesbarkeit zu erhöhen und redundanten Code zu minimieren.

Termin für tägliches Scrum-Meeting finden

Ein Ergebnis des Sprint-Planungsmeetings, Zeit- und Treffpunkt des täglichen Scrum-Meetings, wird gern vergessen. Da im ersten Termin vereinbart wird, welcher Kollege womit beginnt, bekommt der Sprint so gleich einen schlechten Start.

Ich bevorzuge Scrum-Meetings am Morgen, habe zugegebenermaßen aber noch nie probiert, sie mittags oder nachmittags abzuhalten.

Der Nachteil von Nachmittags-Terminen ist, das man sich bis zum Folgetag merken muss, was man am Vortag zu erledigen versprochen hat.

Scrum-Meetings am Morgen haben den Nachteil, dass man sich merken muss, was man am Vortag erledigt hat.

Ich denke, der erste Nachteil wiegt schwerer, da es wichtiger ist, zu wissen, was jemand tut, als zu wissen, was er getan hat.

Jeder sollte die Uhrzeit akzeptieren können. Normalerweise entscheiden wir uns für den frühest möglichen Termin, bei dem niemand mehr stöhnt - also 9:00, 9:30 oder 10:00 Uhr.

Wann man aufhören sollte

Was tun wir, wenn uns die Zeit davon rennt? Auf welches Ergebnis des Sprint-Planungsmeetings verzichten wir dann am ehesten?

Hier ist meine Prioritätenliste:

Priorität 1: Sprint-Ziel und Vorführungstermin - das sind die Mindestanforderungen, um einen Sprint zu beginnen. Das Team hat ein Ziel, einen Endtermin und kann Stories direkt aus dem Product Backlog heraus abarbeiten. Das ist zwar nicht ideal und oft ist es ratsam ein Folgemeeting zu vereinbaren. Wenn Sie den Sprint aber unbedingt beginnen müssen, dann geht es auch so.

Priorität 2: Liste aller für den Sprint vereinbarten Stories

Priorität 3: Aufwandschätzungen für eben jene Stories

Priorität 4: Erklärung, wie die Stories vorgeführt werden sollen

Priorität 5: Teamaufstellung und Plausibilitätsprüfung via Ressourcenverfügbarkeit und Entwicklungsgeschwindigkeit (denn: ohne konkretes Team gibt es auch keine Entwicklungsgeschwindigkeit)

Priorität 6: Zeit- und Treffpunkt des täglichen Scrum-Meetings. Auch wenn das sicherlich schnell erledigt ist, kann der ScrumMaster, sollte die Zeit einmal trotzdem nicht ausreichen, auch per Email einen Vorschlag machen.

Priorität 7: Zerlegung der Stories in Aufgaben. Diese Zerlegung kann auch täglich im Scrum-Meeting gemacht werden, was allerdings den Arbeitsfluss des Sprints immer etwas stört.

Technik-Stories

Problematisch ist der Umgang mit Technik-Stories, also nicht-funktionalen Anforderungen, die weder Liefergegenstand in Form einer Story sind, noch von unmittelbarem Wert für den Product Owner sind.

Beispiele für Technik-Stories sind:

- **Server zur Fortlaufenden Integration einrichten**
  - Motivation: Erspart den Entwicklern Unmengen von Zeit und reduziert das Risiko größerer Integrationsfehler am Ende des Sprints.
- **Systemarchitektur dokumentieren**
  - Motivation: Entwickler vergessen gerne einmal wichtige Architekturentscheidungen und schreiben dann davon abweichenden Code. Ein Architekturdokument schafft bei allen ein gemeinsames Verständnis der Architektur.
- **Überarbeiten der Datenzugriffsschicht**
  - Motivation: Unnötig komplizierter Code in der Datenzugriffsschicht raubt Zeit und führt zu Bugs. Räumt man hier auf, erspart man allen Mühe und verbessert die Stabilität des Systems.
- **Ein Bugtracker- Update installieren**
  - Motivation: Ist die aktuelle Version etwas langsam und fehlerhaft, erspart so ein Update dem Team Ärger und Zeit.

Hier stellt sich die Frage, ob dies reguläre Stories sind, oder Aufgaben, die zu keiner Story gehören? Wer entscheidet über ihre Wichtigkeit? Muss sich der Product Owner darum kümmern?

Beim Umgang mit Technik-Stories haben wir vieles ausprobiert. Wir haben sie wie reguläre Stories behandelt. Das hat sich nicht bewährt, weil damit das Priorisieren des Product Backlogs durch den Product Owner zum berüchtigten Vergleich von Äpfeln und Birnen wird.
Wie nicht anders zu erwarten, erhielten Technik-Stories meist niedrige Priorität. Ganz nach dem Motto: "Klar Jungs, ich bin sicher, so ein Integrationsserver ist wichtig. Aber lasst uns erst einmal etwas für den Umsatz tun und Technik-Spielereien später nachholen.“

Manchmal hat der Product Owner sogar recht damit. Da das meistens allerdings nicht zutrifft, haben wir beschlossen, dass er die falsche Person für derartige Abwägungen ist. Wir versuchen stattdessen...:

1) Technik-Stories ganz zu vermeiden oder sie in normale Stories mit messbarem, betriebswirtschaftlichem Nutzen umzuwandeln. So kann der Product Owner auch leichter Kompromisse finden.
2) Gelingt die Umwandlung in eine reguläre Story nicht, versuchen wir sie als Aufgabe in einer der Stories unterzubringen. Da z.B. die reguläre Story "Benutzer bearbeiten" die Datenzugriffsschicht benutzt, ist die Technik-Story "Überarbeiten der Datenzugriffsschicht“ ein Teil von ihr.
3) Klappt auch das nicht, belassen wir sie so und nehmen sie in eine separate Liste für Technik-Stories auf. Der Product Owner kann diese einsehen aber nicht bearbeiten. Wir verwenden den Fokus-Faktor und die geschätzte Entwicklungsgeschwindigkeit, um mit dem Product Owner für Technik-Stories ausreichend Zeit auszuhandeln.

Ein Gespräch, das dem folgenden recht ähnlich war, fand in einem unserer Sprint-Planungsmeetings statt:

- **Team:** “Es gibt einige Technik-Stories, die wir angehen müssen. Dafür möchten wir gern 10% unserer Zeit veranschlagen. Das bedeutet, dass wir den Fokus-Faktor im Gegenzug auf 75 oder 65 Prozent reduzieren müssten. Ist das in Ordnung? ”
- **Product Owner:** “ Gott bewahre - nein! Wir haben keine Zeit für so etwas. ”
- **Team:** “Aber schau Dir doch mal den letzten Sprint an (alle drehen sich zur Wandtafel mit den Werten der Entwicklungs-Geschwindigkeit.). Wir haben eine Entwicklungs-Geschwindigkeit von 80 geschätzt. Tatsächlich lag sie nur bei 30. ”
- **PO:** “ Stimmt! Und genau deshalb fehlt uns die Zeit für internen Technik-Kram. Was wir brauchen sind neue Features! ”

- **Team:** " Klar! Grund für die niedrige Entwicklungs-Geschwindigkeit war aber, dass das Bauen von konsistenten Testreleases zuviel Zeit in Anspruch genommen hat."
- **PO:** "Ja und was heißt das jetzt?"
- **Team:** "Na, das heißt, dass die Entwicklungsgeschwindigkeit weiterhin so mies bleibt, wenn wir nichts dagegen unternehmen."
- **PO:** "Und weiter?"
- **Team:** "Unser Vorschlag ist, im kommenden Sprint 10% der Zeit darauf zu verwenden, Dinge wie einen Server zur fortlaufenden Release-Erstellung einzurichten, um die Integration zu erleichtern. Das verbessert unsere Entwicklungsgeschwindigkeit vermutlich um 20%. Und das dauerhaft! "
- **PO:** "Tatsächlich!? Und warum haben wir das nicht schon früher gemacht?! "
- **Team:** " Ähhm...weil Du das nie gefordert hast... "
- **PO:** " Ach so! Na gut, dann sollten wir es wenigstens jetzt tun! "

Eine Alternative wäre natürlich, den Product Owner erst gar nicht vor die Wahl zu stellen und stattdessen einen unverhandelbaren Fokus-Faktor vorzugeben. Es gibt aber eigentlich keinen Grund, nicht erst einmal den Konsens zu suchen.

Ist der Product Owner ein kompetenter und vernünftiger Zeitgenosse (bei uns bisher immer), rate ich, ihn soweit möglich über alles zu informieren und ihn die Prioritäten festlegen zu lassen. Denn, wie wir wissen, ist Transparenz einer der Grundpfeiler von Scrum.

Bugtracker oder Product Backlog

Excel ist ein großartiges Werkzeug zur Pflege des Product Backlogs. Für die Erfassung von Bugs ist es hingegen wenig geeignet. Aus diesem Grund verwenden wir dafür Jira.

Wie aber beziehen wir Bugs aus Jira in das Sprint-Planungsmeeting ein? Nur die Stories zu betrachten und Bugs zu ignorieren, ist kein Weg.

Wir haben Verschiedenes ausprobiert:

1) Der Product Owner druckt kritische Bugs aus und hängt diese im Meeting zu den anderen Stories an die Wand. Damit trifft er implizit die Entscheidung, wie wichtig die Behebung einzelner Bugs im Vergleich zu anderen Stories ist.

2) Für jeden Bug in Jira erstellt der Product Owner eigene Stories. Zum Beispiel "Beheben der kritischsten Fehler im Auswertungs-Backend: Jira-124, Jira-126 und Jira-180 ".
3) Bugfixing wird nicht als Teil des Sprints behandelt, sondern das Team reserviert sich mit einem ausreichend niedrigen Fokus-Faktor Zeit für Bugfixing. Dabei geht das Team davon aus, dass es pro Sprint einige Bugs in Jira abarbeitet.
4) Das Product Backlog wird nicht in Excel, sondern auch in Jira geführt. Bugs und Stories werden so gleich behandelt.

Noch haben wir uns nicht auf eine Vorgehensweise festgelegt und je nach Team und Sprint wird es anders gemacht. Mein persönlicher Favorit ist wegen ihrer Einfachheit die erste Methode.

Das Sprint-Planungsmeeting ist endlich vorbei

Unglaublich - ich hätte nicht gedacht, dass das Kapitel über Sprint-Planungsmeetings so lang werden würde. Das liegt vermutlich daran, dass ich das Meeting für einen so wichtigen Teil von Scrum halte. Je mehr Sie in eine gute Sprint-Planung stecken, desto leichter geht auch der Rest von der Hand.

Das Sprint-Planungsmeeting ist dann ein Erfolg, wenn es alle im Team und der Product Owner mit einem Lächeln verlassen, am nächsten Morgen mit einem Lächeln aufwachen und das erste Scrum Meeting mit einem Lächeln beginnen.

Sicherlich kann später noch einiges schief laufen – an der Planung des Sprints lag es dann aber nicht. :o)

5

So machen wir Sprints bekannt

Es ist wichtig, alle im Unternehmen über den Stand der Dinge auf dem Laufenden zu halten. Andernfalls beschweren sich Leute darüber oder - was noch schlimmer ist - treffen ihre eigenen Annahmen.

Um das zu vermeiden gibt es bei uns so genannte Sprint-Infoseiten.

Team Jackass, Sprint 15

Sprint-Ziel
- Betatest-fähiger Release

Sprint Backlog (geschätzter Aufwand in Klammern)
- Geld einzahlen (3)
- Migrations-Tool (8)
- Login Backend (5)
- Benutzerverwaltung (5)

Geschätzte Entwicklungsgeschwindigkeit: 21

Termine
- Sprint Dauer: 06.- 24.Nov 2006
- Tägliches Scrum Meeting: 9:30–9:45 Uhr, Teamraum
- Sprint Vorführung: 24.Nov 2006, 13 Uhr, Cafeteria

Team
- Jim
- Erica (Scrum Master)
- Tom (75%)
- Eva
- John

Manchmal beschreiben wir sogar, wie einzelne Stories vorgeführt werden.

Der ScrumMaster legt sofort nach dem Sprint-Planungsmeeting im Wiki diese Seite an und informiert alle darüber per Email.

Betreff: Team Jackass beginnt Sprint 15

Hallo zusammen! Das Team Jackass hat begonnen, am Sprint 15 zu arbeiten. Unser Ziel ist es, am 24. November ein Betatest-fähiges Release vorzuführen.

Mehr zu diesem Sprint findet Ihr im Wiki unter:
http://wiki.meinefirma.de/jackass/sprint15

Zusätzlich haben wir in unserem Wiki eine Übersichtsseite mit Links zu allen derzeit laufenden Sprints.

Aushang Unsere Firma

Laufende Sprints
- Team X Sprint 15
- Team Y Sprint 12
- Team Z Sprint 1

Der ScrumMaster druckt die Sprint-Infoseite aus und hängt sie vor das Teambüro. Jeder, der vorbeikommt, sieht so woran das Team gerade arbeitet. Und da auch Zeit- und Treffpunkt des täglichen Scrum-Meetings vermerkt sind, weiß man sogar, wie man an zusätzliche Details kommt.

Nähert sich ein Sprint seinem Ende, erinnert der ScrumMaster jeden noch einmal an die bevorstehende Sprint-Vorführung.

Betreff: Sprint-Vorführung Team Jackass, morgen 13 Uhr Cafeteria.

Hallo zusammen! Wir möchten Euch einladen an unserer morgigen Sprint-Vorführung teilzunehmen. Wir zeigen ein Betatest-fähiges Release. Seid dabei. Morgen am Freitag 13 Uhr in der Cafeteria.

Weitere Details findet Ihr im Wiki unter:
http://wiki.meinefirma.de/jackass/sprint15

Es gibt also keine Entschuldigung dafür, dass irgend jemand nicht Bescheid weiß.

6

So sieht unser Sprint Backlog aus

Gute Arbeit! Sie sind schon recht weit vorangekommen.

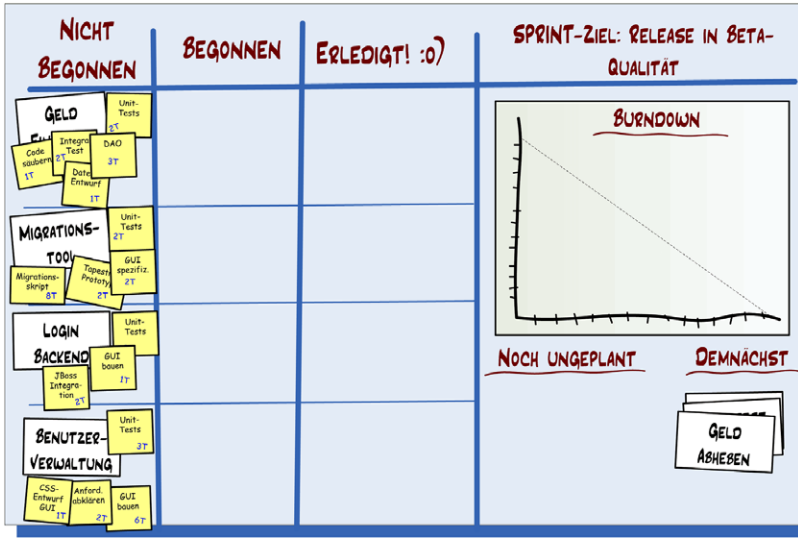
Jetzt wo das Sprint-Planungsmeeting vorbei ist und alle Welt von unserem tollen neuen Sprint weiß, wird es für den ScrumMaster höchste Zeit, ein Sprint Backlog anzufertigen. Das sollte nach dem Sprint-Planungsmeeting und vor dem ersten Scrum-Meeting passieren.

Sprint Backlog Formate

Wir haben mit verschiedenen Tools und Formaten für das Sprint Backlog herumprobiert, z.B. mit Jira, Excel und Wandtafeln. Früher haben wir oft Excel eingesetzt, weil es viele kostenlose Vorlagen für Sprint Backlogs gibt. Manche sind sogar in der Lage automatisch Burndown-Diagramme zu erzeugen.

Auf unsere eigenen Erweiterungen solcher Excel-Vorlagen möchte ich nicht weiter eingehen und lieber von dem Format reden, mit dem wir die besten Erfahrungen gemacht haben - einer Aufgabentafel an der Wand.

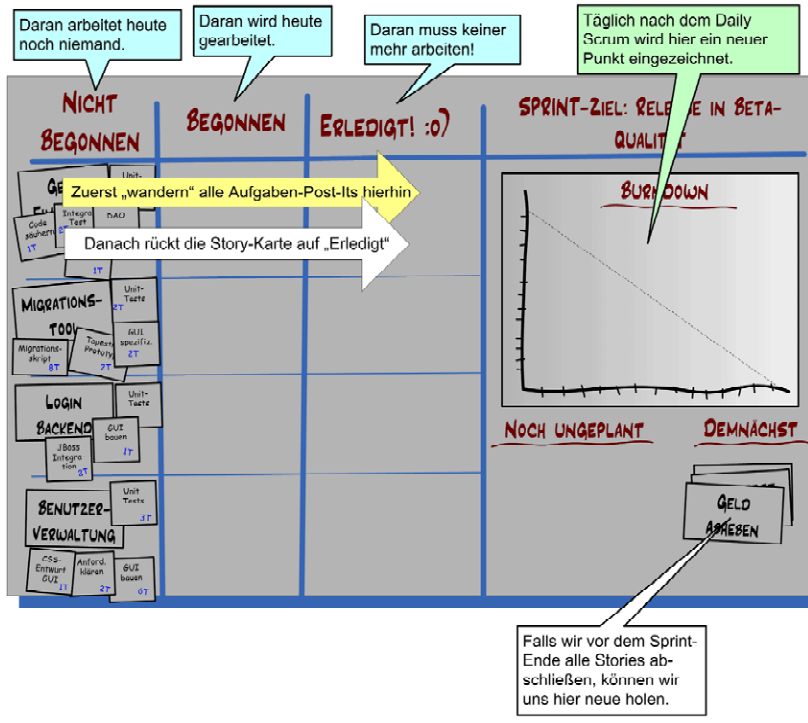
Suchen Sie sich eine große Wandfläche, die frei ist oder nur mit Firmenlogos, alten Diagrammen oder hässlichen Bildern verschwendet wird. Entfernen Sie alles (fragen Sie dafür nur in Ausnahmen um Erlaubnis) und bekleben Sie die Wand mit einem riesigen Papier (mindestens 2x2 Meter groß, für große Teams noch besser 3x2 Meter).



Natürlich können Sie dafür auch Schreibtafeln verwenden. Da Sie diese aber auch für Designskizzen brauchen könnten, ist das reine Verschwendung. Lassen Sie diese lieber frei und nutzen stattdessen eine der anderen Wände.

HINWEIS: Denken Sie unbedingt daran, Aufgaben-Klebezettel mit Klebeband zu sichern. Andernfalls liegen diese bald alle auf dem Boden.

So funktioniert die Aufgabenwand

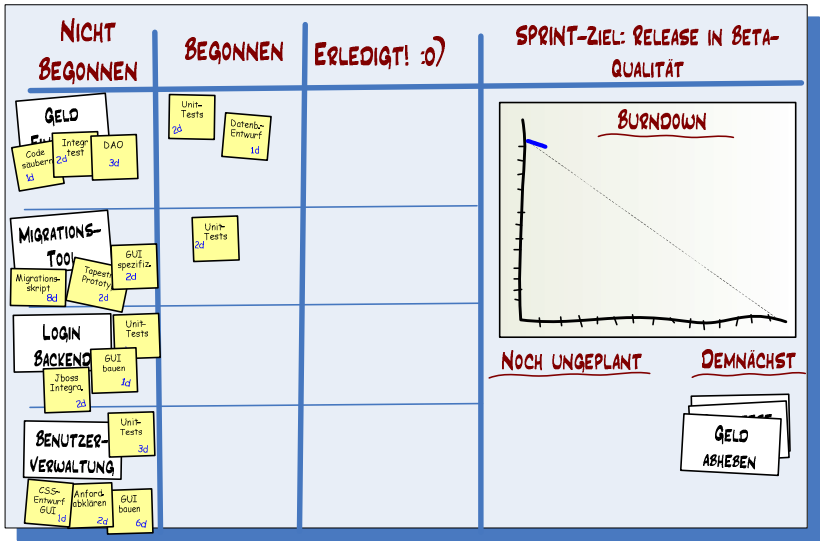


Sie könnten selbstverständlich weitere Spalten wie "Bereit zum Integrationstest" oder "Gestrichen" hinzufügen. Überlegen Sie sich aber gut, ob es sich wirklich lohnt, dafür alles zu verkomplizieren.

Meiner Erfahrung nach ist Einfachheit an dieser Stelle von außerordentlichem Wert. Zusätzliche Details lohnen sich nur in Ausnahmefällen.

Beispiel 1 - nach dem ersten Scrum-Meeting

Nach dem ersten Scrum-Meeting sieht die Aufgabenwand eventuell so aus:

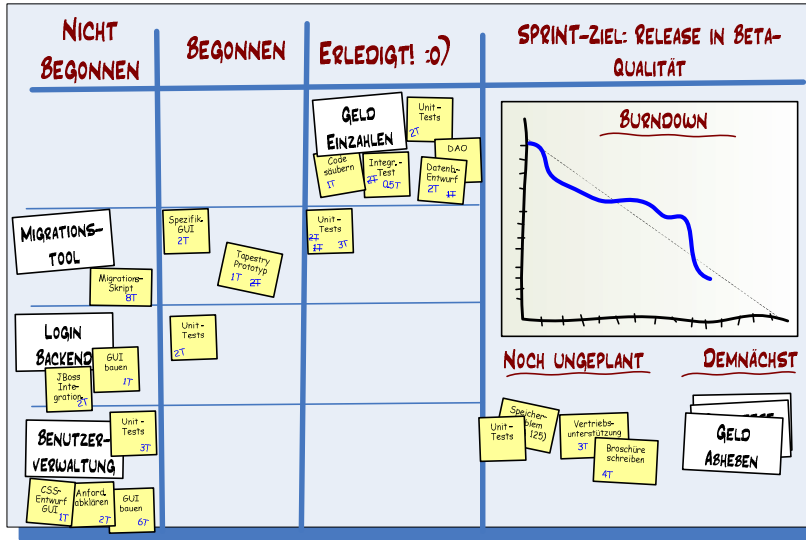


Wie Sie sehen, wurden drei Aufgaben "ausgecheckt", d.h. das Team wird diese heute bearbeiten.

In großen Teams kommt es vor, dass Aufgaben ewig in diesem Status verbleiben. Einfach weil vergessen wurde, wer sie ursprünglich bearbeitet hat. Passiert das häufiger, sollten Sie den Namen der Person, welche die Aufgabe zuerst bearbeitet hat, auf die Karte schreiben.

Beispiel 2 – nach ein paar Tagen

Ein paar Tage später sieht die Aufgabenwand dann vielleicht so aus:



Man sieht, die Story "Geld einzahlen" wurde erledigt. Das heißt soviel wie, dass der Code in die Versionskontrolle eingecheckt, getestet und refaktoriert wurde. Die Arbeit an den Stories "Migrations-Tool" und "Login Backend" wurden begonnen, die Arbeit an der „Benutzerverwaltung“ jedoch noch nicht.

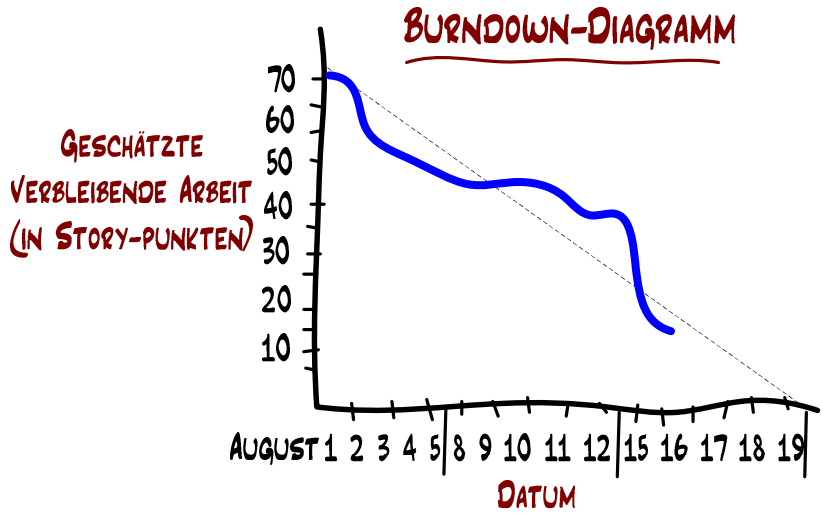
Außerdem sieht man in der Ecke rechts unten, dass es vier Themen gibt, die noch gar nicht eingeplant sind. Denken Sie bei der Sprint-Retrospektive daran.

Im Folgenden sehen Sie ein echtes Sprint Backlog am Ende eines Sprints.

Es ist nicht allzu schlimm, dass das Backlog im Laufe der Zeit immer chaotischer wird. Da Sie für jeden Sprint sowieso ein neues Backlog anlegen, wird es nicht mehr lange gebraucht.

So funktioniert ein Burndown-Diagramm

Lassen Sie uns das Burndown-Diagramm genauer betrachten:



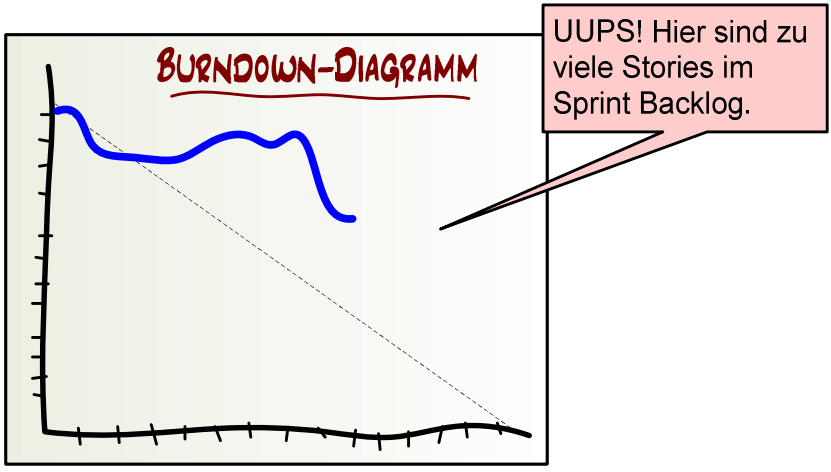
Das gezeigte Diagramm zeigt, dass:

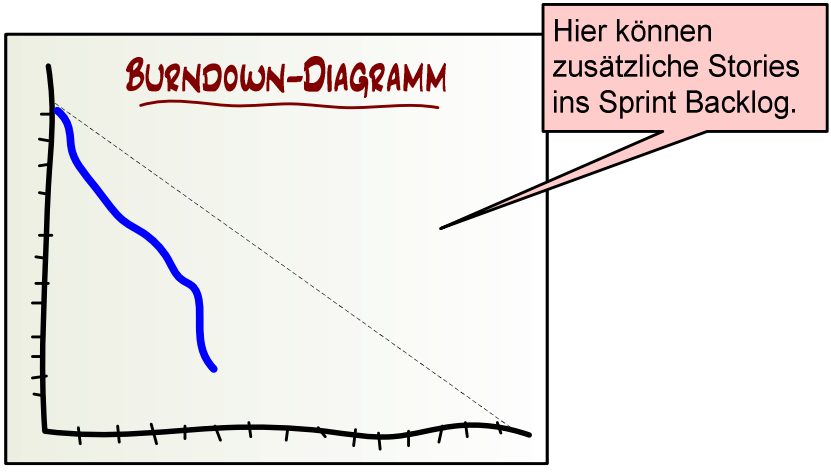
- am 1. August, dem ersten Tag des Sprints, das Team von 70 Story-Punkten für den Sprint ausgeht. Dies ist auch die geschätzte Entwicklungsgeschwindigkeit des Sprints.
- am 16. August schätzt das Team die verbleibende Arbeit auf 15 Story-Punkte. Die gestrichelte Linie zeigt, dass man ungefähr im Zeitplan liegt und bis zum Ende des Sprints alles abschließt, wenn man die Geschwindigkeit beibehält.

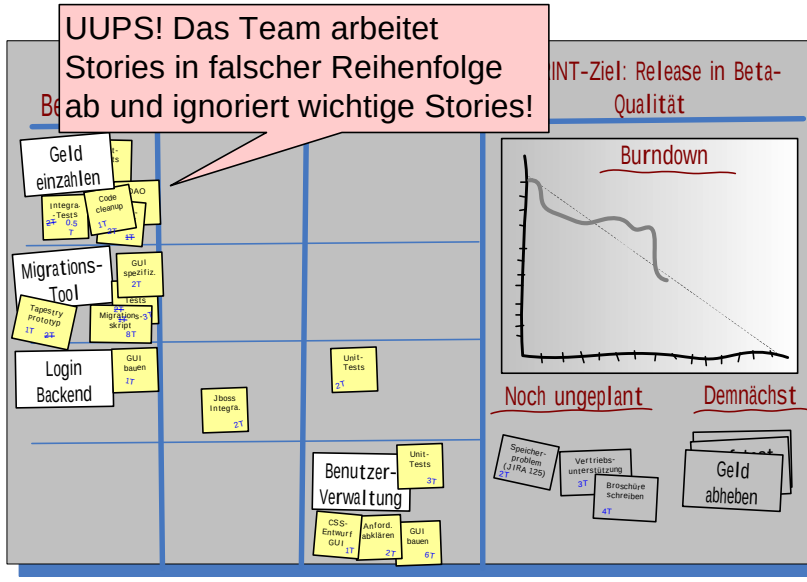
Da wir nur selten am Wochenende arbeiten, haben wir ganz darauf verzichtet, diese in der X-Achse aufzuführen. Nimmt man, wie wir früher, Wochenenden dazu, flacht die Linie an Wochenenden so ab, dass das fälschlicherweise als Alarmsignal interpretiert werden könnte.

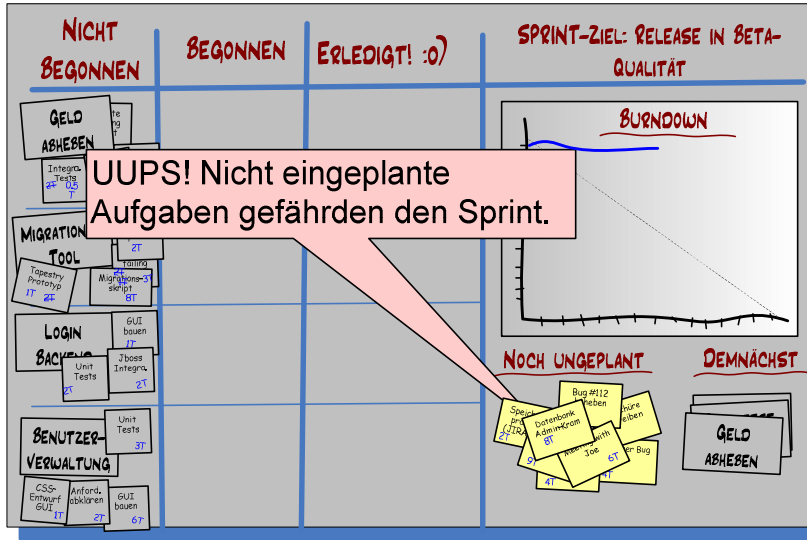
Alarmsignale auf der Aufgabenwand

Ein kurzer Blick auf die Aufgabenwand verrät schnell, wie es im Sprint vorangeht. Es liegt in der Verantwortung des ScrumMaster, dass das Team auf Alarmsignale auch reagiert.






UUPS! Das Team arbeitet Stories in falscher Reihenfolge ab und ignoriert wichtige Stories!
INT-Ziel: Release in Beta-Qualität
Burndown
Geld einzahlen
Migrations-Tool
Login Backend
GUI bauen
Jboss Integra.
Unit-Tests
Benutzer-Verwaltung
Noch ungeplant
Demnächst
Geld abheben


NICHT BEGONNEN
BEGONNEN
ERLEDIGT! :O)
SPRINT-ZIEL: RELEASE IN BETA-QUALITÄT
BURNDOWN
UUPS! Nicht eingeplante Aufgaben gefährden den Sprint.
NOCH UNGEPLANT
DEMNÄCHST
GELD ABHEBEN
LOGIN
BENUTZER-VERWALTUNG

Wie sieht's mit der Rückverfolgbarkeit aus?!

Die beste mir bekannte Methode, um Änderungen nachvollziehen zu können, ist es, täglich Fotos der Aufgabenwand zu machen. Obwohl ich gelegentlich solche Fotos mache, habe ich sie später praktisch nie wieder herausgeholt.

Aufgabenwände sind nicht das richtige Mittel, wenn für Sie die Rückverfolgbarkeit von Änderungen sehr wichtig ist.

Aber selbst in diesem Fall, rate ich Ihnen den Nutzen detaillierter Rückverfolgbarkeit zunächst zu hinterfragen. Mal ehrlich - wen interessiert am Ende des Sprints noch, welche Stories am Tag 5 abgeschlossen wurden. Die Hauptsache ist doch, dass am Ende des Sprints funktionierender, dokumentierter Code ausgeliefert wird. Kaum jemand wird sich dann noch dafür interessieren, wie hoch die Schätzung für die Unit-Tests einer bestimmten Story war!

In Tagen oder Stunden schätzen

In den meisten Büchern und Artikeln zum Thema Scrum werden Aufgaben in Stunden und nicht in Tagen geschätzt. Für eine kurze Zeit haben wir das auch so gemacht und sind dabei von sechs effektiven Stunden pro Arbeitstag ausgegangen.

In den meisten Teams hat sich das aber nicht bewährt, weil:

- die vielen kleinen ein- bis zweistündigen Aufgaben zu verstärktem Mikromanagement geführt haben.
- Viele Kollegen weiterhin in Mann-Tagen dachten und ihre Schätzungen einfach mit 6 multiplizierten – ganz nach dem Motto: "Hmmm, diese Aufgabe dauert ungefähr einen Tag. Dann schreibe ich mal sechs Stunden hin."
- das Verwenden von zwei Einheiten zu Verwirrungen der Art "Haben wir hier in Tagen oder Stunden geschätzt?" führen kann.

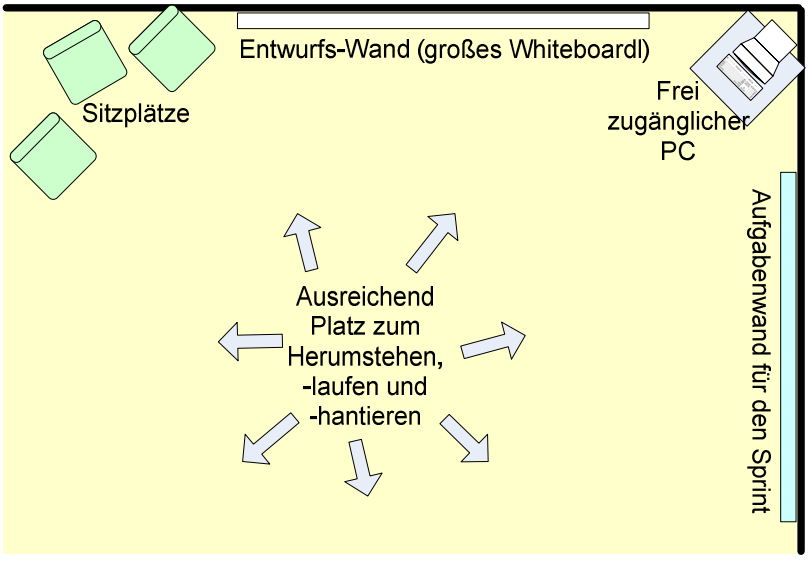
Aus den genannten Gründen machen wir Aufwandschätzungen in Mann-Tagen (die wir trotzdem Story-Punkte nennen). Der niedrigste Wert liegt bei 0,5. Aufgaben mit noch geringerem Aufwand führen wir entweder gar nicht auf, legen sie mit anderen Aufgaben zusammen, oder belassen ihren Wert bei 0,5. So einfach!

7

So richten wir den Teamraum ein

Die Entwurfs-Ecke

Weil die interessantesten und ergiebigsten Diskussionen bei uns oft an der Aufgabenwand entstehen, richten wir den Bereich gleich als Entwurfs-Ecke ein.



Das ist wirklich sehr praktisch, denn es gibt keinen besseren Weg, schnell einen Einblick ins System zu gewinnen, als durch einen Blick auf die Wände der Entwurfs-Ecke und das Ausprobieren des aktuellen Release an dem PC. Das Release steht natürlich nur dann zur Verfügung, wenn Sie, wie in Abschnitt "So kombinieren wir Scrum und XP" beschrieben, Fortlaufende Integration betreiben.

Unsere Entwurfs-Wand ist eine große Tafel mit den wichtigsten Architekturskizzen, Ausdrucken von Sequenzdiagrammen, Oberflächenprototypen oder Domänen-Modellen aus Architektur-Dokumenten.

Oben: ein Scrum-Meeting in der Entwurfs-Ecke

Komisch...das Burndown-Diagramm sieht verdächtig unauffällig und gerade aus!? Ob das Team wohl glaubt, dass die Realität so aussieht? :0)

Setzen Sie das Team zusammen!

Wenn es um die Anordnung der Arbeitsplätze geht, kann man eine Sache gar nicht oft genug sagen: Setzen Sie das Team unbedingt zusammen!

Setzen Sie das Team unbedingt zusammen!

Damit Sie mich auch richtig verstehen; ich sagte:

Setzen Sie das Team zusammen!

Die meisten Leute sträuben sich gegen Umzugsaktionen - zumindest dort, wo ich bisher gearbeitet habe. Keiner hat Lust den Rechner abzubauen, seinen ganzen Kram zusammenzupacken und alles am neuen Platz wieder aufzubauen und einzustöpseln. Je geringer der Abstand, desto stärker der Widerstand. "Ach, Chef, was sollen bitte diese 5 Meter bringen?!"

Um leistungsfähige Scrum-Teams aufzubauen, gibt es keine Alternative. Sorgen Sie auch dann dafür, dass das Team zusammensitzt, wenn Sie Einzelnen drohen und den Krempel selbst umräumen und alte Kaffeeflecken entfernen müssen. Wenn Sie keinen Platz finden, dann schaffen Sie welchen. Selbst wenn Sie auf den Keller ausweichen müssen. Verrücken Sie Bürotische, bestechen Sie den Office-Manager; tun Sie einfach alles, was nötig ist, um das Team zusammenzusetzen.

Ist das erst einmal geschafft, werden Sie den Unterschied bemerken. Schon nach dem ersten Sprint wird das Team bestätigen, dass der Umzug eine gute Idee war. Aber selbst wenn Ihr Team zu dickköpfig ist, das zuzugeben, bei uns jedenfalls war es so.

Aber was heißt "zusammensitzen" denn nun eigentlich? Was die konkrete Anordnung der Arbeitsplätze angeht, bin ich nicht wirklich festgelegt. Einfach deshalb, weil Sie da vermutlich durch Umstände wie das Nachbar-Team, die Toilettentür oder den Snackautomaten beschränkt werden und nur wenig Spielraum haben.

Mit "Zusammensetzen" meine ich insofern eher Dinge wie:

- **Hörbarkeit:** Jeder kann mit allen sprechen, ohne dafür seinen Platz verlassen oder schreien zu müssen.
- **Sichtbarkeit:** Jeder kann alle anderen sehen. Außerdem sollte jeder so nah an der Aufgabenwand sitzen, dass er sie lesen oder zumindest sehen kann.

- **Abgeschiedenheit:** Niemand außerhalb des Teams wird gestört, wenn sich plötzlich eine lebhafte Architektur-Diskussion entwickelt. Das gilt natürlich auch für Störungen durch andere Teams.

Mit Abgeschiedenheit meine ich nicht, dass das Team völlig isoliert sein muss. In einer Arbeitsumgebung mit Trennwänden ist es sicherlich ausreichend, dem Team einen eigenen Bereich zu bauen, dessen Stellwände hoch genug sind, um Umgebungsgeräusche zu dämpfen.

Ist das Team allerdings räumlich verteilt, haben Sie Pech. Nutzen Sie dann alle technischen Mittel wie z.B. Videokonferenzen, Webcams, Desktop-Sharing-Software, um die Probleme zu minimieren.

Halten Sie den Product Owner auf Abstand

Der Product Owner sollte in der Nähe sitzen, damit das Team bei Fragen schnell bei ihm vorbeischauen und für kurze Diskussionen zur Aufgabenwand gehen kann. Er sollte nicht direkt beim Team sitzen, weil sonst die Gefahr besteht, dass er sich in Details einmischt und ein Zusammenschweißen der Kollegen verhindert. Nur so erreicht das Team aber seinen hochgradig produktiven und eigenständigen Arbeitsmodus.

Allerdings bleibt das eine reine Vermutung, da mir kein Fall bekannt ist, bei dem der Product Owner mit dem Team zusammen saß. Mein Rat beruht also weniger auf Erfahrungswerten, als meinem Bauchgefühl und dem Hörensagen anderer ScrumMaster.

Halten Sie Manager und Berater auf Abstand

Da ich früher sowohl Manager als auch Berater war, fällt es mir etwas schwer, über dieses Thema zu schreiben...

Damals war es meine Aufgabe, so nah wie möglich mit dem Team zusammenzuarbeiten. Ich habe Teams zusammengestellt, in den einzelnen Teams in Paaren programmiert, ScrumMaster ausgebildet, Sprint-Planungsmeetings durchgeführt und vieles mehr.
Viele betrachten es im Rückblick als eine gute Sache, weil ich bereits einige Erfahrung mit Agiler Softwareentwicklung hatte.

Das Dumme war aber, dass - jetzt bitte Darth Vader Melodie einspielen – ich damals auch Abteilungsleiter der Softwareentwicklung, also ein Vorgesetzter war.

Dadurch, dass ich Teil des Teams wurde, machte ich es gleichzeitig unselbständiger. So in der Art: “Was soll's! Der Chef hat sicher klare Vorstellungen darüber, wer was zu tun hat. Dann lass ich ihn lieber mal entscheiden.”

Worauf es mir ankommt, ist Folgendes: Bringen Sie sich anfangs stark ein, wenn Sie sowohl Scrum-Trainer, als auch Manager sind. Ziehen Sie sich nach einer gewissen Zeit aber zurück, damit das Team zusammenwachsen und sich selbst organisieren kann. Kontrollieren Sie hin und wieder - aber nicht zu oft - den Stand der Dinge, indem Sie an Sprint-Vorführungen und Scrum-Meetings teilnehmen oder an der Aufgabenwand vorbeischauen. Falls Sie Verbesserungsmöglichkeiten entdecken, nehmen Sie den ScrumMaster beiseite und leiten ihn im Zweiergespräch an. Vertraut das Team Ihnen so, dass es sich in Ihrer Anwesenheit nicht gehemmt fühlt, sollten Sie auch an Retrospektiven teilnehmen. Dazu aber mehr im Abschnitt "So laufen unsere Sprint-Retrospektiven ab".

Geben sie Scrum-Teams, die bereits gut zusammenarbeiten, alles, was sie zur Arbeit brauchen und gehen ihnen ansonsten aus dem Weg. Dieser Rat gilt natürlich nicht für die Sprint-Vorführung.

8

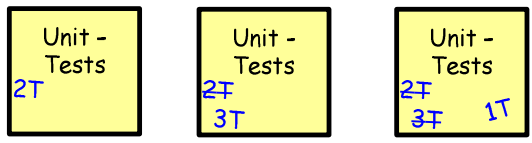
So läuft das tägliche Scrum-Meeting ab

Unsere Scrum-Meetings laufen eigentlich ganz nach Vorschrift ab. Sie finden jeden Tag pünktlich zur selben Zeit am selben Ort statt. Während wir früher dafür noch einen eigenen Raum und ein elektronisches Sprint Backlog hatten, machen wir Scrum-Meetings heute direkt vor der Aufgabenwand im Teambüro. Besser geht es nicht.

Wir machen das Meeting im Stehen, weil wir so viel seltener die geplanten 15 Minuten überziehen.

So aktualisieren wir die Aufgabenwand

Dabei aktualisieren wir oft auch gleich die Aufgabenwand. Derjenige, der gerade erzählt, was er getan hat oder tun wird, ordnet parallel die Klebezettel an der Wand um. Für bisher nicht geplante Aufgaben fügt er weitere Klebezettel hinzu. Zur Korrektur von Schätzungen, streicht er auf den Zetteln den alten Wert und fügt den neuen hinzu. Diese Arbeit an den Klebezetteln übernimmt manchmal auch der ScrumMaster.



Andere Teams haben vereinbart, dass jeder für sich vor dem Meeting seine Änderungen an der Aufgabenwand macht. Auch das funktioniert ziemlich gut. Legen Sie sich einfach auf eine Regelung fest und bleiben Sie dabei. Beziehen Sie alle in die Aktualisierung des Sprint Backlogs ein, unabhängig von seinem konkreten Format. Wir haben versucht, dass ausschließlich der ScrumMaster dafür verantwortlich ist, täglich den Status bei allen zu erfragen und die Wand zu aktualisieren.

Nachteile dieser Vorgehensweise sind:

- Anstatt das Team zu unterstützen und Hindernisse zu beseitigen, verbringt der ScrumMaster viel zuviel Zeit mit organisatorischen Aufgaben.
- Die Teammitglieder müssen sich nicht mehr um den Sprint Backlog kümmern und kennen deshalb den aktuellen Status des Sprints auch nicht. Damit fehlt eine Feedback-Möglichkeit und die Agilität und Konzentration des Teams sinkt.

Ein gut funktionierendes Sprint Backlog muss von jedem leicht aktualisiert werden können.

Direkt nach dem Scrum-Meeting werden die Aufwandschätzungen noch unerledigter Aufgaben zusammengezählt und das Ergebnis als Punkt ins Burndown-Diagramm eingetragen.

Wie gehen wir mit Nachzüglern um?

Einige Teams haben ein Sparschwein. Jeder, der zu spät kommt - sei es auch nur eine Minute - muss einen bestimmten Betrag zahlen. Ohne Widerrede! Auch dann, wenn man vor dem Meeting anruft und Bescheid gibt, dass man sich verspätet. Als Ausreden gelten lediglich Arztbesuche, die eigene Hochzeit oder Ähnliches.

Das Geld wird für gemeinsame Unternehmungen verwendet, z.B. um Hamburger essen zu gehen oder für einen gemeinsamen Spieleabend. Diese Methode funktioniert sehr gut, ist aber nur in Teams nötig, in denen die Leute oft zu spät kommen. Andere Teams brauchen so etwas nicht.

Hilfe bei “Und was soll ich heute machen?”

Nicht selten sagt jemand: "Gestern war ich beschäftigt mit XYZ, aber ich habe keinen blassen Schimmer, was ich heute tun soll". Was dann?

Nennen wir die beiden einmal Joe und Lisa.

Bin ich der ScrumMaster, notiere ich mir, wer nichts zu tun hat und gehe zum Nächsten in der Reihe weiter. Nachdem die Runde abgeschlossen ist, bitte ich das Team zur Aufgabenwand und sorge dafür, dass alle den aktuellen Status und die einzelnen Story-Karten verstehen. Ich stelle es den Kollegen frei, zusätzliche Klebezettel anzubringen. Anschließend frage ich die beiden "Arbeitslosen", ob ihnen mittlerweile eingefallen ist, was sie tun könnten, und hoffe auf eine positive Antwort.

Kommt diese nicht, überlege ich, ob es eine Möglichkeit gibt, als Paar zu programmieren. Will Niklas heute etwa mit der Programmierung der Benutzeroberfläche des Administrations-Backends beginnen, frage ich ihn freundlich, ob Joe oder Lisa mitmachen können. Das funktioniert normalerweise.

Funktioniert auch das nicht, gibt es hier ein weiterer Trick:

ScrumMaster: "OK, wer möchte uns den Betatest-fähigen Release vorführen?" (angenommen, das war das Sprint-Ziel)
Team: *Betretenes Schweigen*
ScrumMaster: "Wir sind doch fertig, oder?"
Team: "Ähm...nein!"
ScrumMaster: "Verdammt! Warum nicht? Was fehlt denn noch?"
Team: "Also, wir haben noch nicht einmal einen Testserver und das Build-Skript funktioniert auch noch nicht."
ScrumMaster (hängt zwei Zettel an die Wand): "Aha! Joe, Lisa, könnt ihr uns heute vielleicht bei diesen Themen helfen?"
Joe: "Hmm... ich schaue mich gleich mal nach einem Testserver um."
Lisa: " ... und ich kümmere mich um das Build-Skript."

Wenn Sie Glück haben, führt tatsächlich jemand ein fertiges Release vor. Perfekt! Sie haben ihr Sprint-Ziel erreicht. Was, aber, wenn Sie mitten in einem Sprint sind? Auch kein Problem! Beglückwünschen Sie das Team zu seiner guten Arbeit, nehmen sie ein bis zwei neue Stories vom „Demnächst"-Bereich unten rechts auf der Aufgabenwand und schieben sie in die Spalte "Nicht begonnen". Danach wiederholen Sie das Scrum-Meeting und teilen dem Product Owner mit, dass sie dem Sprint Stories hinzugefügt haben.

Was aber tun Sie, wenn Joe und Lisa nichts Sinnvolles einfällt, obwohl das Sprint-Ziel noch nicht erreicht ist? Normalerweise wende ich dann eine der folgenden Strategien an (alle eher Notlösungen als angenehme Kompromisse):

- **Schuldgefühle machen:** "Nun, falls Ihr nicht wisst, wie Ihr dem Team helfen könnt, schlage ich vor, Ihr geht nach Hause, lest ein Buch oder tut worauf Ihr Lust hast. Oder Ihr wartet, bis jemand Euch um Hilfe bittet." ".
- **Wie in alten Zeiten:** Weisen Sie den beiden einfach eine Aufgabe zu.
- **Gruppenzwang:** Sagen Sie: "Joe, Lisa, nehmt Euch die Zeit, die ihr braucht, um zu überlegen, wie Ihr uns dabei unterstützen könnt, das Sprint-Ziel zu erreichen. Wir warten gerne."

- **Diener:** Schlagen Sie vor: "Ihr könnt dem Team heute indirekt helfen und Diener spielen. Bringt Kaffee, massiert uns, bringt den Müll weg, kocht Mittagessen oder tut, was man sonst so von Euch verlangt."

Sie werden sich wundern wie schnell Joe und Lisa sinnvolle technische Aufgaben einfallen werden. :o) Müssen Sie mit bestimmten Kollegen regelmäßig so weit gehen, sollten Sie diesen unter vier Augen etwas Nachhilfe in Teamarbeit geben. Ändert selbst das nichts, sollten Sie überlegen, ob diese Person für Ihr Team wirklich wichtig ist.

Ist sie es nicht, sollten Sie versuchen, sie loszuwerden.

Ist sie aber wichtig, stellen Sie ihr einen Mentor zur Seite. Joe ist vielleicht ein großartiger Entwickler oder Architekt, bevorzugt es trotzdem, wenn ihm andere Leute sagen, was er tun soll. Fein! Machen Sie Niklas zum Mentor von Joe oder übernehmen sie die Rolle selbst. Falls Joe für ihr Team wichtig ist, dann ist das auch die Mühe wert. Wir hatten solche Fälle, die auch mehr oder minder erfolgreich verliefen.

9

So präsentieren wir Sprint-Ergebnisse

Die Sprint-Vorführung (oder Sprint-Review, wie manche Leute sagen) ist ein wichtiger und oft unterschätzter Bestandteil von Scrum.

“Wozu denn eine Demo? Was können wir schon Tolles vorführen?”
“Wir haben keine Zeit, die &%$# Vorführung vorzubereiten!”
“Ich habe keine Zeit, mir die Vorführungen anderer Teams anzusehen!”

Bei uns gilt: Kein Sprint ohne Vorführung

So unspektakulär sie auch sein mag; eine gutgemachte Sprint-Vorführung hat eine wichtige Wirkung, weil:

- das Team Anerkennung für seine Leistung bekommt. Das fühlt sich gut an.
- Andere Leute mitbekommen, woran das Team gerade arbeitet.
- Interessenvertreter unerlässliches Projekt-Feedback geben können.
- Vorführungen wichtige soziale Events sind, oder es zumindest sein sollten. Außerdem bieten sie allen die Gelegenheit für Gespräche und Diskussionen über die Arbeit.
- Vorführungen das Team zwingen, Arbeit wirklich abzuschließen und zumindest auf der Testumgebung als Release zu installieren. Ohne Vorführungen häuft sich ein riesiger Berg von zu 99 Prozent abgeschlossenen Arbeiten auf. Durch die Vorführungen hingegen wird insgesamt zwar weniger, dafür aber vollständig abgearbeitet. Uns ist das wesentlich lieber.

Sollen Teams etwas vorführen, das noch nicht abgeschlossen ist, wird es meist recht peinlich: Stammelnde Teamkollegen, magerer Applaus und Kollegen, die sich darüber beschweren, für eine miserable Vorführung Zeit vergeudet zu haben.

Das tut weh, wirkt aber wie bittere Medizin. Im nächsten Sprint wird sich das Team Mühe geben, seine Arbeit abzuschließen. In ihren Köpfen wird ungefähr Folgendes vorgehen: "Nun, vielleicht können wir beim nächsten Sprint nur zwei statt fünf Features zeigen, aber dann werden sie - verdammt noch mal – perfekt funktionieren!".

Ein Team, dass weiß, dass es, komme was wolle, eine Vorführung machen muss, wird wesentlich wahrscheinlicher auch etwas Funktionierendes zeigen. Das ist zumindest meine Erfahrung.

Checkliste für Sprint-Vorführungen

- Erklären Sie das Sprint-Ziel gut. Nehmen Sie sich Zeit, solchen Zuschauern das Produkt zu erklären, die es noch nicht kennen.
- Vergeuden Sie keine Zeit, die Vorführung großartig vorzubereiten, erst recht nicht für schicke Präsentationen. Verzichten Sie auf diesen Kram und beschränken Sie sich auf lauffähigen Code.
- Drücken Sie aufs Tempo und nutzen Sie Vorbereitungszeit dafür, die Vorführung flott und temporeich zu machen, anstatt sie hübsch aussehen zu lassen.
- Lassen Sie technische Details weg und konzentrieren Sie sich auf kundenrelevante Aspekte. Also: "Was haben wir gemacht" anstatt "Wie haben wir es gemacht".
- Lassen Sie das Publikum das Produkt eventuell ausprobieren.
- Verzichten Sie darauf, kleinere Bugfixes oder triviale Features zu zeigen. Diese zu erwähnen reicht, da es ansonsten zu lange dauert und die Aufmerksamkeit von wichtigen Features ablenkt.

Unvorführbares vorführen

Teammitglied: “Diesen Punkt führe ich nicht vor, weil ich nicht weiß, wie ich das zeigen soll. Die Story lautet 'Verbessern der Skalierbarkeit, so dass das System 10.000 Benutzer gleichzeitig verarbeiten kann'. Woher soll ich 10.000 Benutzer für die Vorführung auftreiben? ”
ScrumMaster: “Ist diese Anforderung denn implementiert?”
Teammitglied. “Ja, natürlich!”.
ScrumMaster: “Wie kannst Du Dir da sicher sein?”
Teammitglied: “Ich habe das System einem Lasttest mit acht Servern unterzogen, die das System mit gleichzeitigen Anfragen bombardieren.”
ScrumMaster: “Aber bist Du sicher, dass es 10.000 Benutzer aushält?”
Teammitglied: “Klar. Obwohl die Testrechner Schrott sind, konnten sie bei den Tests locker mit 50.000 gleichzeitige Anfragen umgehen.”
ScrumMaster: “Woher weißt Du das?”
Teammitglied (frustriert): “ Ich habe diesen Report hier! Schau, er zeigt, wie ich den Test aufgesetzt habe und die Anzahl der Anfragen! ”
ScrumMaster: “ Sehr gut! Das ist Deine "Vorführung". Zeige den Report und gehe ihn mit dem Publikum durch. Besser als nichts, oder? ”.
Teammitglied: “Und das reicht? Der Report ist nicht besonders ansehnlich. Ich werde ihn etwas hübsch machen.”
ScrumMaster: “Aber verliere nicht zu viel Zeit. Er soll nicht schön, sondern informativ sein. ”

10

Wie wir Sprint-Retrospektiven durchführen

Warum wir auf Retrospektiven bei allen Teams bestehen

Am wichtigsten bei Retrospektiven ist, dass Sie dafür *sorgen, dass sie stattfinden.*

Aus unerfindlichen Gründen sind manche Teams nicht besonders scharf darauf, Retrospektiven zu machen. Ohne sanften Druck wird einfach zum nächsten Sprint übergegangen und auf die Retrospektive verzichtet. Vielleicht hat das auch Ursachen in der schwedischen Kultur... Man weiß es nicht. Trotz alledem sind sich alle einig, dass Retrospektiven sehr nützlich sind. Ich halte sie, neben den Sprint-Planungsmeetings, für die zweitwichtigste Sache bei Scrum, weil sie zeigen, was sich verbessern lässt.

Um gute Ideen zu entwickeln, brauchen Sie natürlich kein Meeting. Das können sie auch Zuhause in der Badewanne tun. Ob das Team solche Ideen aber übernimmt ist fraglich. Wahrscheinlicher ist es auf jeden Fall dann, wenn die Idee direkt "aus dem Team heraus“, also bei der Retrospektive, wo sich jeder einbringen und mitdiskutieren kann, entsteht.

Sie werden feststellen, dass ein Team ohne Retrospektiven seine Fehler wieder und wieder macht

So organisieren wir unsere Retrospektiven

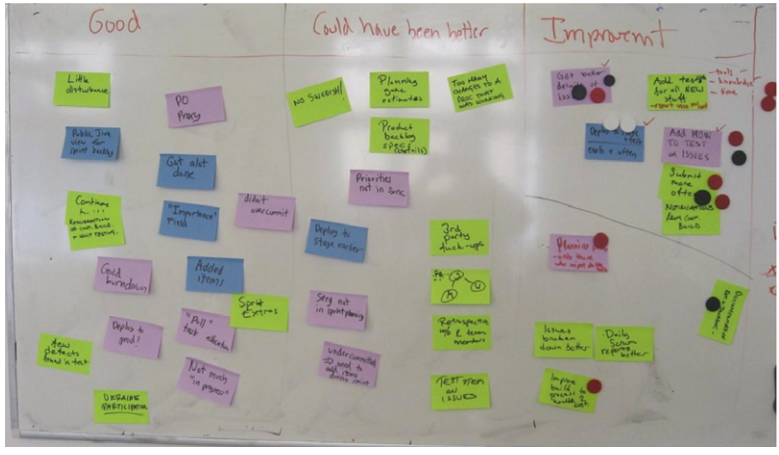
Selbst wenn wir gelegentlich leicht davon abweichen, ist der Ablauf meist der folgende:

- Je nach Gesprächsbedarf reservieren wir für das Meeting ein bis drei Stunden.

- Teilnehmer: Der Product Owner, das gesamte Team und Sie selbst.
- Wir verlegen das Treffen an einen Ort, wo man ungestört reden kann. Ein geschlossenes Zimmer, eine gemütliche Sitzecke, die Dachterrasse oder einen ähnlichen Ort.
- Normalerweise vermeiden wir Retrospektiven im Teambüro, da dort die Aufmerksamkeit der Leute sehr schnell wieder bei anderen Themen ist.
- Wir ernennen einen Teilnehmer als Sekretär.
- Der ScrumMaster geht die Einträge des Sprint Backlogs durch und rekapituliert gemeinsam mit dem Team wichtige Ereignisse und Entscheidungen des Sprints zusammen.
- Der Reihe nach hat jeder Gelegenheit, zu sagen, was gut lief, was besser gemacht hätte werden können und was er in kommenden Sprints verbessern möchte.
- Dann vergleichen wir die geschätzte von der tatsächlichen Entwicklungsgeschwindigkeit. Gibt es größere Abweichungen, versuchen wir herauszufinden, warum das so ist.
- Am Ende des Meetings fasst der ScrumMaster zusammen, welche konkreten Verbesserungsvorschläge für den kommenden Sprint gemacht wurden.

Unsere Retrospektiven sind normalerweise nicht allzu strukturiert. Der rote Faden des Meetings sollte immer die Frage sein "Was können wir beim nächsten Sprint besser machen?".

So sah kürzlich nach einer Retrospektive das Whiteboard aus:



Drei Spalten:

- **Gut:** Wenn wir den letzten Sprint noch mal wiederholen könnten, was würden wir genauso machen.
- **Hätte besser laufen können:** Wenn wir den letzten Sprint noch mal wiederholen könnten, was würden wir anders machen.
- **Verbesserungsvorschläge:** Konkrete Vorschläge, was wir in Zukunft besser machen können.

In Spalte 1 und 2 blicken wir in die Vergangenheit, in Spalte 3 in die Zukunft.
Nachdem die Teammitglieder durch Brainstorming diese Post-Its produziert haben, bestimmen sie mit einer 'Punktwertung', auf welche Verbesserungen sie sich im kommenden Sprint konzentrieren wollen. Jeder Kollege bekommt drei Magnete und kann damit die für ihn wichtigsten Verbesserungen bestimmen. Die Magnete dürfen frei verteilt werden - auch alle auf denselben Verbesserungsvorschlag.
Es werden fünf Prozessverbesserungen ausgewählt, die als nächstes angegangen werden sollen, und deren Erfolg in der nächsten Retrospektive überprüft wird.
Aber seien Sie nicht zu ehrgeizig und beschränken Sie sich pro Sprint auf ein paar wenige Verbesserungen.

Erfahrungsaustausch zwischen Teams

Die Erkenntnisse aus Sprint-Retrospektiven sind meist sehr wertvoll. Bereitet es dem Team z.B. Schwierigkeiten, bei der Sache zu bleiben, weil der Vertriebschef Entwickler immer wieder als "Technikexperten" zu Verkaufsgesprächen entführt? Gut zu wissen! Vielleicht haben andere Teams dasselbe Problem? Mit ein paar zusätzlichen Daten kann das Produktmanagement ja vielleicht selbst den Vertrieb unterstützen?

Bei Sprint-Retrospektiven geht es um weit mehr, als nur darum, die Arbeit eines einzelnen Teams im nächsten Sprint zu optimieren.

Bei uns hat sich folgendes eingebürgert. Eine Person - in diesem Fall ich - nimmt an allen Retrospektiven teil und fungiert als Wissensübermittler. Das alles läuft recht formlos ab.

Denkbar ist auch, dass jedes Scrum-Team einen Ergebnisbericht der Sprint-Retrospektive veröffentlicht. Wir haben das ausprobiert und festgestellt, dass diese Berichte kaum gelesen und noch weniger beherzigt werden. Also haben wir uns für den einfachen Weg entschieden.

Wichtige Eigenschaften des Wissensübermittlers sind:

- Er muss gut zuhören können.
- Er sollte darauf vorbereitet sein, mit einfachen, gezielten Fragen die Diskussion in Gang zu bringen, wenn die Retrospektive zu zaghaft verläuft. Beispielsweise mit dieser Frage: "Wenn ihr die Uhr zurückdrehen könntet und diesen Sprint von vorne beginnen könntet, was würdet ihr anders machen?
- Er muss bereit sein, die Retrospektiven aller Teams zu besuchen.
- Er sollte genug Autorität haben, Verbesserungsvorschläge umzusetzen, die außerhalb des Einflussbereichs des Teams liegen.

Das funktioniert ziemlich gut. Sicherlich können andere Ansätze sogar noch viel bessere Resultate bringen. In diesem Fall warte ich auf Ihre Tipps.

Ändern oder nicht ändern?

Nehmen wir einmal an, das Team kommt zur Einsicht, dass "wir zu wenig innerhalb des Teams kommuniziert haben, so dass wir uns immer wieder gegenseitig auf die Füße getreten sind und uns gegenseitig die Architektur verpfuscht haben".

Was lernen Sie daraus? Dass Sie tägliche Architekturmeetings oder bessere Kommunikationsmittel einführen müssen? Oder, dass Sie mehr Wiki-Seiten schreiben müssen? Ja - vielleicht. Vielleicht aber auch nicht.

Wir haben festgestellt, dass es oft schon ausreicht, ein Problem klar zu benennen, um es im nächsten Sprint automatisch zu vermeiden. Insbesondere, wenn die Ergebnisse der Retrospektive an der Wand im Teamraum ausgehängt werden. (Das vergessen wir immer wieder. Schande über uns!) Jede Veränderung hat ihren Preis. Ziehen Sie auch in Betracht, gar nichts zu tun und nur darauf zu hoffen, dass das Problem von alleine verschwindet oder kleiner wird.

Das Beispiel oben "das Team hat zuwenig mit einander kommuniziert" ist typisch für die Sorte von Problemen, die am Besten dadurch gelöst werden, dass man gar nichts tut.

Führen Sie bei jeder Beschwerde große Änderungen durch, lauen Sie Gefahr, die Mitarbeiter abzuschrecken bei der Identifizierung kleinerer Problemchen mitzuhelfen. Und das wäre schlimm.

Beispielhafte Ergebnisse einer Retrospektive

Hier einige typische Probleme in Bezug auf das Sprint-Planungsmeeting und passende Gegenmaßnahmen:

"Wir hätten mehr Zeit darauf verwenden sollen, die Stories in Unter-Stories und Aufgaben zu zerlegen"

Es kommt recht häufig vor, dass sich die Teammitglieder bei täglichen Scrum-Meeting mit der Frage "Ich weiß wirklich nicht, was ich heute machen soll." konfrontiert sehen. Obwohl es effektiver wäre, dies bereits vorher zu tun, findet die Identifikation konkreter Aufgaben dann nach jedem Scrum-Meeting statt.

Typische Maßnahme: Keine, den wahrscheinlich wird das Team bereits beim nächsten Sprint-Planungsmeeting für Abhilfe sorgen. Bleibt das Problem bestehen, planen Sie mehr Zeit für das Sprint-Planungsmeeting ein.

"Zu viele Störungen von Außen"

Typische Maßnahmen:
Raten Sie dem Team im kommenden Sprint,

- seinen Fokus-Faktor zu reduzieren, um einem realistischeren Plan zu bekommen
- Urheber und Dauer von Störungen zu protokollieren. So kann das Problem zu einem späteren Zeitpunkt besser gelöst werden.
- alle Störungen an den ScrumMaster oder Product Owner weiterzuleiten.
- Einen "Torwart" zu benennen, der alle Störungen abfängt, so dass der Rest des Teams sich konzentrieren kann. Diese Person kann entweder der ScrumMaster sein oder man lässt die Rolle rotieren.

"Wir haben zu viel versprochen und nur die Hälfte geschafft"

Typische Maßnahme: Keine. Im nächsten Sprint wird sich das Team sicher nicht mehr so übernehmen. Zumindest nicht so stark.

"Unser Arbeitsplatz ist zu laut und unordentlich"

Typische Maßnahmen:

- Verbessern Sie den Arbeitsplatz oder ziehen Sie mit dem Team um. Mieten Sie ein Hotelzimmer. Was auch immer. Lesen Sie auch den Abschnitt "Den Teamraum einrichten"

- Ist eine Verbesserung nicht möglich, lassen Sie das Team den Fokus-Faktor für den nächsten Sprint reduzieren und machen Sie deutlich, dass dies aufgrund der lauten und chaotischen Umgebung geschieht. Hoffentlich führt das dazu, dass der Product Owner sich beim oberen Management über die Situation beschwert.

Zum Glück habe ich noch nie damit drohen müssen, mit dem Team auszuziehen. Ich würde es tun, wenn es sein muss :O

11

Leerlauf zwischen Sprints

Im wirklichen Leben kann man nicht ständig sprinten. Zwischen Sprints muss man ausruhen. Denn ein dauerhafter Sprint ist eher mit Jogging zu vergleichen.

Dasselbe gilt für Scrum und die Softwareentwicklung im Allgemeinen. Sprints sind ziemlich intensiv. Als Entwickler bleibt einem kaum Zeit Luft zu holen. Jeden Tag muss man an diesen nervigen Meetings teilnehmen und jedem Rechenschaft ablegen, was man am Vortag erledigt hat. Kaum jemand wird sagen wollen, dass er den größten Teil des Tages mit den Füßen auf dem Tisch Blogs gelesen und Cappuccino getrunken hat.

Neben der eigentlichen Erholungszeit gibt es noch einen anderen guten Grund etwas Leerlauf zwischen Sprints zu haben. Die Köpfe der des Produkt Owner und der Kollegen sind nach der Sprint-Vorführung und der Retrospektive mit Ideen und Informationen voll gestopft, die sie erst einmal verdauen müssen. Würden sie sofort wieder losrennen und mit der Planung des nächsten Sprints beginnen, gäbe es kaum Gelegenheit, aus den Erfahrungen zu lernen und der Product Owner hätte keine Zeit, seine Prioritäten zu überarbeiten.

Schlecht:

Montag

09-10: Vorführung Sprint 1
10-11: Retrospektive Sprint 1

13-16: Planung Sprint 2

Wir versuchen, etwas Leerlauf vor dem Start eines jeden Sprints einzuführen (genauer gesagt zwischen der Sprint-Retrospektive und dem Sprint-Planungs-Meeting). Das gelingt uns allerdings nicht immer. Zumindest versuchen wir zu vermeiden, dass die Retrospektive und das

Sprint-Planungsmeeting am selben Tag stattfinden. Alle sollten wenigstens eine sprintfreie Nacht gut geschlafen haben, bevor der neue Sprint startet.

Besser:

Montag	Dienstag
09-10: Vorführung Sprint 1 10-11: Retrospektive Sprint 1	9-13: Planung Sprint 2

Noch besser:

Freitag	Samstag	Sonntag	Montag
09-10: Vorführung Sprint 1 10-11: Retrospektive Sprint 1			9-13: Planung Sprint 2

Eine andere Möglichkeit ist die Einführung von so genannten Labortagen (oder wie Sie diese auch immer nennen mögen). Dabei handelt es sich um Tage, an denen Entwickler eigentlich machen können, worauf sie Lust haben. (OK, ich gebe es zu, inspiriert von Google). Zum Beispiel, um sich über die neuesten Tools und APIs schlau zu machen, um sich auf eine Zertifizierung vorzubereiten, schlaue Diskussionen mit Kollegen zu führen oder an einem Hobbyprojekt zu programmieren.

Unser Ziel ist es, einen Labortag zwischen allen Sprints zu platzieren. So schaffen wir eine natürliche Erholungsphase zwischen den Sprints - und das Team hat die Möglichkeit sein Wissen auf dem aktuellem Stand zu halten. Außerdem macht eine solche Regelung Sie attraktiv für Neubewerber.

Am Besten?

Donnerstag	Freitag	Samstag	Sonntag	Montag
09-10: Vorführung Sprint 1 10-11: Retrospektive Sprint 1	LABOR-TAG			9-13: Planung Sprint 2

Wir haben momentan einen Labortag im Monat. Den ersten Freitag im Monat, um es genau zu sagen. Warum nicht zwischen Sprints? Nun, ich hatte das Gefühl, dass es wichtig war, für die ganze Firma einen Labortag zur gleichen Zeit einzurichten. Ansonsten würden die Leute diesen Tag nicht ernst nehmen. Und da wir die Sprints der verschiedenen Produkte

noch nicht parallel laufen, musste ein Sprint-unabhängigen Rhythmus gefunden werden.

Vielleicht versuchen wir demnächst, die Sprints aller Produkte zu synchronisieren (also gleiche Sprint Starts und Enden für alle Produkte und Teams). Dann würden wir auf jeden Fall einen Labortag pro Sprint etablieren.

12

Wie wir Releases und Festpreisprojekte planen

Manchmal müssen wir mehr als einen Sprint im Voraus planen. Das passiert oft in Verbindung mit Festpreisangeboten, bei denen wir weiter vorausschauen *müssen*, um nicht mehr zu versprechen als wir leisten können. Releaseplanung bedeutet für uns normalerweise, eine Antwort auf die folgende Frage zu finden: Wann können wir *spätestens* eine 1.0 Version des neuen Systems ausliefern?

Wenn Sie wirklich etwas über Releaseplanung lernen wollen, dann überspringen Sie dieses Kapitel und besorgen sich das Buch "Agile Estimating and Planning" von Mike Cohn. Ich wünschte ich hätte dieses Buch gelesen, *bevor* wir dieses ganze Zeug selbst herausfinden mussten. Auch wenn sie weniger ausgereift ist, sollte meine Art der Releaseplanung trotz allem ein guter Ausgangspunkt sein,

Definieren Sie Akzeptanzschwellen

Zusätzlich zum normalen Product Backlog definiert der Product Owner verschiedene Akzeptanzschwellen, die festlegen, was die Wichtigkeit eines Backlog-Eintrags vertraglich bedeutet.

Hier ist eine beispielhafte Liste von Akzeptanzschwellen:

- Stories mit der Wichtigkeit >= 100 müssen in Version 1.0 enthalten sein, oder wir müssen eine drastische Vertragsstrafe fürchten.
- Stories mit der Wichtigkeit 50-99 sollten in der Version 1.0 enthalten sein, können aber notfalls auch in einer raschen Nachlieferung abgedeckt werden.
- Stories mit der Wichtigkeit 25-49 sind wichtig, können aber auch in einem Nachfolge-Release 1.1 realisiert werden.
- Stories mit einer Wichtigkeit unter 25 sind spekulativ - und werden eventuell nie gebraucht.

Und hier folgt nun ein beispielhaftes Product Backlog - farbkodiert nach den oben beschriebenen Regeln

Wichtigkeit	Name
130	Banane
120	Apfel
115	Orange
110	Guave
100	Birne
95	Rosine
80	Erdnuss
70	Donut
60	Zwiebel
40	Grapefruit
35	Papaya
10	Blaubeere
10	Pfirsich

Rot = muss in Version 1.0 enthalten sein (Banane-Pfirsich)
Gelb = sollte in Version 1.0 enthalten sein (Rosine-Zwiebel)
Grün = kann später angegangen werden (Grapefruit-Pfirsich)

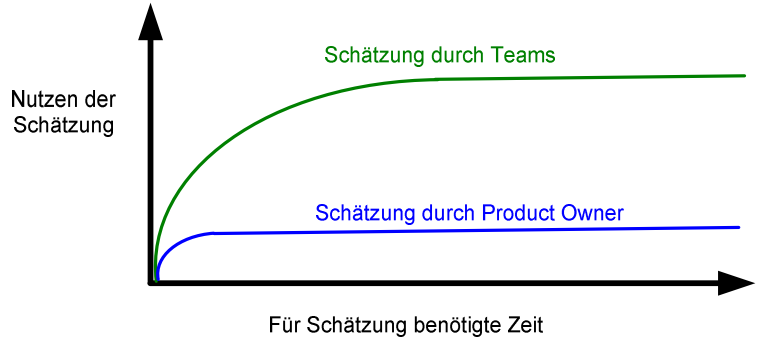
Wenn wir alles von Banane bis Zwiebel bis zum Abgabetermin liefern, sind wir auf der sicheren Seite. Wenn die Zeit knapp wird, können wir noch damit durchkommen, Rosine, Erdnuss, Donut oder Zwiebel wegzulassen. Alles ab Zwiebel ist optional.

Wie man die wichtigsten Einträge schätzt

Für die Releaseplanung benötigt der Product Owner Aufwandschätzungen; zumindest für alle jene Stories die im Vertrag vereinbart wurden. Genau wie bei der Sprint-Planung erhalten wir die wieder in Gemeinschaftsarbeit von Product Owner und dem Team: das Team schätzt, der Product Owner erläutert die Stories und beantwortet Fragen.

Eine Aufwandschätzung ist brauchbar, wenn sie der Realität recht nah kommt; weniger brauchbar ist sie, wenn sie - sagen wir - 30% abweicht. Hat Sie gar keinen Bezug zur Realität, ist sie unbrauchbar.

Den Wert einer Zeitschätzung im Verhältnis zum Urheber und Zeitaufwand, der für Ihre Erstellung nötig war, schätze ich folgendermaßen ein.



Konkret gesagt, heißt das:

- Lassen Sie das Team schatzen
- Begrenzen Sie die Zeit, die sich das Team für die Schätzung nimmt.
- Machen Sie dem Team klar, dass das Ergebnis nur grobe Schätzungen und keine Versprechen sind.

Normalerweise versammelt der Product Owner das alle in einem Raum, stellt ein paar Erfrischungen bereit und erklärt die Aufwandschätzung der z.B. ersten 20 Stories des Product Backlogs zum Ziel des Meetings. Nachdem er jede Story kurz erläutert hat, lässt er das Team an der Schätzung arbeiten. Er bleibt im Raum, um Fragen zu beantworten und um notfalls den Umfang eines Eintrags klarzustellen. Wie schon bei der Sprint-Planung, ist das Feld "Vorführung" sehr nützlich, um Missverständnisse aufzudecken.

Das Meeting sollte in einem engen zeitlichen Rahmen ablaufen, da Teams sonst zu viel Zeit für die Schätzung zu weniger Stories aufwenden. Wenn der Product Owner mehr Zeit investieren möchte, kann er ein zusätzliches Meeting einberufen. Es liegt in der Verantwortung des Teams, dem Product Owner die Auswirkungen des Meetings auf den laufenden Sprint klar zu machen. Nur so wird er lernen, dass auch Aufwandsschätzungen nicht umsonst zu haben sind.

Hier das Ergebnis einer beispielhaften Aufwandschätzung (in Story-Punkten):

Wichtigkeit	Name	Schätzung
130	Banane	12
120	Apfel	9
115	Orange	20
110	Guave	8
100	Birne	20
95	Rosine	12
80	Erdnuss	10
70	Donut	8
60	Zwiebel	10
40	Grapefruit	14
35	Papaya	4
10	Blaubeere	
10	Pfirsich	

Entwicklungsgeschwindigkeit schätzen

In Ordnung, jetzt haben wir eine grobe Zeitschätzung für die wichtigsten Stories. Der nächste Schritt ist die Einschätzung der durchschnittlichen Entwicklungsgeschwindigkeit je Sprint.

Das bedeutet, wir müssen zunächst unseren Fokus-Faktor bestimmen. Lesen Sie dazu den Abschnitt "Wie das Team entscheidet, welche Stories im Sprint enthalten sind".

Der Fokus-Faktor besagt primär, wie viel Zeit das Team mit der Arbeit an den ausgewählten Stories aufbringen kann. Da immer Zeit für ungeplante Aufgaben, dem Wechsel zwischen Aufgaben, der Unterstützung anderer Teams, dem Lesen von Emails, der Reparatur defekter Computers oder politische Diskussionen in der Kaffeeküche aufgewendet wird, liegt er nie bei 100%.

Angenommen, wir legen für das Team einen Fokus-Faktor von 50% fest (das ist recht niedrig, üblicher ist ein Wert von 70%). Und nehmen wir außerdem an, bei einer Teamgröße von 6 sei unsere Sprint-Länge drei Wochen, also 15 Tage.
Jeder Sprint umfasst demnach 90 Manntage, von denen - wegen des Fokus-Faktors von 50% - lediglich 45 für die Arbeit an Stories verwendet werden.

Nach dieser Rechnung beträgt unsere geschätzte Entwicklungsgeschwindigkeit 45 Story-Punkte. Wenn jede Story eine Zeitschätzung von 5 Tagen hätte (was sie nicht haben), könnte das Team im Schnitt neun Stories pro Sprint abliefern.

Alles zu einem Releaseplan zusammenfügen

Jetzt wo wir Aufwände geschätzt und die Entwicklungsgeschwindigkeit (hier 45) berechnet haben, gelingt es leicht, das Product Backlog auf die einzelnen Sprints aufzuteilen:

Wichtigkeit	Name	Schätzung
		Sprint 1
130	Banane	12
120	Apfel	9
115	Orange	20
		Sprint 2
110	Guave	8
100	Birne	20
95	Rosine	12
		Sprint 3
80	Erdnuss	10
70	Donut	8
60	Zwiebel	10
40	Grapefruit	14
		Sprint 4
35	Papaya	4
10	Blaubeere	
10	Pfirsich	

Jeder Sprint beinhaltet so viele Stories wie möglich, ohne jedoch die geschätzte Entwicklungsgeschwindigkeit von 45 zu überschreiten.

Wir sehen, dass wir wahrscheinlich drei Sprints benötigen, um die Muss- und Soll-Features fertig zu stellen. 3 Sprints = 9 Kalenderwochen = 2 Kalendermonate. Ob wir daraus bereits einen Liefertermin ableiten und dem Kunden zusagen können, hängt stark davon ab welche Vertragsart gewählt wurde oder wie festgelegt die Liefergegenstände sind. Wir rechnen für gewöhnlich einen großzügigen Zeitpuffer dazu, um uns gegen schlechte Zeitschätzungen, unerwartete Probleme oder Features abzusichern. So könnten wir uns in diesem Fall auf einen Liefertermin in

drei Monaten festlegen - mit einem Monat "Reserve". Der Vorteil daran wäre, dass wir dem Kunden alle 3 Wochen etwas Brauchbares demonstrieren können und ihm trotzdem ermöglichen, während des Projekts Anforderungen zu ändern (was wiederum von der Art des Vertrages abhängt).

Den Releaseplan anpassen

Die Realität wird sich nicht dem Plan anpassen, demnach müssen wir es anders herum machen. Nach jedem Sprint betrachten wir die tatsächliche Velocity des Sprints. Weicht sie stark von der geschätzten Velocity ab, passen wir die geschätzte Velocity für die folgenden Sprints entsprechend an und aktualisieren den Releaseplan.

Geraten wir dadurch in Probleme, kann der Product Owner, ohne Vertragsbruch zu begehen, mit dem Kunden eine Reduzierung des Lieferumfangs aushandeln. Oder er und das Team erhöhen die Entwicklungsgeschwindigkeit, weil sie ein größeres Hindernis aus dem letzten Sprint aus dem Weg räumen können.

Oder der Product Owner ruft den Kunden an und sagt ihm: "Hi, wir hinken dem Zeitplan ein wenig hinterher, aber ich bin zuversichtlich, dass wir den Termin halten können, wenn wir das aufwändige Feature "Integriertes Pacman-Spiel" weglassen. Wenn Sie möchten, liefern wir es drei Wochen später in einem Folge-Release nach."

Auch wenn das für den Kunden keine guten Nachrichten sind, sind wir ehrlich und lassen ihm die Wahl: eine pünktliche Auslieferung der wichtigsten Features oder eine Verspätung aller Features. Meistens fällt ihm die Entscheidung nicht schwer :o)

13

Wie wir Scrum mit XP kombinieren

Es ist nicht gerade eine gewagte Behauptung dass es sich lohnt, Scrum mit XP (Extreme Programming) zu kombinieren. Da die meisten Quellen im Internet dies bekräftigen, erspare ich mir weiter auf die Gründe einzugehen. Eines möchte ich aber noch anmerken.

Die Praktiken von Scrum konzentrieren sich auf Management und Organisation, während XP sich mit dem eigentlichen Programmieren beschäftigt. Sie arbeiten Hand in Hand, einfach weil sie verschiedene Aspekte behandeln und sich gut kombinieren lassen. In diesem Sinne möchte auch ich zum empirischen Beweis beitragen, dass Scrum und XP zusammen eine lohnende Kombination sind!

Ich werde auf viele der wertvolleren XP-Praktiken eingehen und berichten, wie sie in unserem Arbeitsalltag zum Einsatz kommen. Nicht jedem Team ist es gelungen, alle Praktiken zu etablieren. Wir haben mit den meisten Aspekten der Kombination Scrum/XP herumexperimentiert. Einige Praktiken des Extreme Programming wie "Gesamtes Team", "Benutzergeschichten" und "Planungsspiel" wurden direkt in Scrum übernommen und sind demnach mehrfach definiert. In diesen Fällen halten wir uns der Einfachkeit halber an Scrum.

Programmierung in Paaren

Eines unserer Teams arbeitet seit kurzem nach dieser Methode - und es funktioniert eigentlich recht gut. Die anderen Teams programmieren noch kaum in Paaren. Nachdem ich es aber in einem Team für ein paar Sprints ausprobiert habe, möchte ich auch sie ermuten, es einfach mal auszuprobieren.

Hier unsere aktuellen Erkenntnisse zur Programmierung in Paaren:

- Programmierung in Paaren verbessert tatsächlich die Codequalität

- Programmierung in Paaren fokussiert das Team (zum Beispiel dadurch, dass der Kerl hinter einem fragt, "Hey brauchen wir das Zeug da wirklich für den aktuellen Sprint?").
- Erstaunlicherweise haben viele Programmierer, die sich gegen die Programmierung in Paaren aussprechen, es noch nie probiert. Und lernen es schnell zu schätzen, wenn sie erst einmal angefangen haben. Programmierung in Paaren ist anstrengend und sollte nicht den ganzen Tag praktiziert werden.
- Es ist gut, häufig die Paare durchzuwechseln.
- Programmierung in Paaren fördert den Wissensaustausch in der Gruppe. Sogar überraschend schnell.
- Manche Leute können sich einfach nicht auf Programmierung in Paaren einlassen. Setzen Sie keinen exzellenten Programmierer auf die Strasse, nur weil er Programmierung in Paaren nicht mag.
- Code Review ist eine akzeptable Alternative zur Programmierung in Paaren
- Der "Copilot" (derjenige, der gerade nicht die Tastatur benutzt), sollte einen eigenen Computer haben. Nicht fürs Entwickeln, sondern kleinere Recherche und Nachschlageaufgaben, wenn der für kleinere Rechercheaufgaben, dem Such, suchen in Dokumentationen, z.B. wenn der "Pilot" (derjenige an den Tasten) nicht weiter kommt.
- Zwingen Sie niemandem zur Programmierung in Paaren. Ermutigen Sie die Leute dazu, geben Sie ihnen die richtigen Werkzeuge - aber lassen Sie ihnen auch die Zeit sich damit anzufreunden.

Testgetriebene Entwicklung

Amen! Für mich ist sie wichtiger als Scrum und XP zusammengenommen. Man kann mir mein Haus, meinen Fernseher und meinen Hund nehmen, nicht aber meinen Glauben an Testgetriebene Entwicklung. Wenn Sie Testgetriebene Entwicklung nicht mögen, sollten Sie mich gar nicht erst ins Gebäude lassen. Denn ich werde nichts unversucht lassen, sie doch hinein zu schmuggeln :o)

Hier ein Abriss über Testgetriebene Entwicklung in 10 Sekunden:

> *Testgetriebene Entwicklung bedeutet folgendes: Erst schreibt man einen automatisierbaren Test, danach gerade mal soviel Anwendungscode, um diesen Test zu erfüllen. Anschließend wird der Code hauptsächlich überarbeitet, z.B. um die Lesbarkeit zu verbessern oder*

redundanten Code zu beseitigen. Und dann das ganze Prozedere wieder von vorn..

Einige Überlegungen zur testgetriebenen Entwicklung (TDD) :

- TDD ist schwierig. Egal wie viel sie auch schulen, coachen und demonstrieren. Bis ein Programmierer es wirklich versteht hat, vergeht eine Weile. Oft ist es erst die Programmierung im Paar mit einem TDD-erfahrenen Entwickler, die einen Testgetriebene Entwicklung wirklich kapieren lässt. Hat man es aber erst einmal kapiert, ist man dauerhaft infiziert und will kaum noch anders arbeiten
- TDD hat einen durchweg positiven Einfluss auf die Systemarchitektur.
- Es dauert eine Weile, bis man TDD effektiv bei Neuentwicklungen einsetzen kann, insbesondere für Blackbox-Integrationstests. Aber der Aufwand zahlt sich schnell aus.
- Gewährleisten Sie eine mühelose Testentwicklung, indem Sie z.B. die richtigen Tools bereitstellen, die Kollegen fortbilden und Ihre APIs um notwenige Hilfs- oder Basisklassen erweitern.

Folgende Tools für die Testgetriebene Entwicklung kommen bei uns zum Einsatz:

- jUnit / httpUnit / jWebUnit. Wir überdenken den Einsatz von TestNG und Selenium.
- HSQLDB als eine integrierte, in-memory Datenbank für Testzwecke
- Jetty als ein integrierter in-memory Web Container für Testzwecke
- Cobertura zum Messen der Testabdeckung
- das Spring Framework für die Konfiguration der unterschiedlichen Test Fixtures (mit und ohne Mock-Objekte, mit externer oder in-memory Datenbank, usw.)

TDD bei Neuentwicklungen

Wir setzen TDD bei jeder Neuentwicklung ein, auch wenn das die Startphase eines Projekts verlängert (z.B. weil wir zusätzliche Tools und Vorarbeiten für die Testumgebung brauchen). Verglichen mit den großen Vorteilen ist dieser Mehraufwand nicht der Rede wert und keinesfalls eine Ausrede, TDD nicht einzusetzen.

TDD bei bestehendem Code

Testgetriebene Entwicklung ist so schon schwierig, sie aber auf einer Codebasis zu betreiben, die nicht von Anfang an mit TDD gebaut wurde ... ist wirklich sehr schwierig! Warum? Nun, darüber könnte ich mich

lange auslassen, aber das spare ich mir für einen eigenen "Frontbericht" zum Thema TDD auf :o)

Wir haben lange dafür gebraucht, bei einem unserer komplexeren Systeme automatisierte Integrationstests einzuführen. Die Codebasis existierte bereits eine Weile und war in einem ziemlich desolaten Zustand. Es fehlten jegliche Tests.

Für jedes Release des Systems war ein eigenes Team von Testern nötig, um eine Reihe komplexer Regressions- und Performancetests durchzuführen. Dass die Regressionstests hauptsächlich manuell durchgeführt wurden, hat unseren Entwicklungs- und Releasezyklus spürbar verlangsamt.

Unser Ziel war es demnach, diese Tests zu automatisieren. Nach einem monatelangen, frustrierenden Kraftakt waren wir unserem Ziel kaum näher gekommen. Danach haben wir eingesehen, dass wir nicht auf manuelle Regressionstests verzichten können - und unsere Herangehensweise geändert. Ab sofort stellten wir die Frage: "Wie können wir den manuellen Testprozess verkürzen?". Es ging damals um eine Spielentwicklung und wir fanden heraus, dass das Testteam die meiste Zeit mit ziemlich trivialen Einrichtungsaufgaben verbrachte, wie z.B. dem Erzeugen von Test-Turnieren im Backend und dem Warten auf deren Spielstart. Wir entwickelten kleine, einfach nutzbare Tastaturkürzel und Skripte, die solche Routineaufgaben erledigten, so dass die Tester sich wirklich aufs Testen konzentrieren konnten.

Der Aufwand hat sich wirklich gelohnt! Vermutlich hätten wir das bereits vom Anfang an machen sollen. Aber wir waren so erpicht darauf, die Tests vollständig zu automatisieren, dass wir vergessen hatten, schrittweise vorzugehen und zunächst das manuelle Testen effizienter zu machen.

Wir haben unsere Lektion gelernt: Verzichten Sie zunächst auf eine Automatisierung ihrer manuellen Regressionstest, außer dies ist wirklich einfach. Schaffen Sie sich stattdessen Mittel, die die Durchführung der Tests erleichtern. Danach können Sie über eine Automatisierung der Tests nachdenken.

Inkrementelle Architektur

Das heißt, man soll die Architektur zu Beginn lieber einfach halten und kontinuierlich verfeinern, anstatt schon am Anfang alles richtig machen und festlegen zu wollen. Darin sind wir mittlerweile recht gut und reservieren uns z.B. ausreichend Zeit für das Refaktorieren und die

Verbesserung der bestehenden Architektur. Nur selten stecken wir viel Zeit in eine vorab durchspezifizierte, alles umfassende Gesamtarchitektur.

Manchmal vergeigen wir es natürlich und erlauben es windigen Architekturen sich festzusetzen. Dann artet das Refactoring meist zu einem größeren Projekt aus. Im Großen und Ganzen sind wir aber recht zufrieden.
Die kontinuierliche Verbesserung des Designs ist ein beinahe schon automatischer Nebeneffekt der testgetriebenen Entwicklung.

Fortlaufende Integration

Die meisten unserer Produkte haben eine recht ausgereifte Laufzeitumgebung zur fortlaufenden Integration auf der Basis von Maven und QuickBuild. Das ist außerordentlich nützlich und zeitsparend. Es ist die ultimative Antwort auf den nur allzu bekannten Satz "Aber auf meinem Rechner läuft es". Unser System zur fortlaufenden Integration dient als eine Art Richter bzw. Referenzpunkt, von dem aus sich der Gesundheitszustand all unserer Codebasen prüfen lässt. Wann immer etwas ins Versionskontrollsystem eincheckt wird, erwacht es, baut auf einem frei zugänglichen Server die Software von Grund auf neu und lässt alle Tests ablaufen. Geht dabei etwas schief, sendet das System an das Team eine Email mit der Nachricht, dass der Buildvorgang fehlgeschlagen ist - inklusive Angaben, wessen Code-Änderung zu dem Problem geführt hat und einem Link zu den Testresultaten.
Jede Nacht baut das System die Software neu und veröffentlicht in unserem Informations-Portal Binär-Archive (ears, wars, etc), Dokumentationen, Auswertungen zu Tests, Testabdeckung und Code-Abhängigkeiten. Einige unserer Produkte werden sogar automatisch auf einer Testumgebung installiert.

So *aufwändig* es auch war, dies alles zum Laufen zu bringen, hat sich doch jede Minute ausgezahlt.

Gemeinsames Code- Eigentum

Wir ermutigen alle zu einer gemeinschaftlich verantworteten Codebasis - also der Abkehr von den individuellen Modulen und Komponenten - aber nicht alle Teams haben diese Praxis schon eingeführt. Wir haben festgestellt, dass das Programmieren in Paaren mit häufig wechselnden Partnern bereits in einem hohen Maß zu gemeinsamer Verantwortlichkeit führt. Teams, die diese besonders gut etabliert haben, erweisen sich als sehr robust. So müssen sie z.B. nur selten einen Sprint unterbrechen, weil eine Schlüsselperson krank geworden ist.

Informativer Arbeitsbereich

Die Schreibtafeln und die große Wandfläche, die wir all unseren Teams zur Verfügung stellen, werden rege genutzt. In der Mehrzahl der Büros sind die Wände gepflastert mit allerlei Produkt- und Projektinformationen. Um dem Problem Herr zu werden, dass sich dort mit der Zeit veralteter Kram ansammelt, wollen wir vielleicht pro Team eine Art Hausmeister-Rolle einführen.
Auch wenn bisher nicht alle Teams davon Gebrauch machen, ermutigen wir auch sie dazu, eine Aufgabenwand zu führen. Lesen Sie dazu mehr im Abschnitt "Wie wir den Teamraum einrichten".

Programmierstandards

Vor kurzem haben wir damit begonnen, Programmierstandards zu definieren. Das ist sehr sinnvoll und ich wünschte, wir hätten schon eher damit begonnen. Man braucht nicht viel Zeit dafür, sondern fängt einfach an und lässt sie wachsen. Legen Sie nur Dinge fest, die nicht ohnehin schon für jeden selbstverständlich sind und verweisen sie, wenn möglich, auf vorhandene Dokumente.

Die meisten Entwickler haben ihren eigenen Programmierstil. Dieser zeigt sich in Details, wie der Ausnahmebehandlung, Kommentaren oder der Entscheidung, wann sie NULL zurückgeben... Manchmal spielen die Unterschiede keine Rolle. Ein anderes Mal führen sie zu einer hochgradig widersprüchlichen Architektur und schwer lesbarem Code.
Programmierstandards sind immer dann von hohem Nutzen, wenn Sie Bereiche regeln, die entscheidend sind.

Hier sind ein paar Beispiele aus unseren Programmierstandards:

- Jede Regel darf verletzt werden, wenn es einen guten Grund gibt. Ist das der Fall, denke daran es zu dokumentieren.
- Befolge im Normalfall die Sun Code Conventions: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- Vergiss niemals den Stack Trace zu loggen oder die Exception weiter zu delegieren. log.debug() reicht aus, achte nur darauf, dass der Stack Trace nicht verloren geht.
- Verwende Setter-basierte Dependency Injection um Klassen zu entkoppeln (außer, eine enge Kopplung ist erwünscht)
- Vermeide Abkürzungen. Übliche Abkürzungen wie DAO sind allerdings in Ordnung.
- Methoden mit Collections oder Arrays als Rückgabewert, sollten nicht NULL, sondern leere Collections oder Arrays zurückgeben.

Nachhaltiges, schwungvolles Arbeitstempo

Viele Bücher über Agile Softwareentwicklung mahnen an, dass extensive Überstunden kontraproduktiv für die Softwareentwicklung sind.
Nach einigen unfreiwilligen Experimenten zum Thema kann ich dem nur aus vollem Herzen beipflichten!

Vor ungefähr einem Jahr häufte unser größtes Team eine enorme Anzahl von Überstunden an. Die Qualität des Codes war furchtbar und das Team musste fast ständig irgendwelche "Brände löschen". Das Testteam (ebenfalls unter Überstundenlast) war weit davon entfernt gründliche Qualitätssicherung machen zu können. Unsere Benutzer waren verärgert und die Fachpresse ließ kein gutes Haar an uns.

Nach ein paar Monaten gelang es uns, die Arbeitsbelastung der Mitarbeiter auf ein erträgliches Maß herabzusetzen. Die Leute arbeiteten, bis auf bei gelegentlichen Projektkrisen, wieder ihr normales Pensum. Und welche Überraschung: Produktivität und Qualität verbesserten sich spürbar. Auch wenn die Verringerung der Arbeitslast sicher nicht der alleinige Besserungsgrund war, so sind wir doch überzeugt, dass sie einen großen Anteil daran hatte.

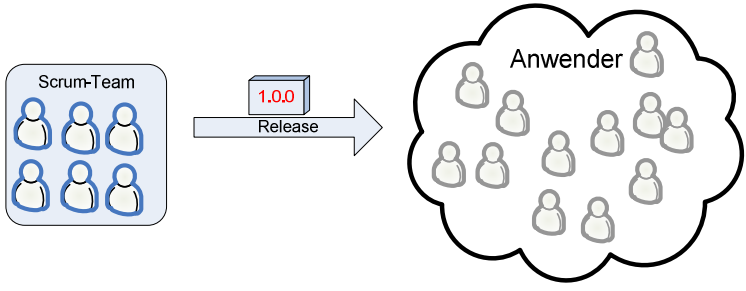
14

Wie wir testen

Jetzt kommen wir zum schwierigsten Teil. Ich bin nicht sicher, ob das an Scrum liegt oder Testen generell der schwierigste Teil bei der Softwareentwicklung ist.
Die Art und Weise wie getestet wird, unterscheidet sich in verschiedenen Unternehmen wahrscheinlich stark. Zum Beispiel in der Anzahl der Tester, dem Automatisierungsgrad der Tests, der Systemumgebung (Auslieferung als Webapplikation oder Kaufsoftware im Laden), Dauer der Release-Zyklen und Kritikalität der Produkte (Software für Blogs oder Flugzeugsteuerung).
Wir haben mit dem Thema Scrum und Test ordentlich herumexperimentiert, und ich möchte hier beschreiben was wir getan und daraus gelernt haben.

Um eine Phase für Abnahmetests werden Sie wahrscheinlich nicht herumkommen

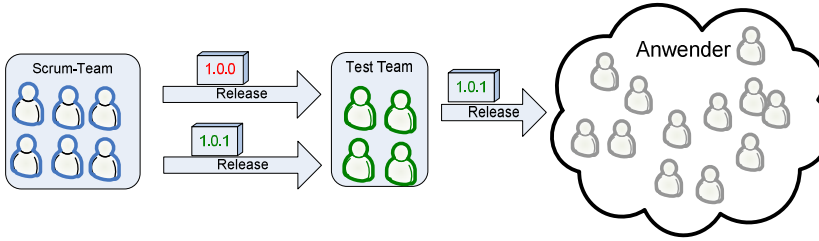
In der perfekten Scrum-Welt ist es aber doch so vorgesehen, dass ein jeder Sprint eine potentiell auslieferbare Version ihres Systems erzeugt. Die bräuchten wir doch nur noch auszuliefern, oder?



Falsch.

Unserer Erfahrung nach klappt das selten. Es wird immer noch hässliche Bugs geben. Wenn Ihnen Qualität auch nur im Geringsten wichtig ist, benötigen Sie eine gewisse Zeit für manuelle Abnahmetests. Solche, bei

denen Tester außerhalb des Teams das System mit Tests bombardieren. Tests für die das Team keine Zeit oder nicht die richtige Hardware zur Verfügung hatte oder an die sie im Traum nicht gedacht hätten. Diese Tester benutzen das System genau auf die Art wie es auch Endbenutzer tun würden - falls das System überhaupt für menschliche Benutzer gedacht ist.



Das Test-Team findet Bugs, das Scrum-Team erstellt Bugfix-Releases und früher oder später – früher wäre mir lieber - können Sie eine fehlerbereinigte Version 1.0.1. an die Endbenutzer ausliefern und die windige 1.0 Version ersetzen.

Wenn ich hier von einer Abnahmetest-Phase spreche, meine ich den gesamten Prozess von Testen, Bugfixen und anschließender Neuauslieferung. Solange, bis schließlich eine Version in Produktionsreife vorliegt.

Minimieren Sie die Abnahmetestphase

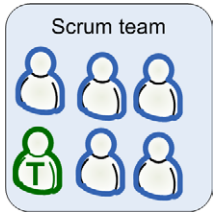
Die Abnahmetestphase schmerzt und fühlt sich ziemlich un-agil an. Auch wenn wir nicht auf sie verzichten können, haben wir immer noch die Möglichkeit sie zu minimieren. Das heißt, die für die Phase reservierte Zeit so kurz wie möglich zu halten. Das erreichen wir, indem wir:

- die Code-Qualität des Scrum-Teams maximieren
- die Effizienz bei manuellen Tests maximieren, also z.B. mit den besten Testern arbeiten, ihnen die besten Werkzeuge zur Verfügung stellen und sicherstellen, dass zeitintensive Aufgaben, die automatisiert werden können, sofort gemeldet werden .

Und wie maximieren wir die Code-Qualität des Scrum-Teams? Nun, da gibt es viele Wege. Hier sind zwei, die bei uns recht gut funktionieren:

- Setzen Sie Tester direkt in das Scrum-Team
- Nehmen Sie sich pro Sprint weniger vor

Tester im Scrum-Team verbessern die Qualität



Ich höre schon die zwei Einwürfe:

- "Das ist doch selbstverständlich! Schließlich sollen Scrum-Teams *funktionsübergreifend* besetzt sein."
- "Spezielle Rollen sollte es in einem Scrum-Teams gar nicht geben. Jemanden, der *nur* testet, können wir uns nicht leisten."

Lassen Sie mich klarstellen. Mit „Tester“ meine ich eher „eine Person, deren Hauptkompetenz das Testen ist" - als eine Person, deren Rolle sich aufs Testen beschränkt.

Entwickler sind oft grauenvolle Tester. Insbesondere dann, wenn sie ihren *eigenen* Code testen.

Der Tester führt die Endabnahme durch

Zusätzlich zu seinen Aufgaben als Teammitglied hat der Tester eine wichtige Verantwortung. Er macht die Endabnahme. In unseren Sprints gilt nichts als erledigt bis es vom Tester abgenommen wurde. Ich habe oft erlebt, dass Entwickler etwas als fertig deklarieren, das noch weit entfernt von der Fertigstellung ist. Selbst wenn Sie die Bedeutung von "Erledigt" klar definiert haben - der Abschnitt "Definition von Fertig" beschreibt warum diese so wichtig ist - erinnern sich Entwickler oftmals nicht mehr daran. Wir Programmierer sind ungeduldige Leute und würden am liebsten so schnell wie möglich schon mit der nächsten Aufgabe weitermachen.

Wie kann unser Mr.T (unser Tester) also wissen, dass etwas wirklich fertig ist? Nun, erst einmal sollte er es testen (Überraschung)! Denn in vielen Fällen stellt sich heraus, dass das was ein Entwickler als "erledigt" bezeichnet, noch nicht einmal testbar ist. Zum Beispiel, weil es nicht eingecheckt , nicht auf dem Testserver installiert wurde, oder gar nicht erst gestartet werden kann. Sobald Mr. T ein Feature getestet hat, sollte er mit dem Entwickler eine Abnahme-Checkliste durchgehen, wenn Sie so etwas haben. Wenn diese Liste z.B. eine Releasenotiz erfordert, muss Mr.

T prüfen, ob diese vorhanden ist. Gibt es, was bei uns nur selten der Fall ist, für ein Feature eine schriftliche Spezifikation, zieht Mr. T diese zum Test heran.

Das gute an diesem Vorgehen ist, dass das Team somit den perfekten Organisator für die Sprint-Vorführung hat.

Was macht der Tester, wenn es nichts zu testen gibt?

Oft wird die Frage gestellt, was Mr. T macht, wenn es gerade nichts zu testen gibt. Es kann eine Woche dauern, bis das Team die erste Story fertig stellt. Was soll der Tester *während dieser Zeit* machen?

Tja - in erster Linie, die *Tests vorbereiten*. Also Testspezifikationen schreiben, eine Testumgebung aufsetzen, usw. Soviel, dass es keine Verzögerungen gibt, wenn ein Entwickler etwas Testbares fertig hat und Mr. T *sofort* mit dem Testen beginnen kann.

Praktiziert das Team testgetriebene Entwicklung, wird vom allerersten Tag an Testcode geschrieben. Der Tester sollte in diesem Fall mit den Entwicklern in Paaren programmieren. Er sollte dies sogar dann tun, wenn er nicht programmieren kann. Dann eben nur in der Rolle des Copiloten, der dem Anderen das Tippen überlässt. Meistens fallen guten Testern andere Arten von Tests ein als guten Programmierern. So *ergänzen* sich beide.

Wird kein TDD praktiziert oder beansprucht das Testen nicht die ganze Zeit des Testers, sollte er das Team nach besten Kräften unterstützen das Sprint-Ziel zu erreichen. Genauso, wie jedes andere Teammitglied. Es ist großartig wenn der Tester programmieren kann, andernfalls muss das Team alle programmierfremden Aufgaben identifizieren.

Obwohl es eine Menge anderer Aufgaben gibt, beschränken sich Teams beim Zerlegen von Stories oft auf *reine Programmieraufgaben*. Nehmen Sie sich aber die Zeit auch *andere* Aufgaben zu identifizieren, stehen die Chancen gut, dass Mr. T selbst dann mit anpacken kann, wenn es nichts gerade nichts zum Testen gibt und er nicht mitprogrammieren kann.

Hier ein paar beispielhafte programmierfremde Aufgaben die häufig in Sprints anfallen:

- Aufsetzen der Testumgebung
- Abklären der Anforderungen
- Das Thema Deployment mit dem IT-Betrieb besprechen

- Verfassen der Deployment-Dokumente (Release-Notizen, RFCs und was sonst in Ihrer Firma so erforderlich ist).
- Kontakt zu externen Ressourcen halten (zum Beispiel GUI-Designern).
- Build-Skripte verbessern.
- Neue Stories in Aufgaben herunter brechen.
- Sich um Antworten auf wichtige Kernfragen der Entwickler kümmern.

Was aber machen wir im Gegenzug wenn Mr. T zum Engpass wird? Angenommen, es ist der letzte Tag des Sprints, auf einen Schlag werden viele Dinge abgeschlossen, aber Mr. T hat keine Chance, das alles noch zu testen. Was nun? Nun, wir können beispielsweise alle im Team zu Mr. Ts Assistenten machen. Er entscheidet dann, was er selbst machen muss und delegiert die einfacheren Tests an die anderen. Genau das ist ja der Zweck von funktionsübergreifenden Teams!

Auch wenn Mr. T eine Sonderrolle im Team *hat*, darf er jederzeit andere Aufgaben übernehmen, und das Team darf ihn jederzeit bei seinen Aufgaben unterstützen.

Die Qualität verbessern, indem Sie sich pro Sprint weniger vornehmen

Das gilt eigentlich bereits für die Sprint-Planung: packen Sie nicht zu viele Stories in den Sprint! Wenn Sie Qualitätsprobleme haben, lange Phasen für Abnahmetests - nehmen Sie sich weniger vor! Das führt fast automatisch zu besserer Qualität, kürzeren Testphasen und weniger kundenkritischen Bugs. Außerdem steigert es langfristig die Produktivität des Teams, wenn es sich auf neue Funktionen konzentrieren kann, statt sich andauernd um Fehler im alten Code zu kümmern.

Es ist fast immer billiger, weniger aber dafür robust zu bauen, als viel zu bauen und anschließend panisches Bugfixing zu betreiben.

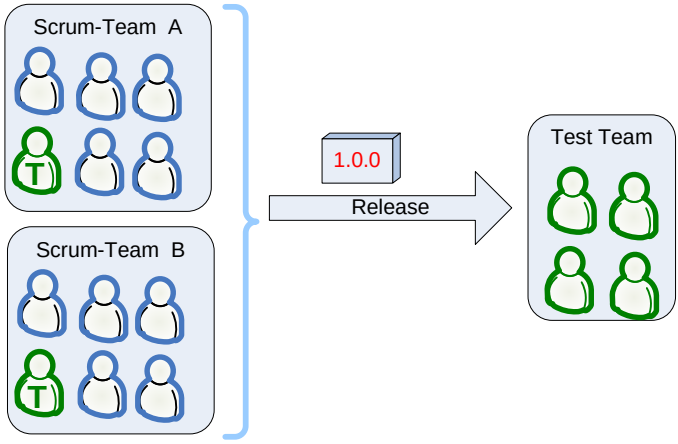
Sind Abnahmetests Teil des Sprints?

Bei diesem Thema sind wir hin und her gerissen. Einige Teams schließen die Abnahmetests in den Sprint mit ein. Die meisten, aus zwei Gründen, allerdings nicht:

- Ein Sprint ist zeitlich beschränkt. Abnahmetests (umfassen meiner Meinung nach Fehlerbehebung und Neuauslieferung) sind schwer zeitlich zu begrenzen. Was machen Sie, wenn die Zeit knapp wird und Sie immer noch mit einem kritischen Fehler kämpfen? Liefern Sie die Software mit einem kritischen Fehler

aus? Oder warten Sie bis zum nächsten Sprint? Da beides meist nicht akzeptabel ist, lassen wir die manuellen Abnahmetests lieber ganz aus dem Sprint heraus.

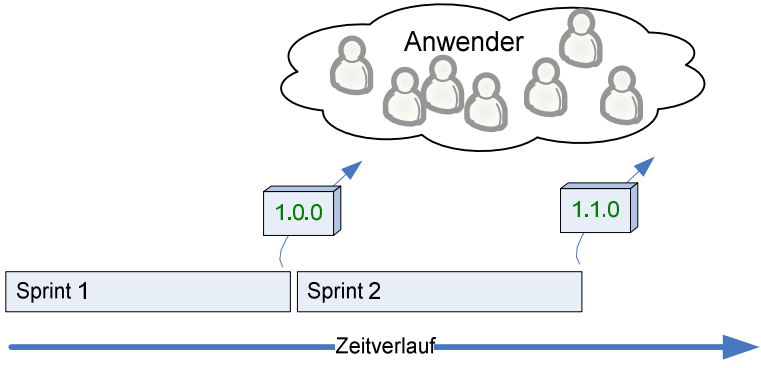
§ Arbeiten zwei Scrum Teams am gleichen Produkt, müssen deren Ergebnisse erst integriert werden, bevor manuelle Abnahmetest durchgeführt werden können. Selbst wenn jedes Team eigene Abnahmetests durchgeführt hat, bräuchten Sie ein zusätzliches Team zum Testen nach der Integration.



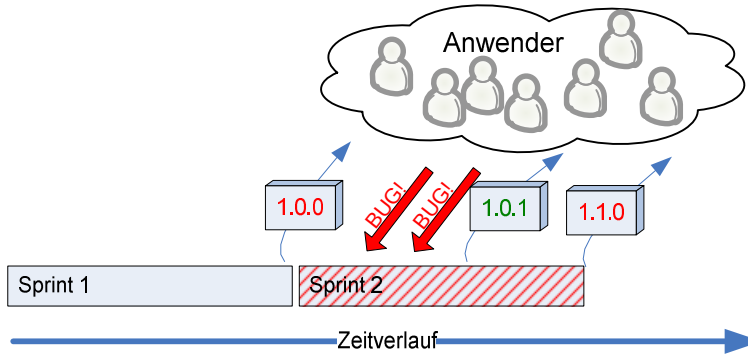
Auch wenn dies mitnichten eine perfekte Lösung ist, ist sie für uns in den meisten Fällen gut genug.

Sprint-Zyklen und Abnahmetest-Phasen

In einer perfekten Scrum-Welt bräuchte es keine Abnahmetests, einfach weil Scrum-Teams mit jedem Sprint eine neue auslieferbare Version des Systems produzieren würden.



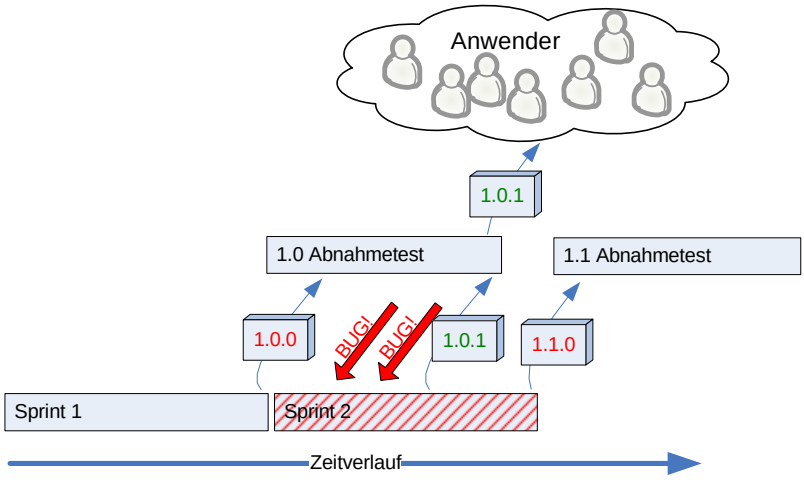
Realistischer ist allerdings dieses Bild:



Nach Sprint 1 wird eine fehlerbehaftete Version 1.0.0 ausgeliefert. Während Sprint 2 werden Sie mit Bug-Meldungen überschüttet, das Team ist ausschließlich mit Fehlerbehebung beschäftigt und muss während des Sprints das Bugfix-Release 1.0.1. ausliefern. Am Ende vom Sprint 2 liefern Sie ein Feature-Release 1.1.0 aus. Da das Team wegen der Probleme beim letzten Release noch weniger Zeit gute Arbeit abzuliefern, enthält dieses Release noch mehr Bugs. Und so geht es immer weiter...

Die diagonalen roten Linien in Sprint 2 symbolisieren Chaos.

Nicht besonders schön, oder? Leider bleibt das Problem sogar dann bestehen, wenn Sie ein eigenes Team für Abnahmetests haben. Der Unterschied ist dann nur, dass die meisten Bugs vom Abnahmetest-Team, anstatt von aufgebrachten Benutzern gemeldet werden. Aus betriebswirtschaftlicher Sicht sicher ein großer Unterschied; nicht jedoch für Entwickler. Einmal davon abgesehen, dass Tester meistens weniger aggressiv sind als Endbenutzer.



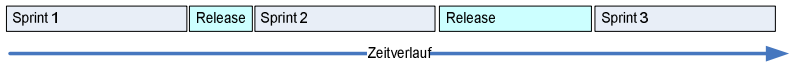
Auch wenn wir für das Problem keine einfache Lösung finden konnten, haben wir doch mit verschiedenen Herangehensweisen experimentiert.

Erstens und wie schon oft erwähnt: Verbessern Sie die Code-Qualität des Scrum-Teams. Die Kosten zur Fehlerbeseitigung sind zu Beginn eines Sprints beträchtlich niedriger als bei nachträglichem Debugging.

Wie auch immer, die Tatsache bleibt bestehen dass es bei allen Bemühungen nach dem Sprint immer noch Bug-Reports geben wird. Wie gehen wir damit um?

Ansatz 1: „Es wird erst mit dem Neuen begonnen, wenn die bestehende Funktionalität ausgeliefert ist.“

Klingt gut, nicht wahr? Spüren Sie auch dieses warme, angenehme Gefühl?
Wir waren mehrmals kurz davor, diese Methode zu übernehmen und haben hübsche Plane für die Umsetzung erdacht. Doch jedes Mal haben uns die Nachteile wieder dazu bewogen, die Idee zu verwerfen. Nämlich die, dass wir zwischen den Sprints zusätzliche, zeitlich nicht beschränkbare Auslieferungsphasen hätten einführen müssen, in denen wir testen und debuggen würden, bis wir bereit zum Ausliefern wären.



Uns gefielen aber die zeitlich nicht beschränkbaren Release-Phasen nicht, wohl hauptsächlich, weil das den Rhythmus unserer Sprints durcheinander bringen würde.

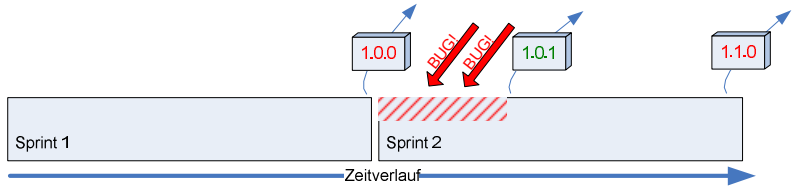
So könnten wir nicht mehr einfach alle drei Wochen einen neuen Sprint starten. Außerdem löst es unser Problem nicht vollständig, denn selbst mit einer Release-Phase würden gelegentlich Bugs "außer der Reihe" gemeldet werden, auf die wir reagieren müssen.

Methode 2: "Es ist in Ordnung, Neues anzufangen. Priorität hat aber die Auslieferung fertiger Funktionalität."

Dies ist unsere bevorzugte Methode - zumindest im Moment.

Einfach gesagt, wir beenden einen Sprint und beginnen den nächsten. Allerdings reservieren wir etwas Zeit für die Behebung von Fehlern aus dem letzten Sprint. Wird ein Sprint arg in Mitleidenschaft gezogen, weil wir viel Zeit für die Behebung alter Fehler aufwenden, prüfen wir, warum das passiert ist und wie wir die Qualität verbessern können. Außerdem sorgen wir dafür, dass Sprints lang genug sind, einige Bugfixes aus dem vorhergehenden Sprint aufzunehmen.

Schrittweise, über mehrere Monate hinweg, ist der Zeitanteil zur Behebung solcher Fehler zurückgegangen. Zudem ist es gelungen, beim Bugfixing immer weniger Personen zu involvieren, und nicht jedes Mal das gesamte Team zu unterbrechen. Jetzt sind wir auf einem annehmbareren Stand angelangt.



Um dem Aufwand für solches Bugfixing Rechnung zu tragen, setzen wir bei der Sprint-Planung den Fokus-Faktor entsprechend niedrig. Mit der Zeit sind die Teams mit ihrer Schätzung immer besser geworden. Die Metrik "Entwicklungsgeschwindigkeit" (engl. Velocity) hilft dabei übrigens sehr (siehe dazu auch den Abschnitt "Wie entscheidet das Teams, welche Stories in dem Sprint enthalten sind?").

Schlechter Ansatz – „Sich auf die Neuentwicklung konzentrieren“

Letztlich hieße das, mit neuen Funktionen anzufangen, *anstatt alte auszuliefern*. Wer könnte so etwas wollen? Wir haben diesen Fehler am Anfang recht häufig gemacht. Ich bin mir sicher, anderen Firmen geht es ähnlich. Die Ursache derartiger Verirrungen ist Stress. Viele Manager verstehen einfach nicht, dass man zumindest bei komplexen Entwicklungen auch bei fertig gestelltem Code typischerweise noch weit von der Auslieferung in die Produktion entfernt ist. Während die Liste der für einen Release "fast fertigen" Funktionen immer länger und unhandlicher wird, fragen Manager oder Product Owner vom Team bereits neue Funktionen an.

Überfordern Sie nicht ihr schwächstes Kettenglied

Nehmen wir an, Ihr schwächstes Glied in der Kette sind die Abnahmetests. Zum Beispiel deshalb, weil Sie zu wenig Tester haben oder die Phase aufgrund der miserablen Code-Qualität einfach zu lange dauert.
Angenommen, das Abnahmetest-Team kann pro Woche nicht mehr als drei Features testen (dies ist nur eine für dieses Beispiel erdachte Metrik). Ihre Entwickler hingegen liefern sechs neue Features pro Woche. Für Manager, Product Owner und selbst das Team ist es natürlich verlockend, sechs neue Features pro Woche einzuplanen.

Tun sie es nicht! Auf die eine oder andere Art und Weise fahren Sie damit ganz schrecklich an die Wand.

Planen Sie lieber drei neue Features pro Woche und kümmern Sie sich in der verbleibenden Zeit um die Überwindung des Engpasses bei den Tests.

Zum Beispiel so:

- Lassen Sie ein paar Entwickler als Tester arbeiten (sie werden Sie dafür lieben....).
- Programmieren Sie Werkzeuge und Skripte zur Vereinfachung der Tests.
- Programmieren Sie mehr automatisierte Tests.
- Verlängern Sie ihre Sprints und machen Sie die Abnahmetest zu einem Teil des Sprints.
- Machen Sie einige Sprints zu reinen Test-Sprints, in denen alle nur an Abnahmetests arbeiten.

§ Stellen Sie weitere Tester ein (selbst wenn Sie dafür Entwickler loswerden müssen).

Bis auf die letzte haben wir alle Varianten ausprobiert. Die langfristig beste Lösung bilden die Punkte 2 und 3, bessere Werkzeuge und die Automatisierung von Tests.

Retrospektiven sind ein gutes Forum, um ihr schwächstes Kettenglied zu identifizieren.

Wieder zurück zur Realität

Ich habe wahrscheinlich den Eindruck vermittelt, dass wir in allen Scrum-Teams Tester sitzen haben, dass für jedes Produkt ein riesiges Team für Abnahmetests existiert, dass wir nach jedem Sprint ein Release Richtung Endkunden machen, und und und.

Aber so ist es nicht.

Manchmal ist es uns gelungen, all diese Dinge zu beherzigen und wir haben die Vorteile gespürt. Von einem akzeptablen Prozess der Qualitätssicherung sind wir generell aber noch weit entfernt. Es gibt noch viel zu lernen.

15

Wie wir mit mehreren Scrum-Teams umgehen

Vieles wird komplizierter, wenn Sie mehrere Scrum-Teams haben, die am gleichen Produkt arbeiten. Das Problem "je mehr Entwickler, desto mehr Komplikationen" gilt generell und ist keine Besonderheit von Scrum.

Wie so oft haben wir auch hier herumexperimentiert. Unser größtes Team bestand aus 40 Mitarbeitern, die alle am gleichen Produkt arbeiteten.

Es ergeben sich die Kernfragen:

- Wie viele Teams soll man schaffen?
- Wie verteilt man die Mitarbeiter auf diese Teams?

Wie viele Teams soll man aufstellen?

Wenn die Verwaltung mehrere Scrum-Teams so schwierig ist, warum befassen wir uns dann überhaupt damit und stecken nicht einfach alle in ein Team?
Das größte Scrum-Team, das wir einmal hatten, bestand aus elf Leuten. Es funktionierte, aber nicht besonders gut. Das tägliche Scrum-Meeting (Daily Scrum) überschritt oft die 15 Minuten-Grenze. Es entstand Verwirrung, weil Teammitglieder nicht wussten, woran andere gerade arbeiteten. Der ScrumMaster hatte seine Schwierigkeiten damit, alle auf ein gemeinsames Ziel einzuschwören und sich um jedes an ihn herangetragenes Problem zu kümmern.

Eine Alternative ist, sich in zwei Teams aufzuteilen. Aber ist das besser?

Nicht zwangsläufig.

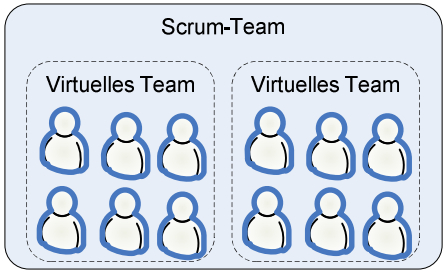
Es ist in jedem Fall eine gute Idee, wenn ihr Team eingespielt und Scrum-erfahren ist, Sie eine sinnvolle Möglichkeit finden, das Gesamtprojekt in zwei Teilprojekte zu zerlegen und sich der Quellcode der Teilprojekte auch noch gut separieren lässt. Gilt das nicht, belassen Sie es trotz der Nachteile lieber bei nur einem großen Team. Meiner Erfahrung nach sind

wenige, aber dafür etwas zu große Teams besser als viele kleine Teams, die sich untereinander behindern. Zerlegen Sie Teams also nur dann, wenn diese sich untereinander kaum behindern.

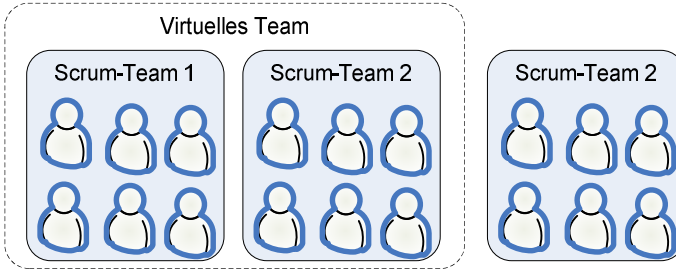
Virtuelle Teams

Woran merken Sie aber, ob Sie bezüglich Teamgröße und -anzahl richtig entschieden haben?
Wenn Sie Augen und Ohren offen halten, merken Sie, ob sich virtuelle Teams bilden.

Beispiel 1: Sie haben sich für ein großes Team entschieden, aber merken anhand der Kommunikationskanäle während des Sprints, dass sich praktisch zwei Unterteams gebildet haben.



Beispiel 2: Sie haben sich für drei kleine Teams entscheiden, merken aber, dass Team 1 und 2 ständig mit einander kommunizieren, während Team 3 isoliert arbeitet.



Was heißt das nun? Etwa, dass Ihre Art der Teamaufteilung falsch war? Die Antwort ist „Ja“, wenn die virtuellen Teams sich als dauerhaft erweisen. Sie lautet „Nein“, wenn die virtuellen Teams nur eine temporäre Erscheinung sind.
Lassen Sie uns noch einmal Beispiel 1 betrachten. Es war vermutlich eine richtige Entscheidung, sie als einzelnes Team zu belassen, wenn sich von Zeit zu Zeit die Zusammensetzung der beiden virtuellen Unterteams

ändert, also Personen von einer Gruppe zu anderen wandern. Gibt es keine solche Wanderung, sollten im nächsten Sprint über eine Aufteilung nachdenken.

Nun betrachten wir noch einmal Beispiel 2. Wenn Team 1 und Team 2 miteinander kommunizieren, aber Team 3 während des Sprints außen vor lassen, ist es im nächsten Sprint vielleicht sinnvoll, Team 1 und Team 2 zusammenzufassen. Legen Sie alle 3 Teams zusammen, wenn Team 1 und Team 2 in der erste Hälfte des Sprints interagieren und in der zweiten Hälfte Team 1 und Team 3. Oder lassen Sie einfach alles so wie es ist. Lassen Sie doch in der Sprint Retrospektive die Teams selbst entscheiden.

Das Thema Teamaufteilung ist bei Scrum eine schwierige Nuss. Zerbrechen Sie sich nicht den Kopf oder optimieren sich zu Tode. Experimentieren Sie, halten Sie Ausschau nach virtuellen Teams und nehmen Sie sich in den Retrospektiven Zeit dieses Thema zu behandeln. Früher oder später finden Sie die für Ihre Situation beste Lösung. Das Wichtigste ist, dass sich die Teams wohl fühlen und nicht gegenseitig auf die Füße treten.

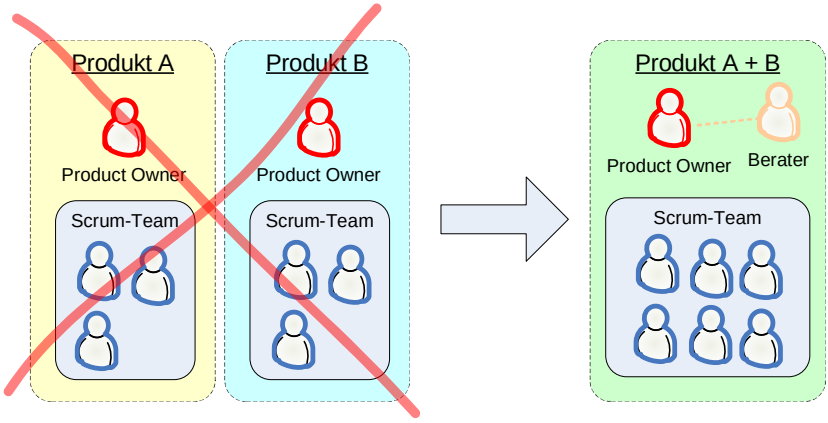
Optimale Teamgröße

Laut der meisten Bücher, die ich gelesen habe, liegt die optimale Teamgröße zwischen 5 und 9 Mitgliedern.

Meine eigenen Erfahrungen bestätigen das, auch wenn ich 3 bis 8 Leute für noch besser halte. Man sollte wirklich zu einigem bereit sein, um Teams dieser Größe zu erhalten.

Besteht ihr Scrum-Team aus zehn Leuten, dann denken Sie darüber nach, die schwächsten Personen herauszunehmen. Habe ich das wirklich gesagt?! Uups!

Nehmen wir einmal an, Sie haben zwei verschiedene Produkte und pro Produkt ein Dreier-Team, von denen beide jeweils zu langsam vorankommen. Es wäre vielleicht eine gute Idee, sie zu einem Sechser-Team zusammenzufassen, das sich um beide Produkte kümmert. In diesem Fall können Sie auf einen der Product Owner verzichten oder ihm eine Beratungsrolle geben.

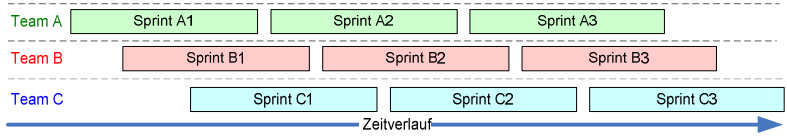


Nun nehmen wir an, sie haben ein einzelnes Scrum-Team mit 12 Mitgliedern und der Code ist zu schlecht, um zwei Teams getrennt voneinander daran arbeiten zu lassen. Um irgendwann das Team sinnvoll aufteilen zu können, stecken Sie lieber etwas Mühe in die Verbesserung des Codes, als an neuen Features zu arbeiten. Diese Investition zahlt sich sicherlich bald aus.

Parallel laufende Sprints - Ja oder Nein?

Angenommen, drei Scrum- Teams arbeiten am selben Produkt. Sollten ihre Sprints parallel ablaufen, also gemeinsam starten und enden? Oder sollten sie sich überlappen?

Am Anfang haben wir mit zeitlich überlappenden Sprints gearbeitet.

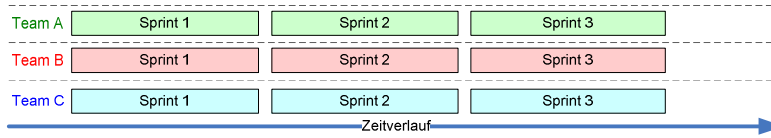


Uns hat gefallen, dass es so immer jeweils einen Sprint gibt, der bald zu Ende geht und einen der bald startet. Außerdem wäre das Arbeitspensum des Product Owners gleichmäßig verteilt, es gäbe regelmäßig irgendwelche Releases, Vorführungen... Ach wie wunderbar!

Das klang gut. Zu jedem Zeitpunkt würde es einen laufenden Sprint geben, der kurz vor dem Abschluss steht und einen neuen, der bald beginnen würde. Die Arbeitslast des Product Owners würde sich gleichmäßig über die Zeit verteilen. Das System würde kontinuierlich Releases ausspucken. Vorführungen in jeder Woche. Hallelujah.

Sie lachen...aber damals *klang* das alles wirklich überzeugend!

Gerade als wir anfingen so zu arbeiten, konnte ich ihm Rahmen meiner Scrum-Zertifizierung mit Ken Schwaber reden. Unsere Methode hielt er für eine *schlechte* Idee und empfahl stattdessen, die Sprints zu synchronisieren. Ich erinnere mich nicht mehr an seine genaue Begründung, war aber nach einiger Diskussion überzeugt.

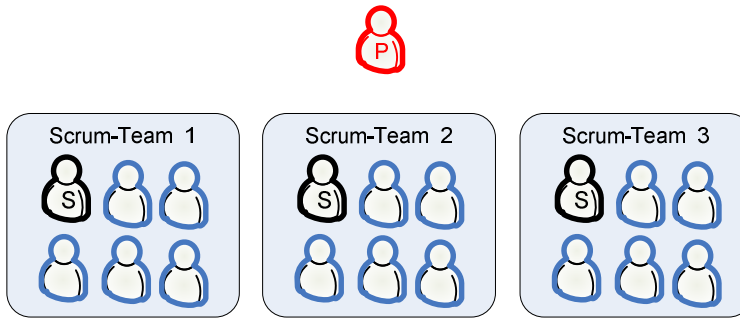


Diese Lösung haben wir seitdem immer verwendet und es nie bereut. Ich weiß nicht, ob uns die Variante mit sich überlappenden Sprints geschadet hätte; ich vermute aber ja. Der Vorteil von synchronen Sprints ist:

- Es ergibt sich ein guter Zeitpunkt Teams neu aufzustellen - zwischen den Sprints. Mit überlappenden Sprints wäre das nicht möglich, ohne zumindest ein Team im Sprint zu stören.
- Alle Teams können auf ein gemeinsames Sprint-Ziel hinarbeiten und die Sprint-Planungsmeetings gemeinsam abhalten. Das führt zu einer besseren Zusammenarbeit zwischen Teams.
- Der administrative Aufwand reduziert sich, z.B. weil weniger Planungsmeetings, Vorführungen und Releases nötig sind.

Warum wir eine Teamleiter-Rolle eingeführt haben

Gehen wir wieder von einem einzelnen Produkt mit drei Teams aus.



Die rote Person (P) ist der Product Owner. Die schwarzen Personen (S) sind ScrumMaster. Die restlichen sind ~~gewöhnliche~~ - ich meine respektable - Teammitglieder.

Wer entscheidet in dieser Konstellation, welche Mitarbeiter in welchen Teams sein sollten? Der Product Owner? Die ScrumMaster gemeinsam? Oder darf sich jeder sein Wunschteam aussuchen?

Was wäre, wenn sich dann jeder Team 1 auswählt; z.B. weil ScrumMaster 1 so *gut aussieht*?

Oder was passiert, wenn sich später herausstellt, dass es wirklich nicht möglich ist, dass zwei Teams parallel am Programmcode arbeiten und wir also doch mit zwei Neuner-Teams anstelle von drei Sechser-Teams arbeiten müssen? Dann wiederum benötigen wir nur zwei ScrumMaster. Welcher der drei ScrumMaster soll dann aber seines Amtes enthoben werden?

Nur in wenigen Firmen sind das nicht gerade empfindliche Themen.

Es erscheint verlockend, dem Product Owner die Zusammenstellung und Umordnung der Teams zu überantworten. Zählt das aber eigentlich zu seinen Aufgaben!? Er ist es, der dank seiner Fachexpertise dem Team die Entwicklungsrichtung aufzeigt, nicht aber sich um derartige Details kümmern sollte. Insbesondere, weil er im Sinne der Scrum Metapher von Hühnern und Schweinen (engl. chicken and pigs) ein "Huhn" ist. Wenn Sie diese Geschichte nicht kennen, dann googlen Sie nach "chickens and pigs".

Wir haben das Problem durch die Einführung einer Teamleiter-Rolle gelöst. Er ist so etwas wie der oberste ScrumMaster, der Chef, "Master aller ScrumMaster" oder wie Sie diese Rolle sonst nennen wollen. Er leitet kein einzelnes Team, sondern ist für teamübergreifende Aufgaben zuständig und entscheidet beispielsweise, wer für welches Team ScrumMaster wird, wie die Mitarbeiter in Teams aufgeteilt werden, usw.

Wir haben es uns nicht leicht gemacht, einen Namen für diese Rolle zu finden und "Teamleiter" war noch der brauchbarste, den wir finden konnten.

Aber egal wie sie die Rolle nennen. Die Idee eines Teamleiters hat bei uns gut funktioniert und ich kann sie nur empfehlen.

Wie wir Mitarbeiter in Teams aufteilen

Es gibt zwei allgemeine Arten Mitarbeiter, die am selben Endprodukt arbeiten sollen, in Teams aufzuteilen.

- Die Zuordnung wird durch einen ausgewählten Mitarbeiter vorgenommen, z.B. den oben erwähnten Teamleiter, den Product Owner, oder auch den Abteilungsleiter, insofern dieser für eine gute Wahl ausreichend involviert ist.
- Die Aufteilung wird durch die Mitarbeiter selbst vorgenommen

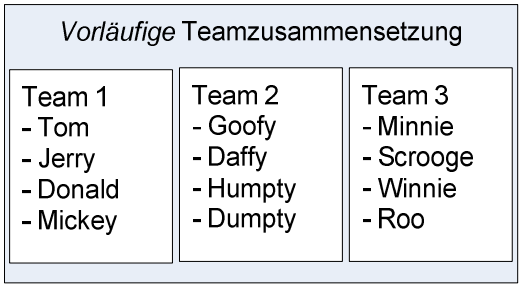
Wir haben mit allen drei Varianten herumexperimentiert. Drei? Ja: Strategie 1, Strategie 2 und einer Kombination der beiden.

Die Kombination funktioniert bei uns am besten. Dazu beruft der Teamleiter vor der Planung des Sprints ein eigenes Meeting zur Aufteilung des Teams mit dem Product Owner und allen ScrumMastern ein.

Wir sprechen über den vergangenen Sprint und entscheiden, ob Umordnungen der Teams nötig sind. Vielleicht überlegen wir dann, Teams zusammenzulegen oder Leute zwischen den Teams zu verschieben. Wir dokumentieren unsere Entscheidung über die *empfohlene* Teamzusammensetzung, um sie später im Sprint-Planungsmeeting einzubringen.

In diesem gehen wir zuerst die oberen Einträge des Product Backlogs durch. Dabei könnte der Teamleiter so in etwa sagen:

"Hallo zusammen. Für den kommenden Sprint schlagen wir die folgende Teamzusammensetzung vor."



"Wie ihr seht, sieht unser Vorschlag eine Reduzierung von vier auf drei Teams vor. Hier ist eine Liste mit den Mitgliedern der verschiedenen Teams. Bitte kommt hier vor die Wand und gruppiert euch nach dieser Liste."

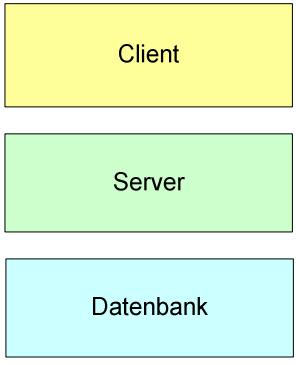
Der Teamleiter wartet ab, während alle durcheinander rennen und sich nach einer Weile vor der Wand drei Gruppen gebildet haben.

"Diese Gruppenaufteilung ist vorläufig! Wir nehmen sie als Ausgangspunkt, um Zeit zu sparen. Während dieses Sprint-Planungsmeetings könnt ihr nach Belieben in andere Teams wechseln, Teams aufteilen oder mit einem anderen zusammenschließen. Lasst einfach gesunden Menschenverstand und die Prioritäten des Product Owners entscheiden.

Wenn Sie so vorgehen, haben sie zu Beginn einerseits eine gewisse Kontrolle und später trotzdem genug Freiraum für Nachbesserungen - das funktioniert unserer Meinung nach am besten.

Spezialisierte Teams - oder nicht?

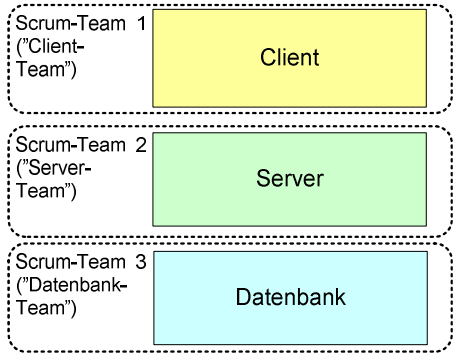
Angenommen, an unserem Produkt arbeiten 15 Mitarbeiter und die eingesetzte Technologie besteht aus drei Kernkomponenten:



Welche Teamaufstellung wählen wir, wenn wir es nicht bei einem einzelnen Team nicht belassen wollen?

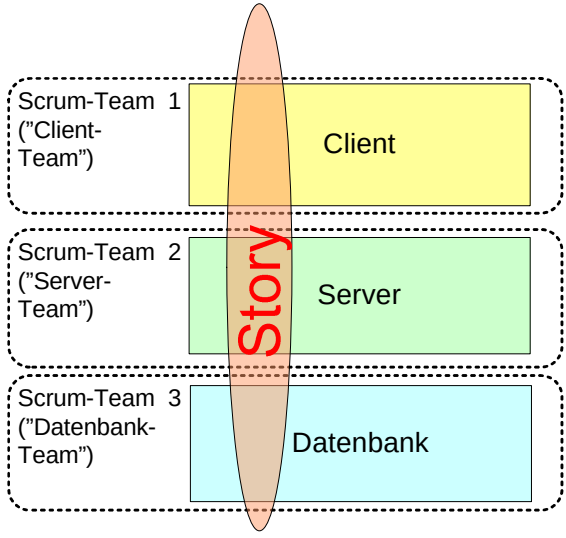
Ansatz 1: Komponententeams

Ein Weg wäre es, Spezialistenteams für die Komponenten zu bilden, also ein „Client“-Team, ein „Server“-Team und ein „Datenbank“-Team.



Mit dieser Aufteilung haben wir angefangen. Sie funktioniert aber nicht besonders gut, wenn der Großteil der User Stories alle Komponenten umfasst.

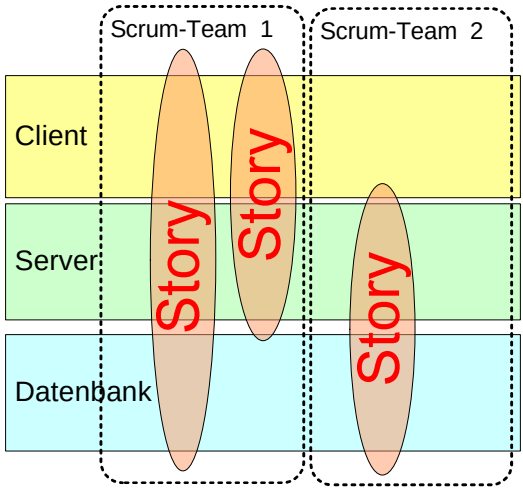
Nehmen wir als Beispiel die Story "Schwarzes Brett, auf dem Benutzer Nachrichten austauschen können". Dieses Feature macht eine Erweiterung der Benutzeroberfläche, zusätzliche Businesslogik auf dem Server und die Erweiterung der Datenbank um zusätzliche Tabellen nötig.



Das wiederum bedeutet, dass alle drei Teams, das Client-Team, das Server-Team und das Datenbanken-Team zur Fertigstellung dieser Story zusammenarbeiten müssen. Das ist nicht gerade ideal.

Ansatz 2: Komponentenübergreifende Teams

Eine andere Möglichkeit bieten komponentenübergreifende Teams, die nicht auf eine spezielle Komponente festgelegt sind.



Dieser Ansatz zur Teamaufteilung ist dann besser, wenn viele Ihrer Stories mehrere Komponenten betreffen. Das ist so, weil jedes Team Stories komplett umsetzen kann; inklusive der Arbeiten an Client, Server und Datenbank. Ein zusätzlicher Pluspunkt ist, dass Teams so unabhängig voneinander arbeiten können.

Eine der ersten Änderungen, die wir bei der Einführung von Scrum durchgeführt haben, war die Auflösung der Komponententeams zugunsten von komponentenübergreifenden Teams. Das führte dazu, dass nur noch selten der Fall auftrat, bei dem Dinge nicht abgeschlossen werden könnten, weil wir z.B. "darauf warten, dass die Serverleute ihren Teil liefern".

Je nach Situation, kann es aber immer noch vorkommen, dass wir "Spezialistenteams auf Zeit" aufstellen.

Teams zwischen Sprints umordnen - oder nicht?

Wie stark sich ein Sprint vom folgenden unterscheidet, hängt stark von den jeweils wichtigsten Stories ab. Die optimale Teamzusammensetzung variiert demnach ebenso stark.

Tatsächlich haben wir fast immer Gründe gefunden, einen Sprint als *unüblich* zu bezeichnen und demnach nach einer Zeit den Begriff "normaler Sprint" ganz verworfen - einfach deshalb, weil es "*normale* Sprints" genauso wenig gibt wie "*normale* Familien" oder "*normale* Menschen".

In einen Sprint ist es sinnvoll, mit einem reinen Client-Team aus lauter Client-Experten zu arbeiten. Im nächsten Sprint sind wiederum zwei komponentenübergreifende Teams inklusive eines Vermittlers aus dem Client-Team eine gute Idee.

Einer der Schlüsselaspekte von Scrum ist der Teamzusammenhalt, denn die gemeinsame Arbeit über mehrere Sprints hinweg schweißt das Team zusammen. Sie lernen, im Fluss mit der Gruppe zu arbeiten und so enorme Produktivität erreichen. Es kann einige Sprints dauern, bis sie das erreichen. Aber wenn Sie ständig die Teams umbauen, erreichen Sie vielleicht auch niemals einen wirklich starken Teamzusammenhalt.

Machen Sie sich also die Konsequenzen bewusst, bevor Sie die Teams neu ordnen. Und fragen Sie sich, ob es sich um eine kurz- oder langfristige Veränderung handelt? Ist sie nur kurzfristig, verwerfen Sie die Änderung eventuell ganz. Ist sie hingegen langfristig, dann zögern Sie nicht. Das alles gilt nicht, wenn Sie für ein großes Team Scrum gerade erst einführen. Dann ist es wirklich sinnvoll, eine Zeit mit der Teamzusammensetzung herumzuexperimentieren, bis Sie zufrieden sind. Machen Sie klar, dass es ein paar Fehlschläge in Ordnung sind, solange Sie sich kontinuierlich verbessern.

Teilzeit-Teammitglieder

Ich kann die Meinung verschiedener Scrum-Bücher nur bestätigen, dass Teilzeit-Kräfte in Scrum-Teams keine gute Idee sind.

Angenommen, Sie wollen Joe als Teilzeit-Mitglied in Ihr Team aufnehmen. Überlegen Sie sich gut, ob Sie ihn wirklich brauchen? Sind Sie sicher, dass Sie ihn nicht Vollzeit bekommen können? Was sind seine anderen Verpflichtungen und können diese an Andere delegiert werden, so dass Joe nur noch als passiver Berater beteiligt ist? Kann Joe dem Team im nächsten Sprint Vollzeit unterstützen und in der Zwischenzeit einen anderen einlernen?

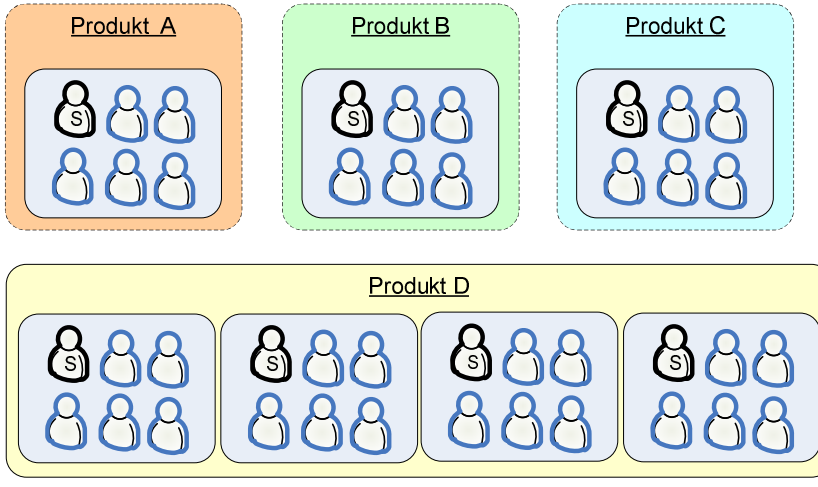
Manchmal gibt es aber keine Alternative. Sie brauchen Joes Hilfe dringend, weil er der einzige Datenbank-Administrator im Haus ist. Da die anderen Teams ihn aber mindestens genauso brauchen und Ihre Firma keine zusätzlichen Administratoren einstellen will, wird er Ihren Team nie Vollzeit zur Verfügung stehen (Genau so war es bei uns!). Gut - in diesem Fall ist ein Teilzeit-Engagement natürlich erlaubt. Beurteilen Sie solche Fälle aber jedes Mal aufs Neue.

Prinzipiell ist mir ein Team mit drei Vollzeitkräften lieber, als eines mit acht Teilzeitkräften.

Und auch wenn Sie Mitarbeiter haben, die wie der erwähnte Administrator, ihre Zeit auf mehrere Team verteilen müssen, ist es sinnvoll diese einem Team zuzuordnen. Finden Sie das Team was ihn mit mutmaßlich am häufigsten braucht und machen Sie es zu seinem Haupt-Team. Wenn er kann, besucht er die täglichen Scrum-Meetings, Sprint-Planungsmeetings und Retrospektiven dieses Teams.

Wie wir Scrum-of-Scrums- Meetings durchführen

Scrum-of-Scrums sind eigentlich reguläre Meetings, bei denen sich alle ScrumMaster zum Austausch zusammenfinden.
Einmal arbeiteten wir an vier Produkten, von denen nur drei ein einzelnes Scrum-Team hatten. Am vierten Produkt arbeiteten 25 Mitarbeiter, die auf mehrere Scrum-Teams verteilt waren. Das sah ungefähr so aus:



Wir hatten also Scrum-of-Scrums auf zwei Ebenen. Eines auf Produktebene, mit allen Teams für Produkt D und eines auf Unternehmensebene, für alle Produkte.

Scrum-of-Scrums auf Produktebene

Dieses Meeting war von großer Bedeutung. Wir hielten es einmal pro Woche ab; manchmal auch häufiger. Wir diskutierten Probleme bei der Integration, bezüglich der Teamauslastung, bereiteten die anstehenden Sprint-Planungsmeetings vor, usw. Die anfangs veranschlagten 30 Minuten für das Meeting reichten selten aus. Eine Alternative wären tägliche Scrum-of-Scrums Meetings gewesen; nur haben wir es nie geschafft, diese auszuprobieren.

Die Agenda unserer Scrum-of-Scrums lautete:
1) Der Reihe nach berichtet jeder, was sein Team in der vergangenen Woche erreicht hat, was sie für diese Woche einplanen und welche Probleme auftraten.
2) Behandlung team-übergreifender Themen wie z.B. Integrationsprobleme.

Aus meiner Sicht ist bei den Scrum-of-Scrums allerdings nicht die Agenda von entscheidender Bedeutung, sondern die regelmäßige Durchführung.

Scrum-of-Scrums auf Unternehmensebene

Dieses Meeting haben wir den "Pulsschlag" genannt und in unterschiedlichen Formen und Teilnehmerkreisen durchgeführt. Diese Konzept haben wir erst kürzlich verworfen und halten stattdessen eine wöchentliche, 15-minütige Vollversammlung ab, zu der alle eingeladen sind, die am Entwicklungsprozess beteiligt sind.

Wie bitte!? Eine 15-minütige Vollversammlung mit allen Mitgliedern aller Teams!? Das soll funktionieren!?

Ja, es funktioniert. Nämlich dann, wenn derjenige, der das Meeting leitet, darauf achtet, es kurz zu halten.

Der Ablauf ist folgender:
1) Neuigkeiten und Aktuelles vom Entwicklungsleiter, z.B. Infos zu bald anstehenden Ereignissen.
2) Die Mitglieder jeder Produktgruppe sagen der Reihe nach, was sie in der letzten Woche erreicht haben, was sie in dieser Woche erreichen

wollen und welche Probleme aufgetreten sind. Auch andere Teammitglieder, wie der Leiter des Kundendienstes und der Testabteilung, berichten. Der Rest der Mannschaft steuert Zusatzinformationen bei und stellt Fragen.

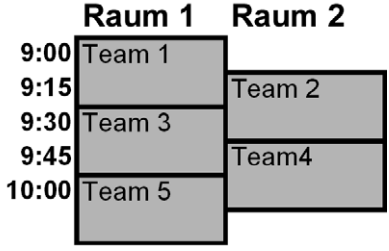
Es geht also um schnellen Informationsaustausch und nicht um Diskussion und Reflektion. Wenn wir das beherzigen, sind 15 Minuten meistens ausreichend. Manchmal dauert es auch länger, aber nur selten mehr als 30 Minuten. Kommt es doch zu interessanten Diskussionen, unterbreche ich diese und bitte die Interessierten, nach dem Meeting weiterzudiskutieren.

Warum aber machen wir die "Pulsschlag" Meetings überhaupt in so großer Runde? Zum einen, weil die Scrum-of-Scrums auf Unternehmensebene sich rein auf Statusberichte beschränken und es nur selten zu Diskussionen in der Gruppe kommt. Zum anderen, weil viele Mitarbeiter außerhalb der Gruppe ganz gierig auf diese Berichte sind. Da letztlich jedes Team wissen will, was die anderen machen und wir uns sowieso zusammensetzen, um einander auf dem Laufenden zu halten, können wir dazu auch gleich alle einladen.

Tägliche Scrum-Meetings aufeinander abstimmen

Wenn bei Ihnen mehrere Teams an einem Produkt arbeiten und alle ihr tägliches Scrum-Meeting zur selben Zeit abhalten, haben Sie ein Problem: Der Product Owner und andere interessierte Personen können nur an einem dieser Meetings teilnehmen.

Aus diesem Grund halten wir unsere Teams an, parallele Scrum-Meetings zu vermeiden.



Der oben stehende Beispiel-Zeitplan stammt noch aus einer Zeit, als unsere täglichen Scrum-Meetings in anderen Räumen als dem Teambüro

abgehalten wurden. Obwohl das Meeting nur 15 Minuten dauert, sieht er für Überziehungen pro Team 30 Minuten vor.

Das ist aus zwei Gründen *sehr praktisch.*

1. Mitarbeiter wie der Product Owner oder ich können alle täglichen Scrum-Meetings an einem einzigen Morgen besuchen. Es gibt keine bessere Art, ein akkurates Bild über den Zustand des Sprints und die wesentlichen Bedrohungen zu erhalten.
2. Teams können die Scrum-Meetings anderen Teams besuchen: Auch wenn das zwar eher selten passiert, kommt es vor, dass Teams, die an ähnlichen Themen arbeiten, sich gegenseitig auf dem Laufenden halten und die Scrum-Meetings der anderen besuchen.

Der Nachteil ist, dass die einzelnen Teams bei der Terminwahl für das tägliche Scrum-Meeting nicht mehr ganz so frei sind. Für uns ist das aber eigentlich nie ein Problem gewesen.

"Feuerwehr-Teams"

Wir hatten einmal die Situation, dass bei einer großen Neuentwicklung Scrum nicht zum Einsatz kam, weil wir ausschließlich mit Notfällen (z.B. ein zu früh ausgeliefertes System zu bereinigen) beschäftigt waren. Es war eine Teufelskreis, weil das viele "Feuerlöschen" uns gleichzeitig davon abhielt die Ursachen der Notfälle zu beseitigen und z.B. die Architektur zu verbessern, Tests zu automatisieren oder Überwachungs- und Warnsysteme einzubauen.

Um dem Problem zu begegnen, stellten wir ein eigenes Feuerwehr-Team und ein Scrum-Team auf.

Die Aufgabe des Scrum Teams war es, gemeinsam mit dem Product Owners zu versuchen, das System zu stabilisieren und damit Notfälle zu vermeiden. Das Feuerwehr- oder Support-Team, - wie wir es offiziell nannten - hatte zwei Aufgaben:
1) Feuerlöschen
2) Das Scrum-Team von Störungen, also z.B. Featureanfragen, die plötzlich wie aus dem Nichts auftreten, abzuschotten.

Um das Scrum-Team auch *physisch* vor Störungen wie z.B. übereifrigen Vertrieblern oder aufgebrachten Kunden, zu *schützen*, saß das Feuerlösch-

Team in der Nähe der Tür, das Scrum Team hingegen im hinteren Teil des Raums.

Damit die Teams nicht zu abhängig von der Kernkompetenz des jeweils anderen Teams wurden, haben wir beiden Teams Senior-Entwickler zugeteilt.

Das war letztlich auch ein Versuch, das Hindernis bei der Einführung von Scrum zu lösen, dass Teams üblicherweise kaum in der Lage sind, ihre Arbeit mehr als einen Tag im Voraus zu planen.

Wie erwähnt, war also unser Ansatz die Gruppe aufzuteilen; und das funktionierte auch recht gut.

Mit genügend Freiheiten ausgestattet, um vorausschauend zu arbeiten, gelang es dem Scrum-Team schließlich das System zu stabilisieren. Das Feuerwehr-Team hingegen verabschiedete sich vollends vom Anspruch im Voraus zu planen und reagierte nur auf die nächste brenzlige Notsituation.

Völlig *ungestört* arbeiten, konnte das Scrum Team allerdings nicht. Oft musste das Scrum-Team einzelne Schlüsselpersonen zu Rate ziehen; im schlimmsten Fall sogar das gesamte Team.

Trotz alledem war das System nach ein paar Monaten so stabil, dass wir das Feuerwehr-Team zugunsten neuer Scrum-Teams auflösen konnten. Und unsere Feuerwehrmänner waren recht froh darüber, ihre verbeulten Helme an den Nagel hängen und zu Scrum-Teams dazu stoßen zu können.

Das Product Backlog aufteilen - oder nicht?

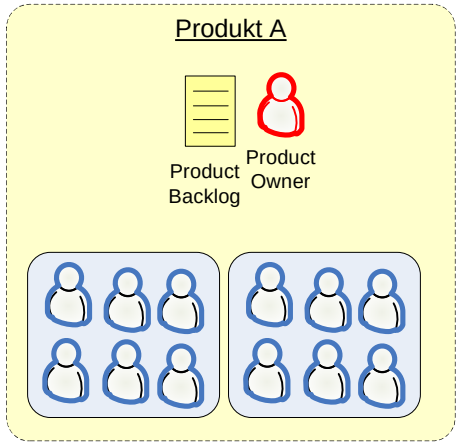
Angenommen Sie entwickeln ein Produkt und haben zwei Scrum-Teams. Wie viele Product Backlogs sollen Sie führen? Wie viele Product Owner? Wir haben drei Varianten ausprobiert, und je nachdem für welche man sich entscheidet, laufen die Sprint-Planungsmeetings anders ab.

Variante 1: Ein Product Owner und ein Backlog

"Es kann nur Einen geben" ist unsere bevorzugte Variante.

Der Vorteil dabei ist, dass sich die Teams anhand der Prioritätenliste des Product Owners weitestgehend selbst bilden. Der Product Owner kann

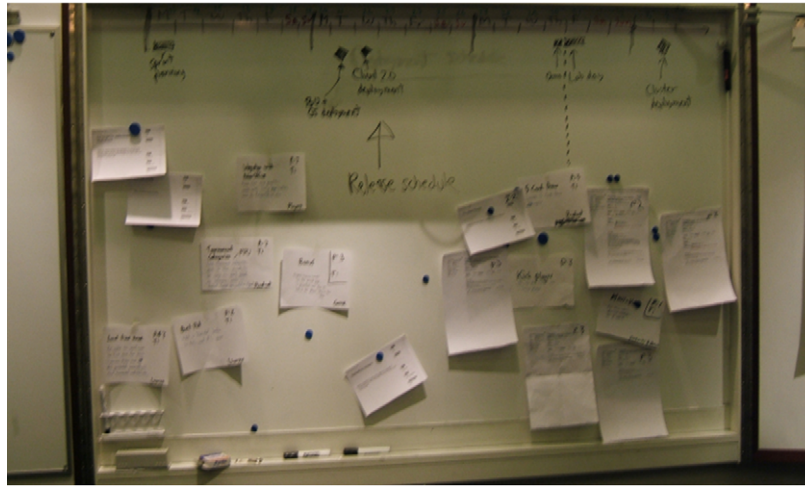
sich so auf seine Anforderungen konzentrieren, während die Teams über die Arbeitsteilung entscheiden.



In der Praxis läuft unser Sprint-Planungsmeeting dann ungefähr so ab:

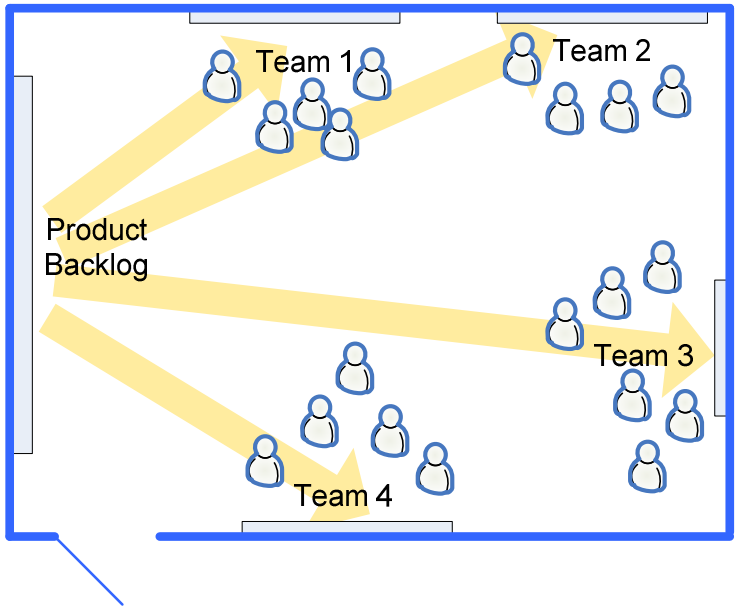
Das Meeting findet in einem Konferenzzentrum außerhalb der Firma statt.

Kurz vor Beginn macht der Product Owner eine Wand zur "Product Backlog Wand", indem er dort die Karteikarten mit den Stories aufhängt und nach Wichtigkeit sortiert. Er bringt so lange Stories an, bis die Wand voll ist. Das dürften dann weit mehr Features sein, also man in einem Sprint schaffen kann.



Jedes Scrum-Team wählt eine leere Wand und hängt dort ein Schild mit seinem Teamnamen auf. Das wird seine "Teamwand". Anschließend holen die Teams sich Stories (die wichtigsten zuerst) von der Backlog-Wand und hängen sie an Ihre Teamwand.

Die unten stehende Abbildung illustriert diesen Vorgang. Die gelben Pfeile zeigen den Fluss der Story-Karteikarten von der Wand mit dem Product Backlog zu den Teamwänden.



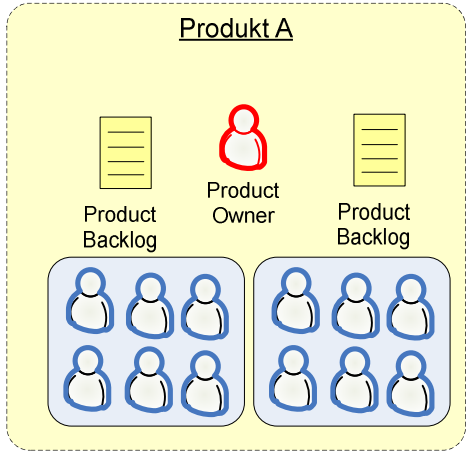
Im weiteren Verlauf des Meetings sprechen Product Owner und die Teams alle Karteikarten mit den Stories durch, verteilen diese auf die Teams, ändern Prioritäten und zerlegen sie unter Umständen weiter. Nach ungefähr einer Stunde hat jedes Team eine erste Version des Sprint-Backlogs auf seiner Teamwand. Anschließend arbeiten die Teams getrennt voneinander an Zeitschätzungen und der Zerlegung der Stories in Aufgaben.

Das wilde Durcheinander, das dabei entsteht, ist zwar anstrengend, aber zugleich auch äußerst effektiv und ein großer Spaß. Am Ende haben alle Teams normalerweise genug Informationen beisammen, um ihren Sprint zu beginnen.

Variante 2: Ein Product Owner, mehrere Backlogs

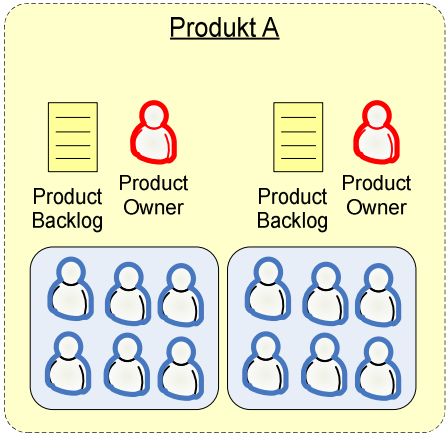
Hier verwaltet der Product Owner *mehrere* Product Backlogs; eines pro Team. Wir haben diesen Ansatz noch nicht ausprobiert, waren aber kurz davor es zu tun. Im Grunde ist dies unsere Ausweichvariante für den Fall, wenn der erste Ansatz nicht funktioniert.

Der Nachteil hierbei ist allerdings, dass der Product Owner den Teams die Stories zuweist, obwohl die Teams das besser selbst machen könnten.



Variante 3: Mehrere Product Owner mit jeweils eigenem Backlog

Das ist vergleichbar mit dem zweiten Ansatz, bei dem jedes Team ein eigenes Product Backlog hat. Zusätzlich hat hier jedes Team einen eigenen *Product Owner*!



So haben wir es noch nie gemacht und so werden wir es vermutlich auch nie machen.

Einfach weil mit einem ernsten Interessenkonflikt zwischen den Product Ownern zu rechen ist, wenn die verschiedenen Product Backlogs dieselbe Codebasis betreffen.

Ist dies nicht der Fall, verhält es sich so wie bei der Zerlegung des Endprodukts in unabhängig voneinander zu bearbeitende Teilsysteme. Wir hätten dann die angenehme und einfache Situation von je einem Team pro Produkt.

Code Branching

Arbeiten mehrere Teams an derselben Codebasis, ist es unvermeidlich im Versionierungssystem mit separaten Entwicklungssträngen (engl. code branches) zu arbeiten. Es gibt eine ganze Reihe Bücher und Artikel, die sich ausschließlich damit beschäftigen, wie mehrere Personen gemeinsame auf der gleichen Codebasis arbeiten können. Da ich dem keine revolutionären Innovationen hinzuzufügen habe, erspare ich mir Details zum Thema. Trotz allem hier ein paar Lektionen, die unser Team bisher gemacht hat.

- Halten Sie den Hauptstrang (engl. trunk) unbedingt in einem konsistenten Zustand. Alles sollte sich kompilieren lassen und die Unit Test sollten fehlerfrei abschließen. Zu jeder Zeit sollte sich ein lauffähiges Release erzeugen lassen. Idealerweise baut ein sog. Continuous Build System jede Nacht ein Release und installiert es auf der Testumgebung.
- Taggen Sie jedes Release! Für jede Auslieferung auf Ihre Abnahme- oder Produktivumgebung sollte auf dem Hauptstrang (engl. trunk) ein Versionstag gesetzt werden, damit sie genau wissen was Sie ausgeliefert haben. So haben sie später jederzeit die Möglichkeit einen entsprechenden Wartungsstrang (engl. release branch) anlegen zu können.
- Legen Sie nur dann separate Codestränge (engl. branches) an, wenn sich dies nicht vermeiden lässt. Eine gute Regel ist es, nur dann einen neuen Codezweig anzulegen, wenn sie andernfalls bestehende Regeln brachen müssten. Im Zweifelsfall aber verzichten Sie lieber auf separate Zweige (engl. branches); da jeder zusätzliche Zweig den Wartungsaufwand und die Komplexität erhöht.
- Nutzen Sie Codezweige vor allem dazu, verschiedene Entwicklungsstränge unterscheidbar zu machen. Es bleibt Ihnen überlassen, ob jedes Scrum-Team einen eigenen Branch erhält. Bedenken Sie aber, dass die Auslieferung von Fehlerbehebungen (engl. patches) schwierig werden kann, wenn langfristige Weiterentwicklungen und kurzfristige Bugfixes in ein und demselben Codestrang gemischt sind.
- Gleichen Sie häufig ab. Wenn Sie auf einem Codestrang arbeiten, sollten Sie Ihren Stand mit dem Hauptstrang abgleichen, sobald sie etwas Kompilierbares haben. Außerdem sollten jeden Morgen, wenn Sie ins Büro kommen, einen Abgleich machen. So können sie auf einem Stand weiterarbeiten, die der aktuellen Änderungen der anderen Teammitglieder enthält. Auch wenn das Zusammenführen der Änderungen manchmal die Hölle ist, sollten Sie bedenken, dass der Verzicht darauf wesentlich schlimmere Konsequenzen haben kann.

Retrospektiven mit mehreren Teams

Es stellt sich die Fragen, wie Sprint-Retrospektiven ablaufen, wenn mehrere Teams am selben Produkt arbeiten?

Sobald die Sprint-Vorführung und der anschließende Applaus und Smalltalk abgeschlossen sind, zieht sich jedes Team in einem separaten

Raum oder an einen gemütlichen Ort außerhalb des Büros zurück. Dort machen sie Ihre Retrospektive genau so, wie ich es im Abschnitt "Wie wir Sprint-Retrospektiven durchführen" beschrieben habe.

In den Sprint-Planungsmeetings, an denen dann ja - wegen der synchronisierten Sprints - wieder alle Teams teilnehmen, fasst der Sprecher jedes Teams kurz die wichtigsten Ergebnisse der Retrospektive zusammen. Pro Team dauert das etwa fünf Minuten. Anschließend wird zehn bis 20 Minuten diskutiert. Es folgt eine kurze Pause, bevor wir mit der eigentlichen Sprintplanung starten. (Siehe dazu auch den Abschnitt "Leerlauf zwischen Sprints")

Auf eine Zusammenfassung der Ergebnisse der Retrospektive verzichten wir, wenn nur ein Team am Produkt arbeitet. In diesem Fall waren ja alle Kollegen bei der eigentlichen Retrospektive dabei.

16

Wie wir mit verteilten Teams umgehen

Was passiert, wenn das Team auf mehrere Orte verteilt ist? Liegt die Magie von Scrum und XP nicht gerade darin, dass alle zusammen sitzen, jeden Tag in Paaren programmieren und ständig persönlichen Kontakt haben?

Auch wir haben Teams, die nicht vor Ort sitzen, und Mitarbeiter, die gelegentlich von zu Hause arbeiten. Unser Umgang damit ist recht simpel. Wir lassen nichts unversucht, die Bandbreite für die Kommunikation mit entfernten Teammitgliedern zu maximieren. Zur Bandbreite zählen für mich nicht nur technische Faktoren wie MBit/s, sondern auch, ob es möglich ist:

- gemeinsam in Paaren zu programmieren,
- sich im Rahmen der täglichen Scrum-Meetings persönlich zu treffen,
- jederzeit miteinander diskutieren zu können,
- sich persönlich treffen und reden zu können,
- spontan Meetings mit dem gesamten Team einberufen zu können,
- sich jederzeit die Infowände mit dem Sprint Backlog, Sprint Burndown-Diagramm und Product Backlog ansehen zu können.

Hier einige Maßnahmen, die wir bereits einsetzen oder planen demnächst einzusetzen:

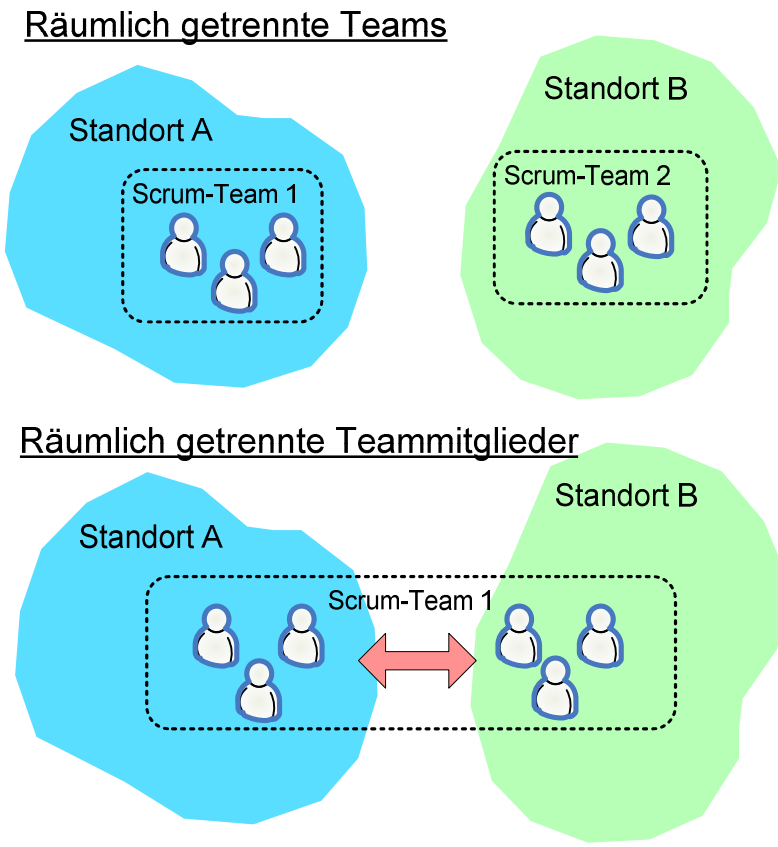
- Jeder Arbeitsplatz ist mit Webcam und Headset ausgestattet.
- Konferenzräume sind über Webcams, Konferenzmikrofone und ständig laufende Computer mit Desktop Sharing Software ständig mit der Außenwelt verbunden.
- Große Monitore an jedem Standort dienen als virtuelle Fenster zwischen Standorten und erlauben permanent einen Blick auf die anderen Standorte. Man kann davor stehen, winken und beobachten, wer gerade an seinem Schreibtisch sitzt oder sich mit anderen unterhält.
- Ein Austauschprogramm, bei dem sich die Mitarbeiter verschiedener Standorte regelmäßig gegenseitig besuchen.

Mit diesen Mitteln haben wir langsam, aber sicher, den Kniff heraus, wie man Sprint-Planungsmeetings, Vorführungen, Retrospektiven und tägliche Scrum-Meetings mit einem verteilten Team macht.

Wie so oft, geht es auch hier ums kontinuierliche Experimentieren. Inspizieren => Adaptieren => Inspizieren => Adaptieren => Inspizieren => Adaptieren => Inspizieren => Adaptieren => Inspizieren => Adaptieren

Offshoring

Wir arbeiten mit diversen Offshore-Teams und haben viel versucht, das erfolgreich mit Scrum unter "einen Hut zu bringen".
Hier gibt es zwei Herangehensweisen; räumlich getrennte Teams oder räumlich getrennte Teammitglieder.



Auch wenn die erste Strategie mit verteilten Teams verlockend ist, haben wir mit der zweiten Strategie begonnen. Unsere Gründe waren:

1. Die Teammitglieder sollen sich untereinander gut kennen lernen.
2. Wir wollen eine exzellente Kommunikationsinfrastruktur zwischen den beiden Standorten und das Team sollte Anreiz genug haben, diese zu etablieren.
3. Das ausgelagerte Team ist anfänglich zu klein, um als effektives, eigenständiges Scrum-Team zu fungieren.
4. Unabhängige Offshore-Teams sind erst dann eine praktikable Alternative, wenn es zuvor eine Phase intensiven Wissensaustausches gegeben hat.

Langfristig kann es sich bei uns aber durchaus auch in Richtung separater Teams bewegen.

Heimarbeit von Teammitgliedern

Von Zuhause aus zu arbeiten, kann manchmal richtig praktisch sein. Manchmal erledigt man daheim an einem Tag mehr Programmierarbeit, als in einer ganzen Woche im Büro. Zumindest, wenn man keine Kinder hat :o)

Nun ist es aber doch ein Grundpfeiler von Scrum, das gesamte Team vor Ort zu haben. Was also tun?

Wir überlassen die Entscheidung, wann und wie oft jemand von daheim arbeitet, im Wesentlichen den Teams selbst. Einige Kollegen arbeiten regelmäßig von daheim, weil sie eine lange Anfahrt haben. Unabhängig davon bestärken wir unsere Mitarbeiter trotzdem, überwiegend am selben Ort wie der Rest des Teams zu sein.

Wenn Kollegen von daheim aus arbeiten, nehmen sie per Skype Sprach- oder Videoanruf am täglichen Scrum-Meeting teil. Außerdem sind sie ständig per Instant Messenger erreichbar. Zwar ist das nicht so gut, wie im gleichen Raum zu sitzen, aber es ist immer noch gut genug.

Vor einiger Zeit haben wir versucht einen sog. "Ruhetag" einzuführen. Das war ein fester Wochentag, an dem jeder bei Bedarf und Rücksprache mit den Kollegen von Zuhause arbeiten könnte. Im Team bei dem wir das ausprobiert haben, funktionierte die Regelung gut. Die meisten Kollegen blieben mittwochs daheim, arbeiteten viel ab und funktionierten trotz allem immer noch gut als Team. Weil man immer nur einen Tag getrennt

war, kam es nie zu größerem Auseinanderdriften. Aus unerfindlichen Gründen hat sich diese Regelung nie bei den anderen Teams durchsetzen können.

Im Großen und Ganzen, war es bei uns auch nie ein Problem, dass Leute von Zuhause aus arbeiten.

17

Checkliste für ScrumMaster

Im letzten Kapitel möchte ich Ihnen unsere Checkliste für ScrumMaster zeigen. Sie enthält die Routineaufgaben, die der ScrumMaster am häufigsten tut. Sie enthält Dinge, die man leicht vergisst; Selbstverständlichkeiten, wie "Hindernisse beseitigen", haben wir hingegen weggelassen.

Zu Beginn des Sprints

- Erstelle nach dem Sprint-Planungsmeeting eine Info-Webseite zum Sprint.
  - Verlinke die Startseite des Wikis mit dieser Seite.
  - Drucke die Seite aus und hänge sie so auf, dass andere sie sehen können, wenn sie vorbeilaufen.
- Informiere alle mit einer Email, dass ein neuer Sprint begonnen hat. Die Email sollte auch das Sprint-Ziel nennen und einen Link zur Sprint-Infoseite enthalten.
- Aktualisiere das Sprint-Statistikdokument. Trage die geschätzte Entwicklungsgeschwindigkeit (engl. Velocity), die Teamgröße, Sprintlänge und ähnliches ein.

Täglich

- Sorge dafür, dass das tägliche Scrum-Meeting pünktlich beginnt und endet.
- Sorge dafür, dass Stories zum Sprint Backlog hinzugefügt bzw. entfernt werden, um den Zeitplan des Sprints zu halten.
  - Sorge dafür, dass der Product Owner über solche Änderungen informiert wird.

- Sorge dafür, dass das Team Sprint Backlog und Burndown-Diagramm aktualisiert.
- Sorge dafür, dass Probleme und Hindernisse entweder behoben oder an den Product Owner oder Entwicklungsleiter delegiert werden.

Am Sprint-Ende

- Veranstalte eine öffentliche Sprint-Vorführung.
- Verschicke 1-2 Tage vorher die zugehörige Einladung an alle Teilnehmer.
- Veranstalte mit dem gesamten Team und dem Product Owner eine Sprint-Retrospektive. Lade auch den Entwicklungsleiter mit ein, damit er die gesammelten Erfahrungen weitergeben kann.
- Aktualisiere das Dokument mit der Sprint-Statistik, und trage die tatsächliche gemessene Entwicklungsgeschwindigkeit und die wichtigsten Ergebnisse der Retrospektive ein.

18

Schlusswort

Puhh! Ich habe niemals gedacht, dass das Ganze so umfangreich werden würde.

Ich hoffe, dass - egal ob Sie beim Thema Scrum Neuling oder alter Hase sind - dieses Dokument Ihnen hilfreiche Einsichten beschert hat.

Da Scrum an den jeweiligen Kontext angepasst werden sollte, ist es natürlich schwierig, allgemein gültige „Beste Praktiken“ zu geben. Nichtsdestotrotz freue ich mich auf Ihre Anregungen, Verbesserungsvorschläge und Rückmeldungen dazu, wie Ihr Scrum-Alltag von unserem abweicht.

Schreiben Sie mir unter **henrik.kniberg@crisp.se.** Auch auf der Mailingliste **scrumdevelopment@yahoogroups.com** bin ich hin und wieder aktiv.

Wenn Ihnen das Buch gefallen, empfehle ich Ihnen auch gelegentlich auf meinem Weblog vorbeizuschauen. Ich hoffe ich komme bald dazu, neue Einträge zum Thema Java und Agiler Softwareentwicklung einzustellen.
http://blog.crisp.se/henrikniberg/

Oh - und bitte vergessen Sie nicht....

Es ist bloß ein Job, oder?

Lese-Empfehlungen

Den folgenden, absolut empfehlenswerten Büchern verdanke ich viele Ideen und Geistesblitze!

Der Autor

Henrik Kniberg (henrik.kniberg@crisp.se) arbeitet als Berater für Crisp in Stockholm (www.crisp.se). Seine Spezialgebiete sind Java und agile Softwareentwicklung. Seit der Erscheinung der ersten XP Bücher und des Agilen Manifests hat Henrik sich agile Prinzipien verschrieben, und versucht diese erfolgreich in verschiedenen Unternehmensformen zum Einsatz zu bringen. Er ist Mitbegründer der Firma Goyoda, wo er zwischen 1998 und 2003 als technischer Geschäftsführer arbeitete. Bei dem Aufbau und der Leitung einer technischen Infrastruktur und eines 30-köpfigen Teams hatte er dort ausreichend Gelegenheit, testgetriebene Entwicklung und andere agile Praktiken auszuprobieren.

Ende 2005 wurde Henrik als Entwicklungsleiter bei einem schwedischen Computerspielhersteller eingestellt. Dieser steckte wegen akuter organisatorischer und technischer Probleme in Schwierigkeiten. In Form von Scrum und XP etablierte er auf allen Unternehmensebenen die Prinzipien von Lean und der Agilen Softwareentwicklung und half der Firma so aus der Krise.

Als Henrik an einem Freitag im November 2006 mit Fieber zuhause im Bet lagt, beschloss er niederzuschreiben, was er im vergangenen Jahr gelernt hatte. Als er mit dem Schreiben erst einmal angefangen hatte, konnte er gar nicht mehr damit aufhören. So entstand nach drei Tagen fieberhaften Schreibens und Zeichnens aus den anfänglichen Notizen ein 80-seitiger Artikel mit dem Titel "Scrum and XP from the Trenches". Daraus ist schließlich auch dieses Buch geworden.

Henrik ist Generalist und liebt es, verschiedene Rollen wie Manager, Entwickler, ScrumMaster, Lehrer und Coach einzunehmen. Egal in welcher Rolle, ist es seine große Leidenschaft, Unternehmen dabei zu helfen, großartige Teams aufzustellen und großartige Software zu entwickeln.

Henrik ist in Tokio aufgewachsen und lebt heute mit seiner Frau Sophia und seinen zwei Kindern in Stockholm. In seiner Freizeit macht er Musik, als Komponist, sowie als Keyboard-Spieler und Bassist in lokalen Bands.

Wenn Sie mehr über Henrik wissen wollen, besuchen Sie die Website **http://www.crisp.se/henrik.kniberg**

Die Übersetzer der deutschen Ausgabe

Robert Sösemann (robert@soesemann.com) arbeitet als IT-Architekt für Softwarelösungen im Bereich Internet, Java, SOA, BPM bei der Logicline GmbH in Stuttgart.

Davor war er IT-Consultant bei einer amerikanischen Unternehmensberatung, Technischer Projektleiter in einer großen Online-Werbeagentur und Entwicklungsleiter bei einem Softwarehersteller für Marketingsoftware.
Fast immer ging es darum, zu hohe Erwartungen, in zu kurzer Zeit mit zu unausgereiften Technologien zu erfüllen. Die Erfahrung, dass IT-Projekte viel öfter an Rahmenbedingungen als an der Technik scheitern, steigerte stetig sein Interesse und Engagement für die andere - eben agile - Art Software zu entwickeln.

Wie kein anderes Buch hat „Scrum and XP from the Trenches“ Robert dazu angespornt, in den eigenen Projekten Agile Praktiken anzuwenden, mit ihnen zu experimentieren und die zugrunde liegenden Werte bei Kunden, Kollegen und Vorgesetzten aktiv zu bewerben. Den Englischmüden unter Ihnen ist diese Übersetzung gewidmet.

Weitere Informationen über Robert findet man unter
https://www.xing.com/profile/Robert_Soesemann

Andreas Schliep (mail@andreas-schliep.de) ist als freiberuflicher Scrum Coach und Trainer international für verschiedene Unternehmen tätig.

Als Team- und Bereichsleiter bei WEB.DE half Andreas mit, eine der ersten größeren Scrum-Implementierungen in Deutschland auf die Beine zu stellen. Vorher arbeitete er als Softwareentwickler an diversen Kommunikations- und Kooperationslösungen für verteilte Arbeitsumgebungen.

Zum Vollzeit-Scrum-Coach wurde er 2006.

Weitere Informationen über Andreas findet man unter
https://www.xing.com/profile/Andreas_Schliep