Schutzgebühr 3,- Euro
Ausgabe 2010
www.agilereview.de

agile review

Das Kundenmagazin von it-agile

Feedback ist entscheidend!

XP DAYS GERMANY
ALL ABOUT AGILE

26.-28. November 2010

Jetzt vormerken

DIE Konferenz zu agiler Softwareentwicklung in Deutschland

ausgerichtet von:

andrena OBJECTS

it-agile

XP
Erfahrungen
Agil Kanban
OpenSpace
Lean Kontakte
Spaß
Soft Skills TDD
Retrospektiven
Scrum

http://www.xpdays.de

editorial

Agile Softwareentwicklung wird zum aktuellen Stand der Kunst. Immer mehr Firmen entwickeln agil. Gleichzeitig wird der Begriff Agilität aber immer weniger scharf verwendet. Manchmal bleibt von der Agilität nur, dass man „irgendwie flexibel“ ist. Das ist ja auch nicht gänzlich falsch. Ich habe aber auch schon Situationen erlebt, in denen diese Flexibilität überinterpretiert wurde: Da kommt dann der Projektleiter oder Chef in das Entwickler-Büro und will mal eben ganz schnell und flexibel, dass alles stehen und liegen gelassen wird, um für ihn ein bestimmtes Feature einzubauen. Der Preis für diese Art der Flexibilität ist Unplanbarkeit, denn was stehen und liegen gelassen wird, war genau das, was das Team geplant hatte. Agilität ist nicht Durchwursteln 2.0! Stattdessen geht es bei den meisten agilen Prozessen darum, dass die kurzen Iterationen bzw. Sprints sehr wohl geplant werden. Den Plan für diese ein bis vier Wochen kann aber nur erfüllen und einhalten, wer möglichst wenigen Störungen ausgesetzt ist.
Andererseits geht es natürlich auch darum, immer nach Prioritäten zu arbeiten und das Wichtigste und Nützlichste zu tun, was gerade getan werden kann. Das gilt dann auch für die Manager. So hatten wir durch recht optimistische, späte Planung und ein paar Krankheiten im letzten Dezember die folgende Situation: Unsere Weihnachtspost lag in Form von ein paar Kartons mit Umschlägen, Briefmarken, Weihnachtskarten, Teamkalendern und Schokoladen unbearbeitet in einem Büro. In der letzten Woche vor Weihnachten gab es aber nichts Wichtigeres als die Weihnachtspost auf den Weg zu bringen. So „traf“ es dann eben Geschäftsführer und Prokurist, die Weihnachtspost einzutüten und zur Post zu bringen. Das war vielleicht ungewöhnlich, stellte nach agilen Grundsätzen aber die beste Lösung in der gegebenen Situation dar. Nun reicht es aber nicht aus, dass wir in vielen unvorhersehbaren Situationen richtig handeln. Es gilt auch, aus diesen Situationen systematisch zu lernen. Agile Prozesse nutzen hierfür Retrospektiven. Konkret sieht nun die Planung unserer nächsten Weihnachtspostaktion sowohl einen früheren Beginn als auch eine Backup-Lösung bei der Ausführung vor.
In diesem Magazin wollen wir einen Teil unseres Wissens und unserer Erfahrungen zu agiler Softwareentwicklung mit Ihnen teilen. In „Welche Methode passt für wen?“ erhalten Sie eine Einschätzung, wann welche Methode für welche Situation angemessen ist. Einen sehr knappen Abriss über die Essenz von Scrum liefert Ihnen „Was ist Scrum?“. Da ein wesentlicher Aspekt von Scrum und anderen agilen Methoden in der Selbstorganisation von Teams liegt, beschäftigt sich der Artikel „Zwischen Selbstorganisation und Selbstillusion“ ausführlicher mit dem Thema.
Um wirklich erfolgreich mit agiler Softwareentwicklung zu sein, sind natürlich neben den methodischen auch technische Voraussetzungen zu beachten. Der Artikel „The Big Ball of Mud“ beschäftigt sich mit der Frage, wie flexible Architekturen beschaffen sein müssen. Die eher (noch) nicht so verbreitete Methode Feature-Driven Development stellen wir Ihnen in unserem FDD-Artikel vor. Wenn Sie einmal wissen möchten, wie denn der Alltag eines agilen Entwicklers aussieht, finden Sie im zweiteiligen Artikel „Die Woche eines agilen Entwicklers“ ein schönes Beispiel dafür.
In „Akzeptanztests mit FIT“ lesen Sie über die Erstellung automatisierter Akzeptanztests, die besonders in großen Systemen eine wichtige Voraussetzung für Änderbarkeit und Erweiterbarkeit darstellen. Im Anschluss finden Sie ein Interview mit dem Kanban-Begründer, David Anderson. Kanban ist ein agiles Vorgehen, das insbesondere auf die Verkürzung der Durchlaufzeiten fokussiert. Dabei unterscheidet es sich in einigen Punkten von anderen Methoden wie Scrum und lässt sich deshalb auch - aber keinesfalls ausschließlich - auf Maintenance-Prozesse anwenden. Schließlich wird im Artikel „Lean Management in der Unternehmens-IT“ anhand einiger Beispiele dargestellt, wie sich Prinzipien aus dem Lean Production System auch auf die Softwareentwicklung übertragen lassen. ■

Viel Spaß beim Lesen wünscht Ihnen Ihr

Henning Wolf, Geschäftsführer von it-agile

Inhalt

Welche agile Methode für wen?

Für viele Teams ist es nicht einfach zu entscheiden, an welcher agilen Methode und welchen agilen Praktiken sie sich bei der Einführung orientieren sollen. Dieser Artikel soll hier Hilfestellung geben. Er regt aber auch agil Erfahrene an, darüber nachzudenken, ob vielleicht für ihre Bedürfnisse und ihr Projekt-Setting neue Impulse aus bisher weniger bekannten Methoden manchmal besser geeignet wären. Von Henning Wolf und Christoph Kemp

Aus unserer Beratungspraxis wissen wir, dass kein Unternehmen und kein Team dem anderen gleicht – und erst recht kein Projekt dem anderen. Das führt auch bei der Einführung oder Durchführung agiler Softwareentwicklung dazu, dass es nicht eine gültige Antwort auf alles gibt. Wir gehen daher von individuellen Anpassungen des Entwicklungsprozesses je Projekt aus, aber erst agil erfahrene Teams können diese alleine durchführen. Diese vermeintliche Beliebigkeit birgt die Gefahr, dass agile Praktiken völlig unpassend miteinander kombiniert werden. Daher ist es sinnvoll, bei der Einführung agiler Methoden mit der Einführung einer vorhandenen Methode zu beginnen, an der man sich gut orientieren kann. Die Methoden enthalten nämlich eine passende und erprobte Zusammenstellung agiler Praktiken. Auch und gerade für das Erlernen einer neuen Methode ist Orientierung wichtig – und sogar wichtiger als die sofortige individuelle Anpassung bzw. Optimierung. Wir suchen also nach der passenden agilen Startmethode und nähern uns dieser in diesem Artikel mit folgenden Schritten:

- Zuerst stellen wir kurz und knapp die Schwerpunkte der vier agilen Methoden Scrum, XP, FDD und Crystal vor.
- Anschließend geben wir – in Abhängigkeit von den Zielen der Einführung (Qualität, Flexibilität etc.) und den organisatorischen Voraussetzungen (Teamorganisation, Projektgröße etc.) – Empfehlungen.

Schwerpunkte der Methoden

Jede agile Methode setzt einen eigenen Schwerpunkt, der bereits einen ersten Hinweis darauf liefert, ob die Methode zu einem Unternehmen, Team oder Projekt passt. Die Auswahl der im Folgenden betrachteten vier agilen Methoden haben wir wie folgt getroffen: Scrum und XP sind die mit Abstand populärsten und bekanntesten agilen Methoden im deutschsprachigen Raum. FDD ist weniger bekannt, unterscheidet sich aber in Schwerpunktsetzung und Projektorganisation stark, so dass es für viele Unternehmen eine Alternative darstellen kann. Crystal ist als Methodenfamilie hilfreich für das Verständnis, dass unterschiedliche Projekt-Settings unterschiedliche Herangehensweisen erfordern.

Scrum

Scrum (vgl. [Sch02]) setzt den Schwerpunkt auf Managementtechniken (Schätzen, Planen, Tracking). Wie das Entwicklungsteam seine tägliche Arbeit verrichtet, wird hingegen nicht adressiert. Scrum geht davon aus, dass alle auftretenden Anforderungen in einer priorisierten Liste, dem so genannten Product Backlog gesammelt werden. Das Product Backlog wird von einer Person, dem Product Owner geführt. Dieser vertritt die Kundenseite in einem Scrum-Projekt. Eine Iteration wird in Scrum Sprint genannt. Die vorgeschlagene Länge eines Sprints beträgt 30 Tage (mittlerweile hat sich eine Sprint-Länge von einer

Was ist Scrum?

Das agile Managementframework Scrum stellt heute den bekanntesten agilen Vertreter dar, weil es durch seine einfache Struktur und die klar definierten Rollen leicht verständlich ist und sich schnell produktiv einsetzen lässt.

Von Arne Roock und Henning Wolf

Selbstgesteuerte Teams

Im Mittelpunkt von Scrum steht das selbstgesteuerte Entwicklungsteam, das ohne Projektleiter auskommt. Um dem Team eine störungsfreie Arbeit zu ermöglichen, gibt es den ScrumMaster, der als Methodenfachmann dafür sorgt, dass der Entwicklungsprozess nicht abbricht. Dem Product Owner (Produktverantwortlichen) kommt die Aufgabe zu, Anforderungen zu definieren, zu priorisieren und auch zu tauschen. Allerdings ist in Scrum klar geregelt, wann er neue oder geänderte Anforderungen beauftragen darf: So gibt es ungestörte Entwicklungszyklen von wenigen Wochen (so genannte Sprints), in denen Änderungen an den Anforderungen nicht zugelassen werden. Während der Sprints arbeitet das Entwicklungsteam alle Anforderungen ab, die für diesen Sprint vorgesehen waren, und zwar timeboxed – es steht also nur ein genau festgelegter Zeitrahmen zur Verfügung und kein Tag mehr. Wie lang die Sprints dauern, wird je nach Gegebenheiten zu Beginn des Projekts festgelegt, in der Regel sind es 2 oder 4 Wochen. Nach Ende jedes Sprints werden dem Product Owner die neuen Funktionalitäten präsentiert, so dass dieser stets auf dem aktuellen Stand ist. Dem ScrumMaster kommt während der Sprints die Aufgabe zu, externe Störungen vom Entwicklungsteam fernzuhalten und auftretende Probleme, die nicht die Entwicklung betreffen, zeitnah zu lösen.

Anforderungsmanagement

Die gesamten Anforderungen an das neue System werden im Product Backlog festgehalten, in das der Product Owner jederzeit Ideen für neue Anforderungen eintragen kann. Welche Anforderungen dann jedoch tatsächlich im nächsten Sprint umgesetzt werden sollen (und in welcher Reihenfolge), wird im Planning Meeting besprochen und vom Product Owner festgelegt. Im Sprint Backlog stehen dann genau die Anforderungen für den nächsten Sprint – Änderungen am Sprint Backlog während des Sprints sind unter keinen Umständen erlaubt.

Daily Scrum

Neben dieser Rückkopplung zwischen Entwicklern und Product Owner nach jedem Sprint gibt es bei Scrum ▶

The Big Ball of Mud

„The most frequently deployed software architecture: the Big Ball of Mud", so haben Foote und Yoder das Ergebnis von 30 Jahren Software-Engineering zusammengefasst [1]. Muss das wirklich so sein? Von Andreas Havenstein

Größere Software-Systeme bestehen häufig aus verklumpten Strukturen. Strukturen, die bis zur Unkenntlichkeit erodiert sind, mit verworrener Kommunikation und Informationsverteilung über weit entfernte Elemente des Systems. Ein Klumpen voller Duplikationen und mit ungeregeltem Wachstum.

Wie kommt es dazu? Kein Softwareentwickler erstellt Code mit der Absicht, ein undurchdringliches Strukturdickicht zu erzeugen. Neben offensichtlichen Gründen wie Unerfahrenheit und Zeitdruck gibt es eine Reihe weiterer Gründe, von denen hier nur einige genannt seien:

- Rotten Neighborhood Theory: Sind bereits kleine Verklumpungen im System, gesellen sich leicht weitere schlechte Strukturen dazu, ohne aufzufallen
- Prototypen, die nicht sofort weggeworfen werden, nachdem sie ihren Demonstrationszweck erfüllt haben, rutschen leicht mit ihrer Quick&Dirty-Implementierung in den Produktivcode.
- Peter-Principle-Programming: An das Berufskarrieren-Peter-Prinzip angelehnt, gilt für Software: Die Komplexität einer Software steigt schnell bis zu genau einen Schritt hinter der Stufe, die einem Entwickler gerade noch verständlich ist [2].
- Conways Law: Eine Software ist immer so komplex wie die Strukturen der Organisation, für die sie entwickelt wird [3].

Weshalb flexible Architekturen?

Warum ist ein derart erodiertes System so schlimm? Weil es die Wartung, Pflege und Weiterentwicklung eines Systems sehr teuer macht! Jede Änderung wird erschwert durch die Unverständlichkeit der Strukturen und die hohe, schwer durchschaubare Kopplung der Systemteile. Besonders für die inkrementelle Softwareentwicklung ist es Voraussetzung, dass Änderungen zu späterer Zeit günstig durchgeführt werden können, damit spätere Inkremente nicht deutlich teurer werden als die Inkremente der Anfangszeit.

Und wenn man den Begriff „inkrementell" etwas lockerer fasst, dann fallen sogar die klassisch per Wasserfall-Entwicklung erstellten Systeme darunter, da nahezu jedes System nach der Erstauslieferung korrigiert und erweitert wird, und damit also auch neue Inkremente erstellt werden.

Prinzipien einer flexiblen Architektur

Wie kann die beschriebene System-Erosion vermieden werden?

Für die Änderbarkeit von Software sind Verständlichkeit und Flexibilität die obersten Prinzipien. Die Kriterien eines flexiblen Systems sind unter anderem:

- Unabhängige Bestandteile (Modularisierung)
- Leicht kombinier-, erweiter- und änderbar

Zwischen Selbstorganisation und Selbstillusion

Wie man ein Scrum-Team führt, ohne es zu führen

Von Stefan Roock

Projektleiter

Wir schreiben das Jahr 2000. Die Welt ist auf Projektleiter getrimmt. Die Kunden und das eigene Management erwarten genau einen Verantwortlichen für das Projekt – den Projektleiter. Kent Beck veröffentlicht eXtreme Programming (XP), eine damals ziemlich eigenartig anmutende Methode zur Softwareentwicklung. Zwei Leute sollen zusammen vor einem Rechner sitzen – das nennt sich dann „Pair Programming". Alle Entwickler im Team sollen gemeinsam für den ganzen Code verantwortlich sein – Collective Ownership. Und so weiter. Und dann gibt es in XP keine Projektleiterrolle. Schräg, aber irgendwie auch attraktiv. Doch wie soll das gehen? Wer verhandelt dann mit dem Kunden? Wer ist gegenüber dem eigenen Management verantwortlich? Naja, XP geht ja auch mit Projektleiter. Mit Pair Programming, Collective Ownership und den anderen Techniken haben wir ja bereits genug zu tun. Dann bleiben wir doch beim guten alten Projektleiter.

Selbstorganisation ...

Der Wirbel um eXtreme Programming lässt über die Jahre langsam nach. Jetzt steht Scrum stärker im Fokus. Auch Scrum kennt keinen Projektleiter. Im Gegensatz zu eXtreme Programming ist Scrum aber expliziter: Die Abwesenheit des Projektleiters ist gewollt. Stattdessen legt Scrum sehr viel Wert auf Selbstorganisation des Teams. So war das auch in eXtreme Programming gedacht. Allerdings täuscht die Fülle der XP-Techniken schnell über diesen schwierigen Aspekt hinweg. Sollen wir also einmal das Risiko eingehen und das Team sich selbst organisieren lassen? Das machen jetzt ja anscheinend alle. So richtig trauen wir uns aber noch nicht. „Hallo, ihr vier! Ja, ihr. Ihr seid unser Scrum-Team. Doch, doch, einen Projektleiter gibt es weiterhin. Er ist aber nur ein Sicherheitsnetz und arbeitet eigentlich als normaler Entwickler mit. Ihr sollt Euch schon selbst organisieren. Der Projektleiter kann euch dabei ja unterstützen. Wenn die Selbstorganisation gut funktioniert, können wir in zukünftigen Projekten aber auch ganz auf den Projektleiter verzichten."

... klappt nicht

Bei der Retrospektive nach Projektende stellen wir fest, dass die Selbstorganisation nicht wirklich eingetreten ist. Es gab immer wieder Konflikte zwischen den Teammitgliedern. Diese konnten sich einfach nicht einigen. Zwei gegen zwei. Und dann musste doch wieder der Projektleiter entscheiden. Die Entwickler haben geradezu danach gebettelt. „Bitte, bitte, bitte, lieber Projektleiter, rette uns und entscheide! Wir können uns einfach nicht einigen." Also ist Selbstorganisation doch nur ein Hirngespinst oder im besten Fall ein Glücksfall?

Der Psycho

Heute haben wir ein Seminar bei so einem Psycho-Heini. Ich steh' da nicht so drauf. Dieser ganze Kram mit den weichen Faktoren. Naja, was soll's. Vielleicht immer noch besser, als den ganzen Tag in Meetings zu sitzen. Der Psycho sagt, dass sich jede Gruppe selbst organisiert. Cool! Dummerweise tut sie das irgendwie. Das diese selbstgewählte Organisation irgendetwas mit dem Ziel des Projekts zu tun hat, ist tatsächlich Zufall. ▶

it-agile

Flexible Architekturen

Reagieren auf neue und geänderte Anforderungen

Nachhaltigkeit durch Flexibilität – Flexibilität durch

- *Verständlichkeit*
- *Redundanzfreiheit*
- *Entkopplung*

Schulungen in Hamburg:

- *Hoher Praxisanteil*
- *Erfahrene Trainer*
- *Auch inhouse möglich*

TDD CAMP

Konstruktive Qualitätssicherung mit testgetriebener Entwicklung (TDD)

Testgetriebene Entwicklung hilft,

- *weil man nur das entwickelt, was auch benötigt wird.*
- *weil alles, was entwickelt wird, auch abgetestet ist.*
- *weil es zu einfacheren Entwürfen und Lösungen führt.*

http://www.it-agile.de

Mit Ordnung zum Ziel

In diesem Artikel wird mit Feature Driven Development (FDD) eine agile Methode beschrieben, die explizit eine vorgelagerte Analyse- und Modellierungsphase vorsieht und für große sowie für Festpreisprojekte gut geeignet ist. Von Stefan Roock und Henning Wolf

Im Gegensatz zu Scrum oder eXtreme Programming (XP) ist FDD weit weniger bekannt und zumindest in Europa auch in der Praxis wenig verbreitet. Das ist insofern ein änderungswürdiger Zustand, als dass FDD interessante andere Schwerpunkte setzt als Scrum oder XP.

Gerade dadurch bietet FDD Antworten auf viele Fragen, die bei Scrum oder XP offen bleiben. Auch wenn Ihr Projekt nicht pur FDD wird, können Sie nützliche Anregungen und Techniken aus FDD mitnehmen und für Ihre Projektsituation verwenden.

FDD wurde als schlanke Methode von Jeff De Luca im Jahre 1997 definiert, um ein großes, zeitkritisches Projekt (15 Monate, 50 Entwickler) durchzuführen. Seitdem wurde FDD kontinuierlich weiterentwickelt. FDD stellt den Feature-Begriff in den Mittelpunkt der Entwicklung. Features sind sehr klein, für den Kunden verständlich und stellen jeweils einen Mehrwert für den Kunden dar. Die Entwicklung wird anhand des Feature-Plans organisiert. Eine wichtige Rolle spielt der Chefarchitekt (engl. Chief Architect), der ständig den Überblick über die Gesamtarchitektur und die fachlichen Kernmodelle behält. Bei größeren Teams werden einzelne Entwicklerteams von Chefprogrammierern (engl. Chief Programmer) geführt.

FDD definiert ein Prozess- und ein Rollenmodell, das gut mit existierenden klassischen Projektstrukturen harmoniert. Daher fällt es vielen Unternehmen leichter, FDD einzuführen als XP oder Scrum. Außerdem ist FDD ganz im Sinne der agilen Methoden sehr kompakt und lässt sich auf 10 Seiten komplett beschreiben [1].

FDD-Prozessmodell

FDD-Projekte durchlaufen fünf Prozesse (vgl. Abb. 1):

- Prozess #1: Entwickle ein Gesamtmodell. Rollen: alle Projektbeteiligten
- Prozess #2: Erstelle eine Feature-Liste. Rollen: in der Regel nur die Chefprogrammierer
- Prozess #3: Plane je Feature. Rollen: Projektleiter, Entwicklungsleiter, Chefprogrammierer
- Prozess #4: Entwirf je Feature. Rollen: Chefprogrammierer, Entwickler
- Prozess #5: Konstruiere je Feature. Rollen: Entwickler

Impressum:

Chefredaktion (verantwortlich): Arne Roock

Redaktionsadresse: it-agile GmbH, Paul-Stritter-Weg 5, 22297 Hamburg, www.it-agile.de

Gerichtsstand und Erfüllungsort: Hamburg

Layout: Jasna Wittmann Kommunikationsdesign, jasnawittmann@googlemail.com

Titelfoto: Christian Kalnbach

Autoren: Alex Bepple, Andreas Havenstein, Christoph Kemp, Arne Roock, Stefan Roock, Henning Wolf

Druck:

Bildnachweise:
S. 19: ©iStockphoto.com/westhoff
S. 21, Abb3: http://www.xdepend.com/images/Capture16-12-2008-12.20.37.png
S. 24: Fotolia.de
S. 39: Fotolia.de
S. 58/59: ©iStockphoto.com/ricardoazoury

Alle anderen Fotos und Grafiken: it-agile

Wir danken SIGS DATACOM, dem Software & Support Verlag und dem dpunkt Verlag für die Erlaubnis zum Abdruck der Artikel.

Geben Sie uns Feedback!

- was hat Ihnen gut gefallen?
- was können wir verbessern?

Schreiben Sie an:
arne.roock@it-agile.de

Bild von Christian

Agil hautnah: Die Woche eines agilen Entwicklers

Entwickeln zwischen Testen, inkrementellem Design, Architekturdiskussionen, Aufwandsschätzungen, Fertigstellungsprognosen und Abstimmungen im Team

Von Alex Bepple und Henning Wolf

Klar hat man schon von agilen Methoden gehört, manches machen die meisten längst agil. Aber wie sieht es denn nun konkret aus, wenn die Projektmanagementmethode Scrum heißt oder nach eXtreme Programming oder Feature Driven Development entwickelt wird?

Führt agile Softwareentwicklung zu besserer Qualität und hoher Wartbarkeit? Oder ist hohe Qualität und gute Wartbarkeit Voraussetzung für agile Softwareentwicklung?

Der Artikel beschreibt aus der Sicht der Entwickler, welche Herausforderungen auf sie zukommen: Wie agil entwickelt wird; wie sich das Design ergibt; wer, wann, wo und wie über Architekturen diskutiert; wie die Qualitätssicherung erfolgt und sich alles immer noch mehr zum Guten wenden soll.

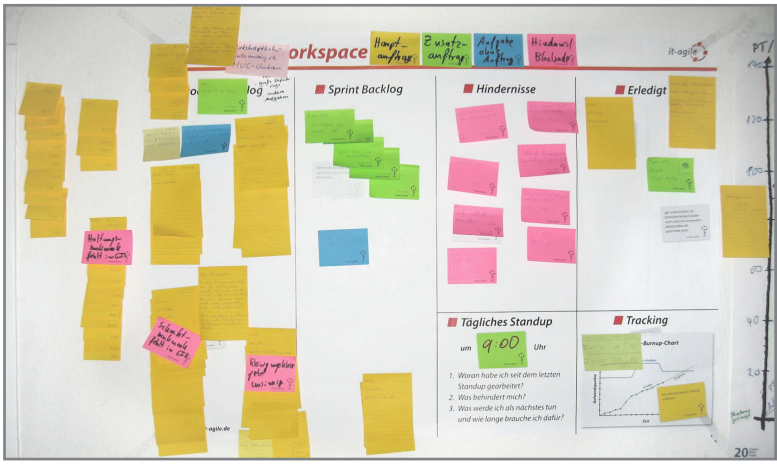
Was haben Entwickler davon? Wirklich weniger Stress und Hektik? Steht nicht ständig der nächste Termin an?

Montagmorgen: Das Setting

Seit ein paar Monaten fahre ich nicht mehr ins Büro meines Arbeitgebers, sondern direkt zu unserem Kunden. Dort habe ich mit meinen Entwicklerkollegen seit Projektbeginn meinen Arbeitsplatz, und direkt gegenüber auf dem Flur sitzt unser Hauptansprechpartner, Herr A. Das ist ziemlich praktisch, wenn man mal eine Frage hat. Das scheint dem Kunden auch so zu gehen, denn er kommt gar nicht so selten mit Fragen zu uns. Oft beziehen die sich dann auf zukünftige Features. Das ist OK, kann aber auch mal zu viel werden, so dass wir eine Zeit lang schon „störungsfreie Zeiten“ vereinbart hatten: jeweils vormittags und nachmittags zwei Stunden am Stück ohne wechselseitige Unterbrechungen.

Wir sind fünf Entwickler im Team und sitzen in einem Raum, der auch nur knapp für uns reicht in der Größe. Der Raum hat drei Schreibtische, und wir sitzen jeweils zu zweit an einem Schreibtisch zum Programmieren in Paaren, also dem gemeinsamen Entwickeln. Wir könnten auch mehr Platz haben – wir haben noch einen Nebenraum, aber den nutzen wir nur selten für Besprechungen oder so. Da stehen auch noch zwei Rechner, wenn wir mal nicht in Pairs arbeiten oder jeder seine Mail bearbeitet. Anfangs dachte ich, dass ich den Krach und die Nähe nicht lange aushalte. Mittlerweile kann ich mir aber gar nicht mehr vorstellen, anders zu arbeiten. Und wirklich laut ist es äußerst selten. Es ist sogar sehr nützlich, dass man manchmal durch Gesprächsfetzen der anderen Entwickler mitbekommt, was sie gerade tun. Dann kann man schnell seinen Senf dazugeben und kommt gemeinsam zu besseren Lösungen. Man kann auch sehr schnell Probleme diskutieren und Fragen stellen. Sonst würde man öfter 15

Abbildung 1: Informative Workspace



Automatisierte Akzeptanztests mit FIT

Von Stefan Roock

Akzeptanztests testen ein Softwaresystem aus Anwendersicht auf Akzeptanzkriterien. Das ist ein alter Hut. Dass man einen Großteil dieser Tests – nämlich mindestens den Test jeglicher Funktionalität – automatisieren kann und sollte, ist da schon eine etwas neuere Idee. Und dass mit dem Open-Source-Werkzeug FIT dafür ein einfaches und mächtiges Instrument verfügbar ist, ist noch viel zu wenig bekannt. Idealerweise werden auch die Fachexperten in das Erstellen der Tests involviert, daher ist dieser Artikel so geschrieben, dass er auch für Fachexperten verständlich sein sollte.

Die Bedeutung von Akzeptanztests

Mit Akzeptanztests* wird die Systemfunktionalität aus Sicht der Anwender überprüft. In jedem Softwareprojekt werden Akzeptanztests durchgeführt. In den meisten Fällen setzen sich dazu ausgewählte Fachexperten an das System und testen manuell. Teilweise werden die Tests ad hoc, ohne vorherige Planung, durchgeführt. Mindestens bei größeren Systemen ist jedoch eine Planung notwendig, so dass ein Testkonzept erstellt wird.

Das manuelle Testen bedeutet bei den modernen iterativen (agilen) Vorgehensweisen, dass die Fachexperten sehr häufig testen müssen. Da mit jedem Inkrement vorhandene Funktionalität prinzipiell in Mitleidenschaft gezogen werden kann, müsste eigentlich nach jeder Iteration das gesamte System getestet werden. Bei Releasezyklen von 1 bis 3 Monaten führt das schnell zu hohen Testaufwänden und zu Frustration bei den Fachexperten – wer hat schon Lust, alle sechs Wochen dieselbe Systemfunktionalität zu testen?

Daher wäre es am einfachsten, wenn Akzeptanztests möglichst weit automatisiert werden. Die sowieso notwendigen Testkonzepte bieten dafür eine geeignete Grundlage, so dass für die Automatisierung nur geringer Mehraufwand entsteht. Und für die Automatisierung selbst bietet sich FIT an.

Was ist FIT?

FIT steht für *Framework for Integrated Tests* und ist ein frei verfügbares Open-Source-Werkzeug, das den Ansatz verfolgt, technische Testausführung und Testdaten voneinander zu trennen. Die Testdaten werden in Tabellen beschrieben, die Testausführung wird über sogenannte Fixtures programmiert (dazu unten mehr).

Zahl 1	*Zahl 2*	*Summe*	*Differenz*
3	*2*	*5*	*1*
0	*0*	*0*	*0*

Tabelle 1: Testdaten für einen Taschenrechner

Beginnen wir mit einem ganz einfachen Fall. Tabelle 1 testet einen ganz einfachen Taschenrechner, der nur addieren und subtrahieren kann. Die erste Zeile der Tabelle stellt die Spaltenüberschriften dar. Aus ihnen können wir ersehen, was die Eingabedaten (Zahl1, Zahl2) und was die erwarteten Ausgabedaten (Summe, Differenz) sind.

Dass die Spalten Zahl1 und Zahl2 die Eingabewerte und Summe und Differenz die Ausgabewerte der Systemfunktion sind, ist in der Tabelle nicht eindeutig erkennbar. Wir

„Scrum funktioniert nicht überall“

Interview mit David Anderson zu Kanban

Von Arne Roock

Abbildung 1: Kanban-Boards visualsieren den Workflow und machen Bottlenecks und andere Probleme schnell sichtbar

funktionierendes Produkt-Inkrement vorgestellt, und es gibt eine Retrospektive. Dies alles ist sehr gut! Die Herausforderung dabei liegt im Rhythmus: Vielleicht passen 4 Wochen nicht, und 2 Wochen sind besser? Ich glaube nicht daran, dass Scrum in jeder Situation funktioniert. Viele Leute besuchen Kanban-Schulungen, weil sie in Umgebungen arbeiten, für die Scurm nicht besonders gut geeignet ist, z. B. Systemadministration und Betrieb. Die Vorstellung, Arbeit für 2 oder 4 Wochen in ein Backlog zu füllen, ist völlig befremdlich für sie. Es ist verständlich, warum Scrum so ein großer Erfolg ist. Aber es funktioniert offensichtlich nicht überall.

Sind Scrum und Kanban konkurrierende Methoden, oder ergänzen sie sich gegenseitig?

Ich sehe sie nicht als konkurrierende Methoden. Denn Kanban ist gar keine Entwicklungsmethode oder Projektmanagement-Methode. Im Moment kann man beobachten, wie die Scrum-Community eine Neu-Definition von Scrum vornimmt. Und dabei gibt es konkurrierende Neu-Definitionen. Wenn man die frühen Bücher über Scrum liest, dann ist es im Wesentlichen eine Projektmanagement-Methode. Tatsächlich finden sich dort keine Entwicklungspraktiken. Vielmehr sagen vielen Scrum-Leute: Man braucht zusätzlich zu Scrum XP-Techniken, damit man Entwicklung und Projektmanagement zusammen hat. Kanban ist keins von beiden. Es ist weder eine Entwicklungsmethode noch eine Projektmanagement-Methode. Es ist eine Change-Management-Methode, eine Art, um Änderungen herbeizuführen, die zu einem schlanken Unternehmen führen, zu koninuierlichen Verbesserungen, zu einer Kaizen-Kultur. Es ist ein Weg, um inkrementelle, evolutionäre Änderungen in einer Organisation herbeizuführen, anstatt die Art zu arbeiten zu revolutionieren. Man verwendet nicht Kanban allein. Ich bekomme häufig die Frage gestellt: „Wir stellen ein neues Team zusammen und haben uns noch nicht für eine Methode entschieden. Sollen wir Scrum oder Kanban verwenden?“ Meiner Meinung nach ergibt diese Frage überhaupt keinen Sinn, denn man kann Kanban nur auf etwas bereits Bestehendes aufsetzen. Das erste Team, mit dem wir Kanban angewendet haben, war ein PSP/TSP-Team, das als ausgelagerter Lieferant in Indien für Microsoft gearbeitet hat. Dort haben wir Kanban aufgesetzt, ohne an diesem Prozess etwas zu ändern. Wir konnten die Produktivität verdreifachen und die Durchlaufzeit um 90% reduzieren. Das nächste Team, mit dem wir Kanban ausprobiert haben, war ein gutes altes Wasserfall-Team, also eine Methode, die aus den 1970ern stammt. Seitdem habe ich viele Teams gesehen, die in unterschiedlichen Situationen Kanban eingeführt haben. Und es gibt Teams, die Kanban auf Scrum aufgesetzt haben. Das Problem dabei ist: Wenn man Scrum um Kanban erweitert, wird sich Scrum sehr schnell verändern. Das Team beginnt Fragen zu stellen: „Warum machen wir 2-wöchige Iterationen?“ Kanban hilft ihnen, Dinge zu entkoppeln: Der Input, also die Priorisierung und die Planung, ist unabhängig vom Output, also dem Release-Mechanismus. Ebenso die Durchlaufzeit. In Scrum sind diese drei Dinge gekoppelt. Wenn man einen 2-Wochen-Sprint hat, dann plant man auch alle zwei Wochen; die Durchlaufzeit beträgt zwei Wochen; und

Was ist Kanban?

Kanban ist ein Vorgehen, das Prinzipien aus dem Lean Thinking mit Erkenntnissen aus der Theory of Constraints vereint und den Fokus auf die Verkürzung von Durchlaufzeiten legt. Hierzu werden die verschiedenen Prozessschritte der Wertschöpfungskette auf einem großen Whiteboard abgebildet. Die einzelnen Features/Storys werden auf Karteikarten oder Haftnotizen festgehalten, und diese Tickets wandern von links nach rechts über das Whiteboard. Nun wird die Menge an Tickets beschränkt, die gleichzeitig an einer Station bearbeitet werden darf. Außerdem wird ein Pull-System etabliert - fertige Tickets werden niemals an die nächste Station übergeben, sondern dürfen stets nur von nachfolgenden Stationen gezogen werden. Auf diese Weise wird sehr schnell deutlich, wo sich Engpässe und andere Probleme befinden, und es können geeignete Verbesserungsmaßnahmen ergriffen werden. In Kanban wird darüber hinaus großer Wert auf eine Kultur der kontinuierlichen Verbesserung („Kaizen") und hohe Qualität gelegt (weil sich nur durch hohe Qualität wirklich hohe Geschwindigkeit erreichen lässt).

der Release-Zyklus beträgt auch zwei Wochen. Für einige Organisationen ist das aber nicht die beste Wahl. Wenn man Kanban einführt, fragt man sich sehr bald, was der richtige Rhythmus ist, und man entkoppelt Input, Durchlaufzeit und Output. Wenn man aber aufhört, gleich lange Sprints durchzuführen, egal ob 2 oder 4 Wochen, ist es dann noch Scrum?

Viele Leute sagen, dass Retrospektiven das Herzstück agiler Vorgehensweisen darstellen. Gibt es auch in Kanban Retrospektiven?

Es ist ganz offensichtlich, dass man nichts lernen kann, so lange man sich nicht die Zeit nimmt, anzuhalten und zu reflektieren. In Kanban ermutige ich die Teams dazu, auf einer mehr organisatorischen Ebene zu reflektieren. Das nenne ich Operations Reviews. Das ist nicht neu, ich habe dieses Konzept nicht erfunden. Worin sich Operations Reviews und typische agile Retrospektiven unterscheiden, ist der Grad der Objektivität. Für ein Operations Review benötigt man in der Regel Manager, die Daten mitbringen. Daten, die sich auf den Workflow beziehen, die Fehlerraten, die Durchlaufzeiten und so weiter. Und diese Daten werden dann bei den Operations Reviews vorgestellt. Hier sieht man sich beispielsweise Cumulative Flow Diagrams an oder die Anzahl geblockter Tasks. Dies sind alles harte Fakten, objektive Daten. Agile Retrospektiven haben eine Tendenz, viel subjektiver zu sein. Wie ging es uns beim letzten Sprint? Was gefiel uns? Was gefiel uns nicht? Was wollen wir wieder so machen? Was wollen wir anders machen? Bei erfahreneren Teams werden auch die Retrospektiven immer objektiver. Aber bei Kanban-Teams versuche ich, diese Objektivität von Anfang an zu etablieren. Und hier sind wir auch wieder beim Entkopplungs-Thema: Agile Methoden koppeln Retrospektiven an Releases. Wenn man also zweiwöchige Iterationen hat, dann führt man auch alle zwei Wochen eine Retrospektive durch. Das kann in Ordnung sein. Aber was ist, wenn man sich nur einmal im Monat zusammensetzen möchte? Oder jede Woche? Oder in irgendeinem anderen Rhythmus? Ich arbeite darauf hin, dass sich Teams ihren eigenen Rhythmus für ihre Operations Reviews suchen, dass dieser Rhythmus entkoppelt ist vom Release- und Planungsrhythmus und dass diese Meetings auf einer organisatorischen Ebene

„Sehr gute Kanban-Teams führen täglich Retrospektiven durch"

stattfinden. Es geht nicht um die Ebene einzelner Projekte. Holt die gesamte Abteilung zusammen, alle sollen gemeinsam an den Reports arbeiten! Der Grund, warum das gut ist, liegt darin, dass sich unterschiedliche Projekte so gegenseitig befruchten können. Und so wird organisatorische Entwicklung und organisatorische Reife gefördert. Ich glaube, das ist der Grund, warum man bei vielen Kanban-Teams beschleunigte Entwicklung feststellen kann. Aber ich habe auch schon Kanban-Teams gesehen, die Retrospektiven auf Projektebene oder Iterationsebene durchführen. Wenn sie ein Release hinter sich haben, in dem etwas schief ging, dann analysieren sie, was da schief gelaufen ist und wie sie das in Zukunft verhindern ▶

können. Und es gibt etwas, das durch Case Studies bekannt wurde, und gar nicht so sehr von den Vordenkern beschrieben wurde: Im Internet schildern Leute, dass ihr Team aufgehört hat, Retrospektiven durchzuführen. Sie machen zwar noch Operations Reviews, aber keine Retrospektiven mehr. Und wenn man dann analysiert, warum das so ist, findet man heraus, dass sich viele Aktivitäten aus der Retrospektive in das tägliche Standup-Meeting verlagert haben. Weil sie jeden Tag über Prozessverbesserung reden, brauchen sie es nicht mehr am Ende jeder Iteration. Man macht es täglich anstatt alle zwei Wochen oder einmal im Monat. Sehr gute Kanban-Teams, die eine wirkliche Kaizen-Kultur entwickelt haben, also einen kontinuierlichen Verbesserungsprozess, die führen jeden Tag Retrospektiven durch. Der Wert von Retrospektiven wird in Kanban also reduziert, weil sie ersetzt werden durch eine Kombination aus täglichen Standup-Meetings und Operations Reviews.

Ein anderer interessanter Punkt bezieht sich auf die Schätzungen. Stimmt es, dass in Kanban komplett auf Schätzungen verzichtet wird?

Das ist eine gute Frage, denn in den ersten Beispielen haben wir tatsächlich nicht geschätzt. Es hat sich herausgestellt, dass Schätzungen optional sind. Wenn du schätzen möchtest, dann tu es! Oft schätzen Leute, um ein Lieferdatum zu versprechen, auf das sich dann Leute verlassen. Aber nicht jede Tätigkeit, die man verrichtet, braucht ein hartes Lieferdatum. Ich höre Leute immer wieder sagen: „Wenn keine Commitments bzgl. des Lieferdatums gemacht werden, dann kann man auch niemanden verantwortlich machen." Meine Einstellung dazu ist: Man setzt ein künstliches Ziel und zwingt das Team dann, es auch zu erreichen. Das führt häufig zu Heldentum. In der Prozessverbesserung und in der Qualitätssicherung gibt es viele Hinweise auf diese Konstellation: Wenn man ein Ziel setzt und erwartet, dass dieses Ziel auch erreicht wird, dann reagieren die Leute darauf mit Heldentum. Aber wer sich heldenhaft benimmt, der hört auf, sich zu verbessern. Deshalb lautet ein Ratschlag von Goldratt und einer ganzen Reihe anderer moderner Vordenker, dass man auf solche Ziele verzichten soll. Ich habe den Eindruck, dass Schätzungen zu Deadlines führen und zu Commitments, die dysfunktional und nicht erstrebenswert sind.

*„Wer sich heldenhaft benimmt,
der hört auf, sich zu verbessern"*

Auf der anderen Seite gibt es Situationen, in denen man einen bestimmten Liefertermin definitiv einhalten muss. Zum Beispiel muss eine Investment-Bank eine neue Anforderung umsetzen, weil die Regierung einige Regeln geändert hat und diese neuen Regeln am 1. Oktober in Kraft treten. Wenn diese Bank bis zum 1. Oktober nicht die neuen Regeln einhält, darf sie bestimmte Wertpapiere nicht mehr handeln. Das sind sehr hohe Kosten, die da drohen. Dann ist es ganz klar, dass man sicher stellen muss, das neue Feature pünktlich auszuliefern. Für diese spezielle Anforderung ist es sinnvoll, zu schätzen, wie lange die Umsetzung dauern wird. Wenn wir schätzen, dass die Entwicklung sechs Wochen dauert und wir die neue Funktionalität bis zum 1. Oktober benötigen, dann ist es sinnvoll, Anfang August mit der Entwicklung zu beginnen. Es ist nicht sinnvoll, schon im Mai zu beginnen, denn ein früher Start schafft hier absolut keinen Wert. Aber wenn man zu spät ist und erst im November liefert, dann fährt man große Verluste ein. Kanban sagt, dass man Schätzungen angemessen nutzen sollte. Man muss sich Gedanken darüber machen, wann Schätzungen sinnvoll sind und diese dann intelligent nutzen. Schätzungen sind also optional. Anstatt alles zu schätzen, schätzen wir nur die wirklich wichtigen Dinge.

Die Grundidee zu Kanban stammt aus der Automobilproduktion, wo wir es mit sich stets wiederholenden, gleichförmigen Abläufen zu tun haben. Softwareprojekte hingegen haben ja häu-

fig eher den Charakter von Produktentwicklung, also eines sehr kreativen und individuellen Prozesses. Ist Kanban überhaupt für Softwareprojekte geeignet, oder liegen die wahren Stärken von Kanban nicht eher in den Bereichen Betrieb und Systemadministration?

Das ist eine gute Frage. Meistens höre ich diese Frage von Leuten aus der Produktion, die skeptisch sind. Kanban ist jedoch ganz eindeutig für Softwareprojekte geeignet! Dafür sehen wir zu viele Leute, die Kanban erfolgreich einsetzen. Aber es taucht eine Reihe interessanter Fragen auf: Warum funktioniert es, da doch Softwareentwicklung viel mehr Variabilität aufweist als Produktion? Und dies führt zu der Frage, wie Kanban auf einen anderen Kontext angepasst wurde, nämlich die Softwareentwicklung. Software-Kanban hat zwar mit denselben Prinzipien wie die Produktion angefangen, aber daraus sind unterschied-liche Praktiken entstanden. Wir haben nicht versucht, die Praktiken aus der Produktion 1:1 zu kopieren. In Wahrheit haben wir sie gar nicht kopiert, sondern wir haben uns die dahinter liegenden Prinzipien angesehen und diese auf die Softwareentwicklung angewendet, um so unsere eigenen Praktiken zu entwickeln. Deshalb kann man nicht wirklich sagen, dass Software-Kanban aus der Produktion entstanden ist.

Vielen Dank für das Interview, David! Abschließende Worte zu Kanban?

Danke, dass ich hier sein durfte! Ich hoffe, die Antworten sind nützlich und können ein wenig Licht ins Dunkel bringen. Mein Ratschlag an alle, die an Kanban interessiert sind lautet: Probiert es aus! Viele Aspekte von Kanban sind zunächst kontraintuitiv. Der einzige Weg, mehr darüber herauszufinden, ist es auszuprobieren! ■

Das vollständige Interview ist im OBJEKTSpektrum 02/2010 erschienen. Abdruck mit freundlicher Genehmigung von SIGS DATACOM (www.sigs-datacom.de)

David J. Anderson gilt als „Vater" des Kanban-Ansatzes in der Softwareentwicklung. Zusammen mit Jeff De Luca hat er in den 1990er Jahren die agile Methode Feature Driven Development entwickelt. Sein Fokus liegt heute darauf, Agilität auch in großen Unternehmen einzuführen, indem er das CMMI-Modell für organisatorische Reife mit Agilen Ansätzen und Lean verbindet. David Anderson leitet er eine Management- Beratung in Seattle.

Lean-Management in der Unternehmens-IT – Lernen von den Autobauern

Der Ruf der IT ist schlecht: Viele Manager meinen, dass IT-Projekten ständig der Misserfolg droht. IT sei überdies zu teuer und unflexibel. Auf der anderen Seite steigt die strategische Bedeutung der IT-Unterstützung in den meisten Organisationen. Von Henning Wolf

nachdenken • klimabewusst reisen

atmosfair

Jeder Flug trägt zur Klimaerwärmung bei, doch oft gibt es außer Verzicht keine praktikable Alternative. Mit einer Spende an atmosfair können Sie Klimaschutzprojekte finanzieren, welche die durch Ihren Flug verursachten Klimagase wieder einsparen. atmosfair unterstützt mit den eingenommenen Spenden ausschließlich erneuerbare Energien, wie Solarlampen in Indien, oder Energiesparmaßnahmen, wie effiziente Kocher in Afrika. Die Maßnahmen helfen nicht nur dem Klima, sie verbessern auch die lokalen Umwelt- und Lebensbedingungen für die Menschen vor Ort.

Die Kantinen indischer Schulen und Krankenhäuser werden mit modernen Solarspiegeln ausgestattet und müssen nicht mehr mit Dieselbrennern kochen. Dies spart CO_2, schafft neue Arbeitsplätze und unterstützt die Verbreitung von neuen Technologien.

www.atmosfair.de

it-agile fliegt konsequent mit atmosfair und hat diese Anzeige finanziert.