JAVA Mag inkl. CD

Transparenz und Qualitätskontrolle in agilen Projekten »108

S&S

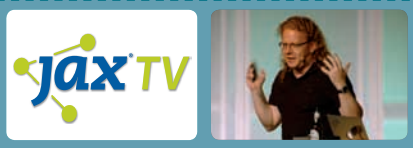
Deutschland €8,50 Österreich €9,80 Schweiz sFr 16,80 7.2011

Java™ magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

CD-INHALT



HTML5

Keynote von Chris Heilman auf der JAX 2011

HIGHLIGHT

Fabasoft app.test

WEITERE INHALTE

- Spring 3.1 M1
- Scala 2.9
- JBoss Weld

uvm.

Alle CD-Infos ab Seite 3

4 194586 708507 07

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

Contexts and Dependency Injection
Framework-Integration »91

Spring 3.1
Was gibt's Neues beim Framework? »21

W-JAX 11
Brandneues Programm »15

RabbitMQ, AMQP und Spring
Die Zukunft des Messaging »80

Präsentation mit Models
GUI leicht gemacht »52

Das Java Magazin Tutorial

Integriertes Monitoring
Admin, wir haben ein Problem! »98

Transparenz und Qualitätskontrolle in agilen Projekten

Agile Selbstheilungsmechanismen

Agile Methoden sind das aktuelle Heilsversprechen der Softwaretechnik. Agiles Vorgehen ist heutzutage nicht mehr revolutionär, sondern in weiten Kreisen etabliert und anerkannt. Und das zu Recht, denn agile Techniken besitzen viele Vorteile. Und doch zeigt die Erfahrung, dass agile Techniken allein keinen agilen Prozess garantieren. Trotz User-Stories, Timeboxing und Daily Scrums feiern altbekannte Probleme, die doch mit dem agilen Ansatz gebannt sein sollten, fröhliche Wiederkehr: Budgets und Deadlines werden überzogen, die Qualität stimmt nicht, der Kunde bleibt auf nicht umgesetzten Kernanforderungen sitzen usw. Wir werfen in unserem Artikel einen kritischen Blick auf den Qualitätsaspekt und zeigen Techniken auf, um durch hohe Qualität Planungssicherheit und eine hohe Teamgeschwindigkeit zu erreichen.

von Jörn Koch und Sebastian Middeke

Agile Projekte sind selbstorganisierend. Allerdings sehen wir auch, dass diese Selbstorganisation nur innerhalb der Grenzen des jeweiligen Projekts greift. Wir erleben oft, dass Innensicht des agilen Projekts und Außensicht des Unternehmens auf das agile Projekt kollidieren. Gerade im Konfliktfall zeigt sich, dass das Projektgeschehen sehr intransparent ist und eine gemeinsame Sicht fehlt.

Transparenz lässt sich über die üblichen agilen Kennzahlen nicht immer ausreichend herstellen. Das ist überraschend, da Feedback doch der zentrale Mechanismus agilen Vorgehens ist. Das agile Feedback trägt offenbar nicht weit genug nach außen. Aber auch die Innensicht ist nicht vor Fehleinschätzungen des Projektzustands geschützt: Besonders oft beobachten wir, dass ein voll ausgelastetes Team der Meinung ist, dass alles „perfekt rundläuft“ und doch weniger produktiv ist, als es die Zahl der bearbeiteten Stories und Bugs vermuten lässt.

Mit den im Folgenden beschriebenen Mitteln können Sie für agile Projekte Problemsituationen sicherer erkennen, einschätzen und deutlich kommunizieren. So können Sie im Bedarfsfall leichter Korrekturmaßnahmen einfordern, die außerhalb der Entscheidungskompetenzen des Projekts liegen.

Qualität bedeutet Planungssicherheit

Um Innen- und Außensicht zusammenzubringen, braucht es Gemeinsamkeiten. Das gemeinsame Verständnis des Projektziels – also eine Software in der geforderten Qualität „on time, on budget“ zu produzieren – ist dafür ein sehr guter Ansatzpunkt: Diese gemeinsame Sicht entsteht, wenn aus Innen- und Außensicht gleichermaßen „transparent“ (erkennbar) ist, inwieweit das Projektziel voraussichtlich erreicht oder verfehlt wird.

Wie erfolgreich ein agiles Projekt durchgeführt wurde, erkennen wir anhand der klassischen vier XP-Variablen „Cost, Time, Quality and Scope“, die allesamt im geforderten Rahmen liegen müssen. Dabei spielt die Qualität allerdings eine Sonderrolle: Sie muss über die gesamte Projektlaufzeit stimmen, um relevantes Feedback erwarten zu können. Prototypisches, Halbfertiges und Fehlerhaftes lässt sich nicht testen und abnehmen. Die *kontinuierliche Qualitätskontrolle* (*QK*) nimmt somit eine zentrale Position im agilen Projekt ein und liefert neben dem Nachweis einer hohen Produktqualität auch wichtige Anhaltspunkte zur Beurteilung des Projektzustands.

Eine hohe Qualität ist aber auch für eine hohe Entwicklungsgeschwindigkeit notwendig. Das scheint auf dem ersten Blick paradox, denn es ist naheliegend, an der Qualität zu sparen, wenn z. B. die Zeit knapp wird (Stichwort „Quick & Dirty“). Diese Rechnung geht jedoch nur auf, wenn die „Mindestqualität“, die der Kunde erwartet, nicht unterschritten wird.

Wird sie beispielsweise durch Fehler oberhalb eines bestimmten Schweregrads unterschritten, sind Nacharbeiten wie Bugfixes und Nachtests nötig. Die dafür notwendigen Aufwände können sehr unterschiedlich

ausfallen und lassen sich naturgemäß nicht einplanen, bevor die Fehler nicht gefunden und analysiert wurden.

Wir erleben oft, dass ein pauschaler „Bugfixing-Overhead" vorgeschlagen wird, der jedoch zu einem unsauberen Entwicklungsstil einlädt und das grundsätzliche Problem nicht lösen kann. Nachbesserungsaufwände sind kein kontinuierlicher Overhead! Sie bedeuten einen Stillstand für eine gewisse Zeit: Das Team arbeitet daran, einen vermeintlich bereits vorhandenen Stand zu erreichen. Bugfixing ist somit – streng genommen – eine ungeplante, unproduktive Tätigkeit.

Interessanterweise erleben wir oft, dass Entwicklungsteams häufiges Bugfixing nicht als problematisch empfinden – im Gegenteil: Bugfixing erscheint höchst produktiv, da die Qualität des Produkts dadurch stark verbessert wird. Diese Wahrnehmung stimmt natürlich, allerdings nur, wenn man das fehlerhafte Produkt als Ausgangspunkt nimmt und nicht das eigentlich angenommene fehlerfreie Produkt.

Nacharbeiten aufgrund von Qualitätsmängeln sind in der Regel nicht planbar: Bugs werden entdeckt und müssen zeitnah behoben werden, wenn es der Schweregrad verlangt. Zudem ist es generell wichtig, Fehler zeitnah zu beseitigen, da sonst Fehlermaskierungen drohen (d. h., ein Fehler verhindert die Aufdeckung eines anderen). Es wird durch derartige unplanbare Nacharbeiten zu einem riskanten Unterfangen, „on time, on budget" abzuliefern. Je später Stillstand im Projekt entsteht, desto unrealistischer ist es, ihn durch eine höhere Teamgeschwindigkeit zu kompensieren (**Abb. 1**), und umso höher ist die Gefahr, Deadline und Budget zu überziehen oder den vereinbarten Scope nicht abzuliefern.

Die Teamgeschwindigkeit berechnen wir hierbei auf Basis der Aufwände eines *product increment*, soweit es von der QK zur Abnahme empfohlen wurde (unter der Voraussetzung, dass auch die Regressionstests erfolgreich verlaufen sind). Der Kern unserer *definition of done* lautet also: „QK erfolgreich!". Die Teamgeschwindigkeit basiert daher auf dem durch die QK nachgewiesenen Projektfortschritt. *(Anmerkung.: Wir setzen voraus, dass die Tests der QK in Umfang und Qualität geeignet sind, die Umsetzung der (Mindest-) Anforderungen nachzuweisen.)*

Eine wichtige Konsequenz aus diesem Nachweis ist, dass die Arbeit an den umgesetzten Anforderungen tatsächlich als abgeschlossen angesehen werden darf: Ungeplante Nacharbeiten in diesen Bereichen sind nicht zu erwarten. Die planmäßig zur Verfügung stehenden Ressourcen werden voraussichtlich nicht für Nachbesserungsarbeiten abgezogen werden müssen. Der Nachweis des Projektfortschritts durch die QK ist eine wichtige Voraussetzung (neben Risikominimierung, personeller Kontinuität etc.), um im Projekt Planungssicherheit zu erreichen.

Umgekehrt betrachten wir *product increments*, die von der QK in Teilen abgelehnt wurden, als „offene Baustellen": Zum einen stehen konkrete Nacharbeiten an, zum anderen können Nachtests aufgrund von Fehlermaskierungen weitere Nacharbeiten erfordern. Die Arbeit am *product increment* ist somit nicht abgeschlossen, sondern *work in progress* (*WiP*). Derartige aufgeschobene Nacharbeiten können sich anhäufen und zu einem unkalkulierbaren Risiko werden (**Abb. 2**).

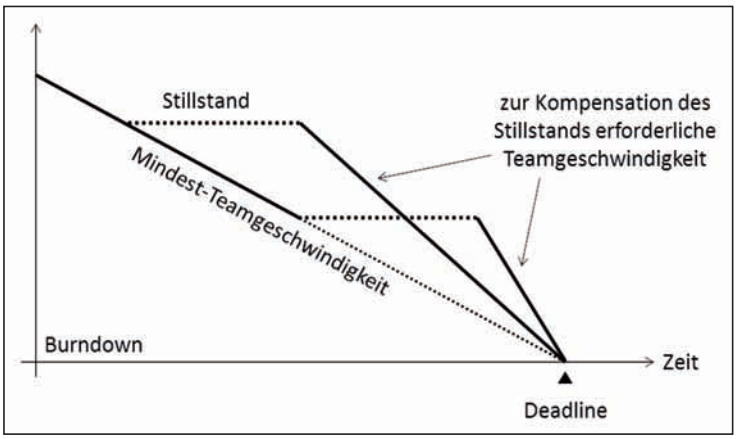


Abb. 1: Später Stillstand im Projekt lässt sich nur durch unrealistische Teamgeschwindigkeit kompensieren

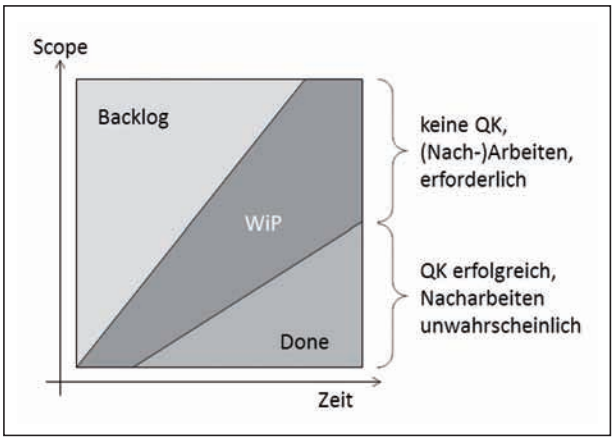


Abb. 2: Risikofaktor „work in progress" (WiP) – vereinfachte Darstellung

Aufwände zur Qualitätssicherung sind dagegen schätzbar und planbar. Selbst QK-Aufwände können geschätzt werden, sofern „die Qualität stimmt", d. h. Nachtests einen vernachlässigbaren Anteil an den QK-Aufwänden haben.

The Good, the Bad, and the Ugly Iteration

Eine der höchsten Prioritäten sollte also sein, das geforderte Qualitätsniveau herzustellen und mit allen Kräften zu erhalten. Statt: „Wir haben keine Zeit für gute Qualität", gilt eher: „Wir haben keine Zeit, uns mit den Folgen schlechter Qualität herumzuschlagen".

Agile Projekte besitzen wirkungsvolle „Selbstheilungsmechanismen", wie z. B. die Retrospektive, die geeignet sind, bis zu einem gewissen Maß Qualitätsmängel wieder auszugleichen. Woran erkennen wir jedoch, ob diese „Selbstheilungsmechanismen" noch ausreichen oder ob tiefgreifende Maßnahmen erforderlich sind, die möglicherweise nur das Management beschließen kann? Wir haben drei Typen von Iterationen identifiziert, „good", „bad" und „ugly", die sich vor

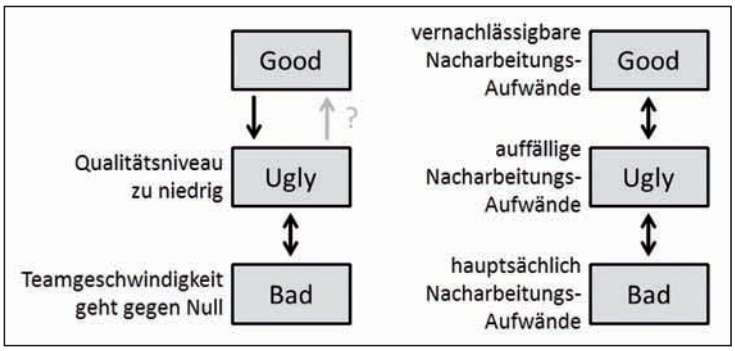


Abb. 3: „Teamgeschwindigkeit" und „Qualitätsniveau" vs. „Anteil der Nacharbeitungsaufwände am Gesamtaufwand einer Iteration"

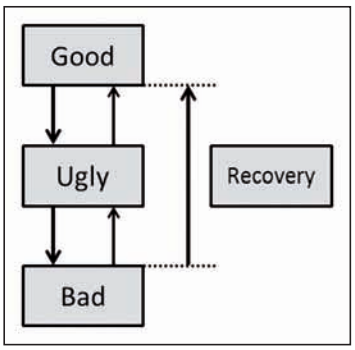


Abb. 4: Die „Recovery Iteration" als Weg zurück in die „Good Iteration"

allem in zwei miteinander korrespondierenden Merkmalen unterscheiden:

- Qualitätsniveau des *product increment*
- Teamgeschwindigkeit

Iterationen, in denen Qualitätsniveau und Teamgeschwindigkeit hoch sind, nennen wir „good". Iterationen, in denen das Qualitätsniveau so gering ist, dass die Teamgeschwindigkeit gegen null geht, nennen wir „bad". Alle Zwischenstufen, in denen häufige Nacharbeiten aufgrund des niedrigen Qualitätsniveaus nur eine geringe Teamgeschwindigkeit zulassen, nennen wir „ugly".

Die Zielsetzung, die diesen Iterationen zugrunde liegt, ist in allen drei Fällen gleich: Mit jeder Iteration soll ein *potentially shippable product increment* abgeliefert werden. Dieses Ziel wird in einer ugly Iteration nur mit Mühe und in einer bad Iteration gar nicht erreicht.

Der Übergang von einer good zu einer ugly Iteration kann nicht am Burndown Chart abgelesen werden: Stillstand, der durch Nacharbeiten entsteht, wird u. U. durch den parallel dazu erreichten Projektfortschritt kaschiert. Hinzu kommt, dass die Teamgeschwindigkeit auch in good Iterations schwankt; de facto funktioniert sie als Indikator nicht. Insofern verstehen wir unter einer hohen Teamgeschwindigkeit in einer good Iteration, dass sie nicht nennenswert durch Nacharbeitungsaufwände verringert wird.

Als Indikator für den Übergang von einer good zu einer ugly Iteration eignet sich aber das Qualitätsniveau gut, das z. B. über Anzahl und Schweregrad der Bugs genau genug ermittelt werden kann (**Abb. 3**).

Bezüglich des Übergangs von einer ugly zu einer bad Iteration verhält es sich genau umgekehrt: Das Qualitätsniveau ist in beiden Fällen niedrig und taugt daher nicht als Indikator. Die Teamgeschwindigkeit geht bei einer bad Iteration jedoch gegen null.

Das Problem mit den beiden Indikatoren Teamgeschwindigkeit und Qualitätsniveau ist jedoch, dass wir darüber nicht feststellen können, ob wir aus einer ugly Iteration wieder in eine good Iteration gelangt sind: Das Qualitätsniveau kann hoch sein, wird aber durch kontinuierlich hohe Nacharbeitungsaufwände teuer erkauft. Damit ist die Teamgeschwindigkeit nicht so hoch, wie sie sein könnte. Wir benötigen also einen weiteren Indikator, um nicht unbemerkt in eine Phase von ugly Iterations hineinzugleiten, die wir fälschlicherweise für good Iterations halten. Dieser Indikator ist der Anteil der Aufwände, die für Nacharbeiten anfallen (Fehler rekonstruieren, lokalisieren, fixen sowie Nachtests), im Vergleich zu den Aufwänden, die für die eigentlichen geplanten Arbeiten anfallen (Entwicklung, Regressionstests, Abnahmetests). Die in agilen Projekten verwendeten Tools zum Tracking der Aufwände erlauben üblicherweise die separate Erfassung der Aufwände. Dies erfordert seitens des Teams jedoch Disziplin bei der Aufwandserfassung.

Bis zu einem gewissen Grad reichen die agilen „Selbstheilungsmechanismen" aus, um den Übergang von einer ugly zu einer good iteration zu schaffen. Die Retrospektive ist z. B. ein wirkungsvolles Mittel, um Hindernisse im Prozess zu identifizieren und Maßnahmen zu beschließen, mit denen diese Hindernisse aus dem Weg geräumt werden. Unsere Beobachtung ist aber, dass die Ursachen für Qualitätsmängel häufig nicht explizit benannt werden können, da sie meist komplex und diffus sind. Besonders deutlich zeigt sich diese Schwierigkeit, wenn das niedrige Qualitätsniveau selbst der Grund für weitere Qualitätsmängel ist. Das ist z. B. dann der Fall, wenn neue Komponenten aus fehlerhaften Komponenten aufgebaut werden oder wenn „dunkle, unkontrollierbare Ecken" im System verhindern, dass ein einwandfreier Entwurf exakt umgesetzt werden kann: Es ist schwierig, wenn nicht gar unmöglich, ein qualitativ mangelhaftes System qualitativ hochwertig weiterzuentwickeln!

In der Retrospektive zeigt sich das beispielsweise im Wunsch des Entwicklungsteams, Teile des Systems (die „dunklen Ecken") durch Refactorings und qualitätssichernde Maßnahmen auf ein akzeptables Qualitätsniveau zu heben. Es ist jedoch wichtig, den Punkt bestimmen zu können, an dem der Aufwand für solche „Aufräumarbeiten" größeren Stils gerechtfertigt ist, da es sich um nicht unmittelbar produktive Aufwände handelt. Wenn wir uns entschließen, in einer Iteration Aufwände vor allem in die Qualitätssteigerung zu investieren, handelt es sich um eine Verschiebung der Zielsetzung in der Iteration: Oberstes Ziel ist nicht mehr, ein weiteres (möglichst großes) *Product Increment* zu schaffen, sondern die Qualität so weit zu steigern, dass die QK des (möglicherweise kleinen) *Product Increment* erfolgreich verläuft. Eine Iteration, die dies zum Ziel hat, nennen wir *Recovery Iteration* (**Abb. 4**). Die Bewertung von Iterationen als „good",

„bad“ oder „ugly“ ist dabei ein sehr hilfreicher Ansatz, um zu bestimmen, wann es sinnvoll ist, eine oder mehrere Recovery Iteration(s) einzuschieben.

„Ugly Iteration“
Im Allgemeinen ist die *ugly Iteration* der vorherrschende Iterationstyp, wenn agil durchgeführte Projekte ihren Coachingbedarf signalisieren.

- Der Anteil von Nacharbeiten an den Gesamtaufwänden ist auffällig und beeinflusst die Planung.
- Stories werden aufgrund der Nacharbeiten oft nur unvollständig umgesetzt. Durch den hohen Anteil an *Work in Progress* am Ende einer Iteration steigt das Risiko weiterer Nacharbeiten.
- Weder Teamgeschwindigkeit noch Qualitätsniveau sind beurteilbar, die Transparenz sinkt.
- Leerlauf entsteht durch Wartezeiten auf Entwicklungs- und QK-Seite. Der Product Owner versucht häufig, diese mit neuen Stories zu füllen, und riskiert den *Work in Progress*-Anteil weiter zu erhöhen.
- Bugs werden hauptsächlich von der QK gefunden, nicht von den Entwicklern.

Die ugly iteration begünstigt den weiteren Qualitätsverfall und erfordert aktives Gegensteuern.

„Good Iteration“
Die good Iteration ist das angestrebte Ideal und sollte gleichzeitig der Normalfall sein. Hier werden die gewünschten Ergebnisse in der geforderten Zeit und Qualität geliefert.

- Stories werden innerhalb der Iteration abgeschlossen und die QK verläuft positiv
- Nacharbeiten finden nur im vernachlässigbaren Umfang statt
- Kein Stillstand durch Wartezeiten, die Tätigkeiten von Entwicklung und QK sind ineinander verzahnt: Die QK testet das jeweils letzte *Product Increment* und bereitet die Tests für das bereits in Entwicklung befindliche nächste Product Increment vor.
- Diese Verzahnung von Entwicklung und QK ist sehr effizient, aber bei einem Absinken der Qualität schwer aufrechtzuerhalten. Dann droht die Gefahr, in *ugly Iterations* abzurutschen.

„Bad Iteration“
Mit der Zeit entwickelt sich die *ugly Iteration* zunehmend in Richtung einer *bad Iteration*, wenn nicht gegengesteuert wird.

- Das Team ist mit Nacharbeiten voll ausgelastet.
- Ein durch die QK nachgewiesener Projektfortschritt ist aufgrund der hohen Nacharbeitungsaufwände und des niedrigen Qualitätsniveaus nicht mehr möglich.
- Der Großteil der Tests schlägt fehl oder ist nicht mehr anwendbar. Im Extremfall kann die QK keine Tests mehr durchführen, da der Entwicklungsstand nicht mehr zu den Tests passt.
- Anhäufen von technischer Schuld auf Entwicklungs- und Testschuld auf QK-Seite
- Agile Selbstheilungsmechanismen können den Projektzustand nicht mehr verändern. Aus eigener Kraft kann das Projekt nicht wieder in die Spur kommen.

„Recovery Iteration“
Ein bewährter Weg aus ugly und bad Iterations heraus besteht darin, den eingefahrenen Ablauf zu unterbrechen und den Prozess über eine oder mehrere Recovery Iterations in die Spur zurückzubringen. Normalerweise erfordern Re-

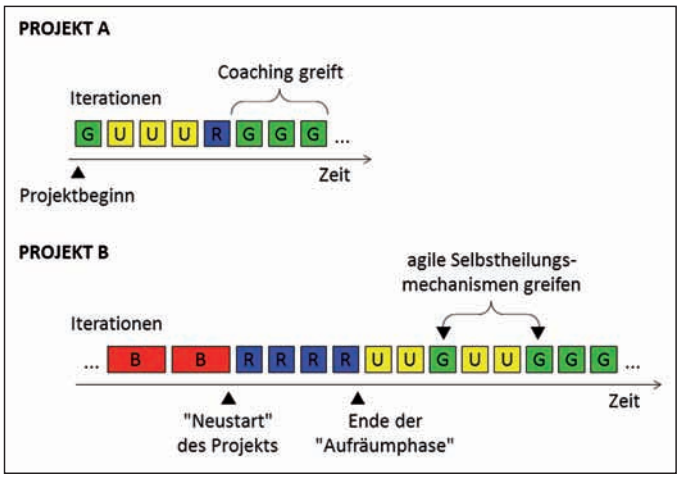


Abb. 5: Projektverlauf der Beispiele

covery Iterations die Einwilligung von außen (Kunde/Management).

- Das primäre Iterationsziel ist nicht mehr die Schaffung neuer Funktionalität, sondern die Anhebung des Qualitätsniveaus.
- Entwicklung und QK laufen ohne Verzahnung und strikt sequenziell ab:
  - Entwicklung inkl. Tests auf Entwicklerseite
  - Test durch die QK
  - Im Fehlerfall: Erfassen des Bugs durch die QK
  - Bugfixing inkl. Nachtests auf Entwicklerseite
  - Nachtests durch die QK
- Beide Seiten müssen sich auf die vorangegangenen Ergebnisse verlassen können, um ihrerseits eine hohe Qualität ihrer Ergebnisse garantieren zu können.
- Die Durchführung von Aufräumarbeiten vermeidet Leerlauf, z. B. Refactoring von Code und Anheben der Unit-Testabdeckung auf Entwicklerseite sowie Reviews und Überarbeiten des Testdesigns auf QK-Seite.
- Im Extremfall muss das System ohne Beteiligung der QK so weit aufgeräumt werden, dass es testbar ist.

Die Recovery Iteration stoppt den Teufelskreis der ugly Iteration unmittelbar. Die Qualität steigt in den bearbeiteten Bereichen kontinuierlich und (sofern die QK testen kann) nachweislich an. Das Projekt nimmt selbst aus einer bad Iteration heraus wieder Fahrt auf. Die agilen Selbstheilungsmechanismen können jetzt greifen, da Ziele und Hindernisse wieder klarer erkennbar sind.

Als Indikator zur Einstufung der Iterationen in good, bad und ugly verwenden wir – wie oben beschrieben – den Anteil der Nachbesserungsaufwände an den Gesamtaufwänden einer Iteration. Denselben Indikator verwenden wir, um abzuwägen, wann wir die (zeitaufwändigen) Recovery Iterations wieder absetzen.

Beispiele aus der Praxis

In der Praxis setzen wir die Einstufung von Iterationen in „good“, „bad“ und „ugly“ seit knapp zwei Jahren in teils sehr unterschiedlichen Projekten ein (auch wenn wir oft andere Namen dafür verwendet haben, z. B. „failed sign-off/no progress“ statt „ugly/bad“ oder „Aufräum-Iteration“ statt Recovery Iteration).

Das Feedback aus den verschiedenen Teams macht deutlich, dass die unmissverständliche Einstufung der Iterationen in good, bad und ugly eine offenbar hilfreiche Orientierung bietet, und das sowohl bei der Einschätzung des Projektzustands als auch der Faktoren, die zu diesem Zustand geführt haben.

Die Recovery Iterations haben wir mehrfach gezielt einsetzen können, um Projekte wieder in die Spur zu bringen, die teils sehr große Schwierigkeiten mit dem agilen Vorgehen hatten.

Darüber hinaus gewährt die Einstufung der Iterationen einen einfachen und dennoch informativen Überblick über den iterativen Projektverlauf – z. B. für das Reporting gegenüber dem Management. Dies ist in den folgenden beiden, sehr unterschiedlichen Beispielen zu sehen, mit denen wir versuchen, einen knappen Eindruck von der Bandbreite von Projekten zu geben, in denen unser Ansatz anwendbar ist.

Projekt A

- Projektziel: Schaffung einer technischen Basis zur Synchronisation von Informationen zwischen verschiedenen Projekten
- Laufzeit: ca. 5 Monate
- Status: kurz vor Abschluss
- Teamgröße: 2 – 3 Entwickler, 1 QKler
- Vorgehensmodell: iterativ-inkrementell, angelehnt an Scrum; zweiwöchige Iterationen
- Beginn unseres Coachings: mit Projektbeginn

Das Projektteam besaß nur wenig Erfahrung mit agilen Vorgehensweisen und hatte anfangs Probleme, agile Techniken und das Projekt-Tracking konsequent anzuwenden. Unklar formulierte Anforderungen erforderten zahlreiche Nachfragen beim Anforderungsgeber und erschwerten die Situation aus Sicht des Teams noch weiter.

Nach einem hoffnungsvollen Start folgte eine Reihe von ugly Iterations. Eine Trendumkehr mithilfe agiler Selbstheilungsmechanismen schien uns angesichts des unerfahrenen Teams nicht in Sicht, sodass wir uns entschlossen, eine Recovery Iteration einzuschieben, um den Negativtrend zu stoppen. Eine einzelne Recovery Iteration reichte aus, um dem Team wieder Übersicht über den Prozess zu geben, sodass in der Folge mit begleitendem Coaching wieder good Iterations gelangen (**Abb. 5**, Projekt A).

Die Teamgeschwindigkeit ist im Vergleich zu erfahrenen Teams auch in den good iterations etwas niedriger. Dennoch erreicht auch das agil unerfahrene und auf das Coaching angewiesene Team einen angemessenen und kontinuierlichen Projektfortschritt mit stimmiger Qualität.

Zurzeit befindet sich das Projekt in der Endphase. Ein erfolgreicher Abschluss des Projekts erscheint aus momentaner Sicht sehr wahrscheinlich.

Projekt B

- Projektziel: Ablösung einer klassischen Webanwendung durch eine moderne Rich Internet Application – für einen sehr großen Kundenkreis
- Laufzeit: mind. 3 Jahre; Neustart des Projekts nach ca. einem Jahr
- Status: in der Umsetzung
- Teamgröße: ca. 10 Entwickler (vor Neustart ca. 30), ca. 5 QKler
- Vorgehensmodell: Scrum, Begleitung durch Scrum-Coaches; anfangs vierwöchige Iterationen, nach Beginn des Coachings zweiwöchig
- Beginn unseres Coachings: ca. ein Jahr nach Projektbeginn, zum Neustart intensiviert

Das Projektteam besaß zu Beginn unseres Coachings bereits einige Erfahrung mit der agilen Vorgehensweise. Die agilen Techniken waren allen bekannt und wurden teils minutiös angewandt. Retrospektiven wurden regelmäßig durchgeführt. Die QK besaß eine umfangreiche Datenbank mit detaillierten Tests. Bugs wurden von der QK dokumentiert, bewertet und verfolgt.

So weit, so ideal – doch das Qualitätsniveau der realisierten Anwendung passte nicht zu diesem Bild: Prototypische, halbfertige und fehlerhafte Teile durchmischten sich untrennbar mit angeblich funktionierenden Teilen. Die QK konnte zahlreiche Tests gar nicht anwenden oder stieß bei deren Durchführung auf Fehler. Einige auffällige Bugs existierten bereits seit mehreren Iterationen und wurden nicht behoben, da die Umsetzung weiterer Stories eine höhere Priorität hatte – ein Versuch, den Projektstillstand durch „Gasgeben" abzuwenden.

De facto war jedoch schon seit längerem kein Projektfortschritt mehr durch die QK nachweisbar und wurde schließlich auch vom Auftraggeber nicht mehr erkannt. Diese Situation hat schließlich das Management dazu bewogen, einen Neustart des Projekts durchzuführen (**Abb. 5**, Projekt B).

Die Iterationen vor dem Neustart ließen sich auch im Nachhinein klar als *bad Iterations* ausweisen. Mit dem Neustart bekamen wir die Chance, *Recovery Iterations* einzuführen, um darüber die Kontrolle über den Prozess zurückzuerlangen. In diesen *Recovery Iterations* hat das Entwicklungsteam das System massiv überarbeitet, Halbfertiges und Prototypisches ersatzlos ausgebaut und konsequent Fehler behoben – so lange, bis die QK schließlich nach einer Aufräumphase von fast zwei Monaten zum ersten Mal einen vollständigen, erfolgreichen Regressionstest des gesamten Systems durchführen konnte.

Neben der deutlichen Qualitätsverbesserung des Systems waren die Recovery Iterations für das Entwicklungsteam sehr wichtig, um schließlich ein hohes Qualitätsverständnis in der Praxis etablieren zu können, das den technischen Fähigkeiten des Teams entsprach. Die QK hatten wir während der Recovery Iterations größtenteils aussetzen müssen, da keine aussagekräftigen Tests durchführbar waren. Die QK nutzte diese Zeit, um ihre Tests zu sichten und an den Scope des aufgeräumten Systems anzupassen. Mit den ersten erfolgreich durchgeführten Regressionstestfällen konnte die QK ihre reguläre Arbeit schließlich wieder aufnehmen. Auf Basis des aufgeräumten Systems griffen auch wieder agile Selbstheilungsmechanismen, deren Effekt zuvor verpufft war.

Inzwischen rutscht das Team nur noch selten in ugly Iterations ab und findet in solchen Fällen aus eigener Kraft wieder heraus. Die Teamgeschwindigkeit hat mittlerweile ein hohes Niveau erreicht – Entwicklung und QK arbeiten parallel. Nacharbeiten und Nachtests spielen keine nennenswerte Rolle mehr, die Planungssicherheit ist dementsprechend hoch.

Fazit

Eine hohe Qualität der implementierten Software ist für den Projekterfolg notwendig, sie kann aber – wie wir in vielen Projekten erleben – nicht immer erreicht und gehalten werden.

Die Gründe für den Qualitätsverlust sind meist komplex und schwer zu benennen. Agile Selbstheilungsmechanismen wie die Retrospektive sind zwar hilfreich und notwendig, reichen aber in vielen Fällen nicht aus oder greifen unterhalb eines gewissen Qualitätsniveaus gar nicht mehr.

Über die Indikatoren Teamgeschwindigkeit, Qualitätsniveau und den Anteil der Nacharbeitungsaufwände am Gesamtaufwand können wir drei Typen von Iterationen identifizieren: good, bad und ugly Iterations. Diese helfen uns, Transparenz über den Projektzustand herzustellen und zu entscheiden, ab wann und bis wann umfassende qualitätsverbessernde Aufwände gerechtfertigt sind.

Zur Qualitätsverbesserung haben sich Recovery Iterations bewährt, in denen wir die Zielsetzung vom Umsetzen neuer Anforderungen auf die Anhebung des Qualitätsniveaus verschieben. Dies ist ein sehr wirkungsvoller Mechanismus, über den ein „aus der Spur gekommener agiler Prozess wieder korrgiert werden kann.

Jörn Koch ist Senior Softwarearchitekt bei der Workplace Solutions GmbH. Er leitet und coacht seit 2001 agile Projekte und besitzt langjährige Erfahrung in der Einführung und im "Tuning" agiler Prozesse. Kontakt: Joern.Koch@workplace-solutions.de.

Sebastian Middeke ist Senior Consultant bei der Cognizant Solutions GmbH. Er berät seit 2005 agile Projekte zu Themen des Test- und Anforderungsmanagements und coacht die Einführung agiler Prozesse. Kontakt: Sebastian.Middeke@cognizant.de.

Links & Literatur

[1] K. Beck: „Extreme Programming Explained – Embrace Change", Addison-Wesley, 1999

[2] H. Breitling: „XP Revisited", in: Java Magazin 3/2011, S. 50 – 56

[3] ISO/IEC 9126

[4] ISTQB/GTB Standard Glossar der Testbegriffe

JAVA³

Jetzt abonnieren!
www.javamagazin.de

Jetzt 3 TOP-VORTEILE sichern!

1

- Alle Printausgaben frei Haus erhalten
- Intellibook-ID kostenlos anfordern (www.intellibook.de)

2 Abo-Nr. (aus Rechnung oder Auftragsbestätigung) eingeben

3 Zugriff auf das komplette PDF-Archiv mit der Intellibook-ID

S&S Media Group

www.javamagazin.de