Florian Georg
(E-Mail: florian.georg@ch.ibm.com)
ist Software Solution Architect bei IBM Schweiz und berät Kunden zum Einsatz von modernen Softwareentwicklungsprozessen, Praktiken und Werkzeugen - quer durch alle Disziplinen. Er ist ein starker Unterstützer softwaregetriebener Produktinnovation, des Konzepts der „Lean User Experience“ und der Möglichkeiten der Cloud. Florian Georg ist zudem Sprecher auf internationalen Softwarekonferenzen, Gastdozent an Hochschulen und Mitglied des IBM Technical Expert Councils (TEC) für Zentraleuropa.

Dr. Hans-Joachim Pross
(hajo.pross@de.ibm.com)
ist Client Technical Architect bei der IBM in Köln. Nach seinem Studium der Physik arbeitete er zunächst beim Bank-Verlag als DV-Koordinator, war dann lange Jahre als Consultant bei Softlab tätig, ehe er 2001 zu IBM Rational wechselte. Dort ist er seither vertriebsunterstützend für Rational-Produkte tätig. Seine aktuellen Schwerpunkte bilden die Beratung für Lösungen im Bereich Änderungs- und Konfigurations-Management, Asset-Management, DevOps und natürlich Cloud Computing. Seit 2012 ist er Scrum.Org Certified Professional Scrum Master I. Darüber hinaus ist er Community of Practice Lead für Change und Configuration Management.

Dr. Henning Sternkicker
(E-Mail: henning.sternkicker@de.ibm.com)
studierte Chemie an der RWTH Aachen. Nach dem Abschluss des Studiums und der Promotion begann er als technischer Berater bei Rational Software Deutschland. Seit der Übernahme von Rational durch IBM arbeitet er als Client Technical Professional vertriebsunterstützend für IBM Deutschland. Der Tätigkeitsschwerpunkt im Bereich der IBM-Entwicklungswerkzeuge geht dabei vom Anforderungsmanagement, über Kollaborationswerkzeuge in der Softwareentwicklung basierend auf der Jazz-Technologie bis hin zu dem Themengebiet DevOps. Seit 2012 ist er Scrum.Org Certified Professional Scrum Master I.

Agility zu Ende gedacht – von der Entwicklung bis zum Betrieb

Befragt man heutzutage Entwicklungsteams, ob sie agile Methoden einsetzen, wird der überwiegende Teil antworten, dass sie zumindest nach einigen der agilen Prinzipien arbeiten. Dieses Bild bestätigen auch zahlreiche Veröffentlichungen (z. B. [Kom14] und [SwQ14]), die sich mit der Verbreitung von agilen Methoden beschäftigen. Von den unterschiedlichen agilen Vorgehensmodellen ist Scrum mit großem Abstand das verbreitetste. Außerhalb der Softwareentwicklungsabteilungen sinkt der Nutzungsgrad der agilen Vorgehensweisen jedoch erheblich auf etwa ein Viertel.

Eine Ursache für die zunehmende Verbreitung agiler Methoden liegt sicherlich in der gestiegenen Notwendigkeit für Unternehmen, Produkte schneller als bisher (weiter-) entwickeln zu müssen, um das Fenster für Innovationen und das Ergreifen von Marktmöglichkeiten nicht zu verpassen.

Darüber hinaus sind agile Methoden ein modernes Mittel zur Risikominimierung, da schon im agilen Manifest *ausführbare Ergebnisse* und eine *stetige Abstimmung mit dem Kunden* als wichtige Werte beschrieben werden. Schnellere Reaktionen auf Kunden- bzw. Markt-Bedürfnisse sowie kürzere Projektlaufzeiten werden hier durch viele und kurze Iterationen verwirklicht.

Der größte Unterschied agiler Methoden im Vergleich zum traditionellen Projektmanagement besteht darin, dass hier eben gerade nicht auf die einmalige Liefe-

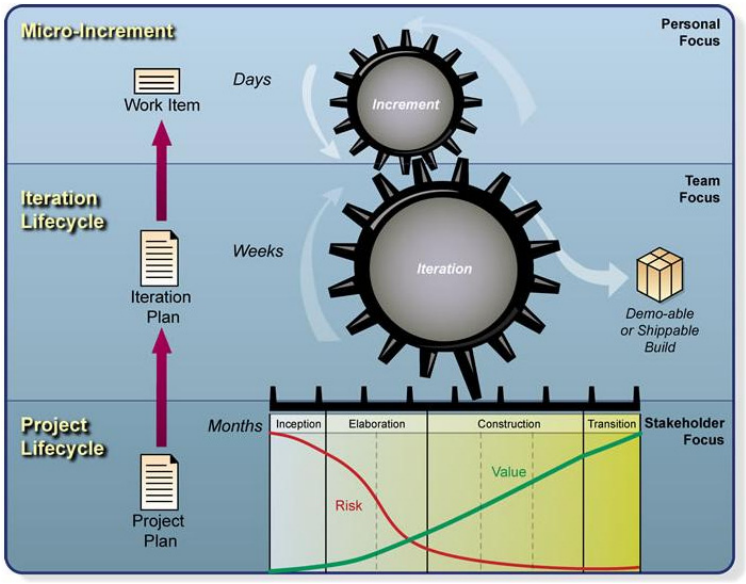

Abb. 1: Agilen Prozessen – wie hier exemplarisch am openUP-Prozess dargestellt [oUP10] – ist zu eigen, dass sie mit der Erstellung eines lieferbaren Ergebnisses enden. Was mit diesem Ergebnis anschließend passiert, ist außerhalb des Fokus

rung eines Ergebnisses abgezielt wird. Die anfängliche, erhebliche Unschärfe über die echten Kundenanforderungen wird nicht mitigiert, sondern aktiv über den gesamten Lebenszyklus angenommen („Embrace Change“) und sukzessive und inkrementell verringert. Deshalb wird ein kontinuierlicher Fluss neuer Versionen und eine kontinuierliche Erweiterung und Verbesserung des Produktes in zahlreichen Teillieferungen angestrebt. Je nach technischen und organisatorischen Rahmenbedingungen sind sogar Iterationszyklen von ein oder zwei Wochen nicht ungewöhnlich.

Aus dem Prinzip, stets ein installier- und testbares System zu haben, ist der Gedanke der Continuous Integration geboren. Statt zu festgelegten Zeiten zu integrieren, werden gelieferte Teilkomponenten kontinuierlich, oft mehrmals täglich, gebaut, integriert und getestet. Abbildung 1 ist zu entnehmen, dass agile Prozesse mit einem „Shippable Build“ oder einem „Working Increment“ enden. Dieses Inkrement ist nur von wirklichem Mehrwert, wenn das Entwicklungsteam direkt mit dem Nutzer interagieren kann.

Neue Herausforderungen oder gar Probleme für ein agiles Team entstehen, sobald die Komplexität durch eine heterogene IT-Landschaft, organisatorisch und räumlich getrennte Abteilungen, rechtliche Anforderungen und Standards oder Ähnliches ansteigt. In diesem Umfeld wird es zunehmend schwieriger, mit der Entwicklungsgeschwindigkeit Schritt halten zu können und die Ergebnisse zeitnah auf

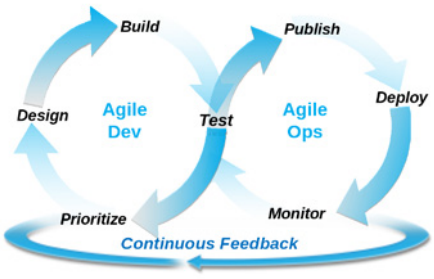

Abb. 2: DevOps: Das Zusammenspiel von agiler Entwicklung mit kontinuierlicher Integration und kontinuierlicher Lieferung ermöglicht kontinuierliches Feedback. Aus der Durchgängigkeit und Kontinuität erwachsen die Vorteile schneller entwickelter und qualitativ hochwertigerer Produkte

Testumgebungen oder gar in die Produktion zu überführen. Für diesen Fall liefern agile Praktiken in der Regel keine Hinweise darauf, wie das von der Entwicklung erstellte Inkrement beim Endkunden ankommen soll.

Dies ist umso erstaunlicher, da es doch im Widerspruch zu den bereits erwähnten agilen Werten steht, welche so formuliert sind, dass sie auch jenseits der reinen Entwicklung angewandt werden können. Üblicherweise ist der direkte Nachbar des Entwicklungsteams, das Operationsteam, eher nach „klassischen“ Abläufen wie etwa ITIL Service Management orientiert, sodass an mancher Stelle inzwischen gern davon gesprochen wird, dass das agile Entwicklungsteam auf die „Operations Wall“ trifft.

Dies ist aus Sicht einer ganzheitlichen Prozessoptimierung im Sinne einer „Lean IT“ [Wikia14] nicht optimal. Während agile, teamorientierte Praktiken, wie etwa „Story-Driven Development“ und „Sprint Reviews“, die Brücke zu den Fachabteilungen gut zu schlagen vermögen, fehlten lange Zeit tragfähige Konzepte für die Zusammenarbeit mit IT Operations. Jüngere Veröffentlichungen wie Scaled Agile Framework (SAFe) [Lef14] oder Disciplined Agile Delivery (DAD) [Amb12] versuchen deshalb, das agile Team zu erweitern und auch andere Gruppen, wie Operations, mit einzubeziehen, um so den Kontext der Agilität auf die gesamte IT-Organisation des Unternehmens zu erweitern.

Dazu müssen bestimmte Praktiken, wie der Gedanke der Continuous Integration, erweitert werden und in eine Continuous Delivery Pipeline fortgeführt werden. Ziel hierbei ist es, von individuellen, teamorientierten Praktiken zu einem ganzheitlichen Prozess zu gelangen, ohne dabei die agilen Aspekte zu verlieren. Zu diesem Themenkomplex gibt es einige lesenswerte Publikationen, wie das bereits erwähnte Buch von Scott Ambler. Im Kontext einer unternehmensweiten IT besteht unbestreitbar die Notwendigkeit der Abstimmung der einzelnen Aktivitäten über den gesamten Lebenszyklus.

Continuous Integration beantwortet durch beständiges Prüfen des Codes auf Integrierbarkeit die Frage, ob das *Produkt richtig gebaut* werden kann. Die wichtigere Frage, ob das *richtige Produkt gebaut* wird, erfordert häufiges Feedback der Nutzer des produktiven Systems. Um aber die Rückmeldung der Nutzer kontinuierlich abholen zu können, ist eine weitergehende Automation, also die Weiterentwicklung der Continuous Integration hin zu Continuous Delivery notwendig. Andernfalls wird die Agilität des Entwicklungsteams das Operationsteam nicht erreichen: Die Vorteile der beschleunigten Entwicklung sind damit für den Kunden nicht sicht- bzw. nutzbar. Die Behebung von Fehlern in der von Kunden verwendeten Version der Produkte dauert zu lange, sodass die Qualitätsanmutung des Produktes leidet.

„Software that isn‘t shipped doesn‘t exist“

Auf gar keinen Fall darf es also zum Ausbremsen der agil arbeitenden Entwicklungsabteilung kommen. Es muss vermieden werden, alte Prozessmuster zu adaptieren und den sogenannten „Water-Scrum-Fall“ [Wes11] zu etablieren. Vielmehr gilt es, den Schwung der Agilität in den Kontext der gesamten IT-Organisation zu übertragen. Um dies zu ermöglichen, ist der Ansatz einer Continuous Delivery Pipeline ein wichtiger Baustein.

Die Continuous Delivery Pipeline hilft, die „Operations Wall“ zu durchbrechen

Aus einer unternehmerischen Perspektive haben alle beteiligten Organisationseinheiten dieselben Ziele: Den Erfolg des Unternehmens durch innovative, stabile und sichere Anwendungen zu gewährleisten. Allerdings resultiert aus der organisatorischen Trennung von Entwicklung und Betrieb oft eine voneinander abweichende Interessenlage, mit der das Ziel erreicht werden soll: Agile Entwickler wollen ihre Änderungen schnell und unbürokratisch in das lauffähige System integrieren, quasi „Push Button Deployments“.

Für den Betrieb ist dies jedoch ein Albtraum, da dieser nicht an der Fähigkeit gemessen wird, möglichst viele Änderungen in kurzer Zeit zu liefern, sondern eher im Gegenteil daran, die Systeme möglichst stabil zu halten oder Service Level Agreements (SLAs) zu erfüllen. Häufige und unklare Änderungen stellen hier ein Risiko hinsichtlich Stabilität und der Einhaltung von Richtlinien (Compliance) dar und werden deshalb über entsprechend komplexe, häufig manuelle Prozesse abgehandelt.

Die Botschaft des DevOps (siehe Abbildung 2), dass häufige Lieferungen durch eine weitestgehend automatisierte Pipeline

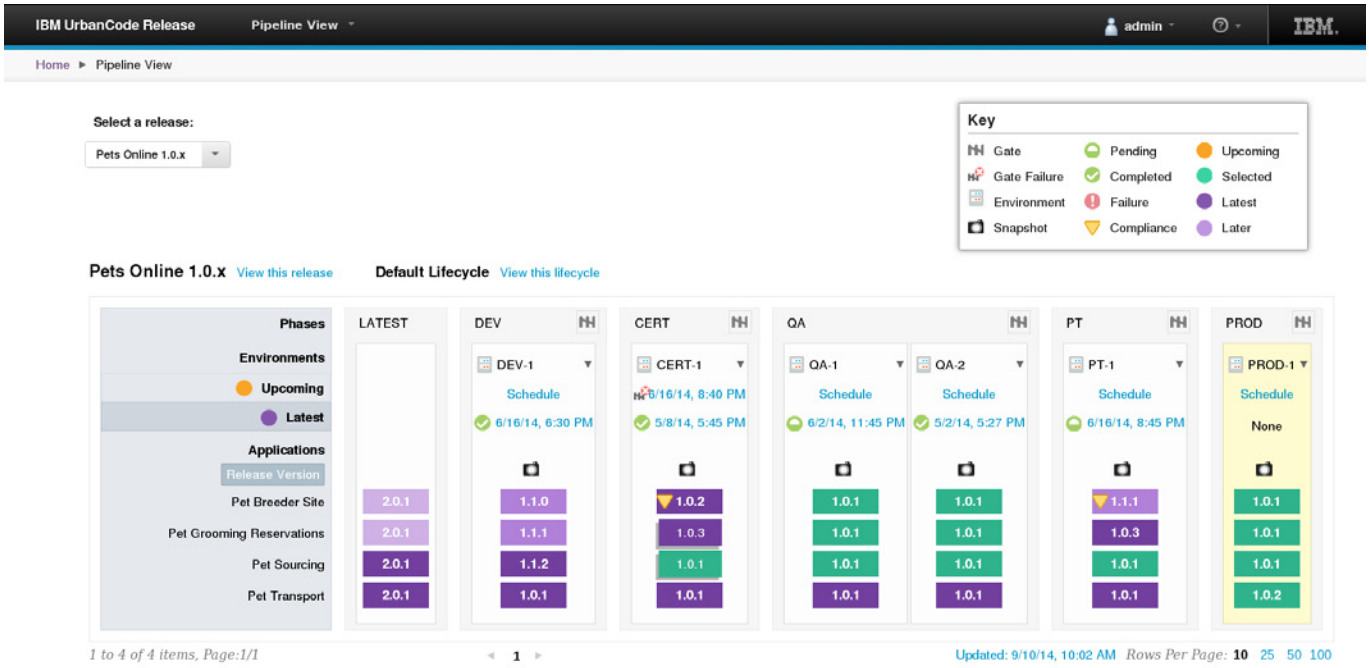

Abb. 3: Die Lösung IBM Urbancode Release erlaubt die Automation und Kontrolle über das Deployment des gesamten Releases. Jederzeit ist der aktuelle Status der Delivery Pipeline sichtbar und kann von einer zentralen Stelle aus gesteuert werden

Risiken vermindern und verbesserte Stabilität des Betriebes hervorbringen, scheint somit auf den ersten Blick paradox. Aber eine möglichst durchgängige Automatisierung führt durch weniger manuelle Schritte zu weniger Fehlern, zu einem reproduzierbaren Verhalten und sogar zu einer schnelleren Interventionsmöglichkeit im Problemfall.

Erfahrungsgemäß ist also anfänglich eine gewisse Skepsis des Betriebs gegenüber dieser Pipeline zu überwinden, da vordergründig Kontrolle verloren geht bzw. diese durch die Werkzeuge mittels Automation scheinbar übernommen wird. Andererseits ist der Betrieb langfristig auch der größte Nutznießer einer solchen Lösung; befreit sie ihn doch von aufwendigen, fehleranfälligen und manuellen Prozessen – und damit von unzufriedenen Entwicklern und Endkunden.

Durch den hohen Automatisierungsgrad werden Betriebsmitarbeiter von „langweiligen“ Arbeiten (z. B. das Abarbeiten von Checklisten) befreit, sodass nicht zuletzt mehr Zeit für Wert-schöpfende Tätigkeiten gewonnen werden kann.

Unter dem Stichwort Application Release Automation (ARA) treten eine Vielzahl von Werkzeugen an, Unternehmen bei dem Aufbau einer Continuous Delivery Pipeline zu unterstützen. Ein Beispiel dafür ist die IBM Urbancode Lösung mit den integrierten Bestandteilen IBM Urbancode Release™ (siehe Abbildung 3) für das Release Management und IBM Urbancode Deploy™ für die eigentliche Automation des Deployments. In IBM Urbancode Deploy werden die Ergebnisse der Entwicklung zu Komponenten und Applikationen zusammengefasst und die Prozesse festgelegt, mit denen diese Applikationen auf unterschiedliche Umgebungen ausgeliefert werden. Durch einfache grafische Prozessdesigner und eine Vielzahl vordefinierter Schritte aus einem breiten Spektrum an bereitgestellten Integrationen in bestehende Systeme gelingt es schnell, die Deployment-Verfahren zu automatisieren.

Die IBM Urbancode Lösung ist plattformübergreifend (Windows, Linux/Unix und Mainframe) und bietet die Möglichkeit, vom Ausliefern einzelner Dateien über die komplette Konfiguration von Applikationsservern bis hin zur Provisionierung von Cloud-Systemen sämtliche Aspekte der Produktionsübergabe bei dem Aufbau einer Continuous Delivery Pipeline zu adressieren. Durch den Einsatz von IBM Urbancode Release kann die Sicherstellung der kontinuierlichen Lieferung und auch die Automation der Release-Verfahren weiter perfektioniert werden. IBM Urbancode Release bündelt die Applikationen zu Releases und erlaubt so, die gegebenen automatisierten Deployment-Verfahren zu automatisierten Release Deployments zusammenzufassen. Durch den Einsatz einer solchen Lösung gelingt es also, durch weitreichende Automation die Kontinuität der agilen Vorgehensweise auf eine komplette Continuous Delivery Pipeline auszubauen.

Agile Projekte profitieren vom Feedback und von den fortdauernden Verbesserungen auf Basis der Interaktion mit den Kunden. Werden nun auch andere Bereiche, wie Operations, nach agilen Vorgehensweisen ausgerichtet, werden die Inkremente schneller veröffentlicht und so auch für den Kunden sichtbar, wodurch diesen die Möglichkeit zu häufigerem Feedback gegeben wird. Das nun häufigere und dadurch auch umfangreichere Feedback vom Kunden und aus dem Betrieb wird zeitnah an die Entwicklungsteams herangetragen.

In den meisten agilen Rahmenwerken existiert keine eindeutige Empfehlung, wie von außen an das Entwicklungsteam herangetragene Fehler z. B. durch CritSits, Hot Fixes oder proaktives Fehlermanagement gehandhabt werden sollen. Die übliche Herangehensweise der agilen Teams ist es, die Fehler in die normale Planung mit aufzunehmen und zu priorisieren, um so frühzeitig diesen, die eigentliche Arbeit behindernden Unrat („Waste“) aus der Welt zu schaffen. Durch das umfangreichere Feedback aus dem erweiterten Kreislauf kann es dann aber zu der Situa-

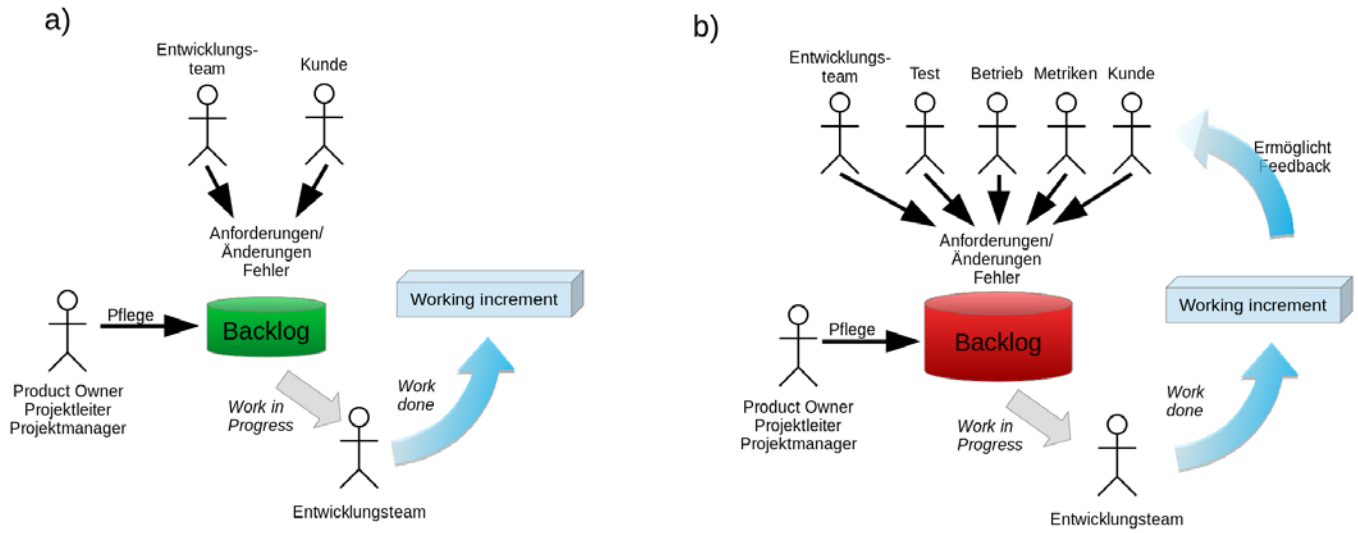

Abb. 4: Im Normalfall interagiert ein agil arbeitendes Entwicklungsteam direkt mit dem Kunden und liefert passende Inkremente. Im unternehmensweiten Kontext besteht die Gefahr, durch umfangreiches Feedback den Fokus auf den Kunden zu verlieren und die Menge an Wünschen nicht mehr beherrschen zu können

tion kommen, dass das Team kaum noch in der Lage ist, Neues zu implementieren, da es nur noch auf die Rückmeldungen von außen reagiert.

Agilität löst keine Probleme, zeigt diese aber früher auf

Mit dieser Aussage im Hinterkopf muss also ein Weg gefunden werden, die Ströme an Feedback, Entwicklungsartefakten und Produkten, die in den immer umfangreicher interagierenden, agil arbeitenden Teams fließen, richtig zu kanalisieren. Hier werden oft Konzepte aus dem „Lean Manufacturing“ (wie Kanban) auf die Softwareentwicklung übertragen, etwa Durchsatzerhöhung von Änderungen durch Limitierung der parallel bearbeiteten Aufgaben („Limit Work in Progress“), konsequente Visualisierung und Überwachung des Arbeitsablaufs [Wikib14]. Ein wichtiger Faktor dabei ist, eine Werkzeug-Unterstützung zu bekommen, die ein hohes Maß an Automation ermöglicht, um auch die Mitarbeiter aus der Entwicklung von manuellen, fehleranfälligen Tätigkeiten zu befreien und auch ihnen Freiräume für wertschöpfende Tätigkeiten zu erhalten.

Die Geister, die ich rief ...

Das Feedback mit dem Endanwender, das „Schärfen“ der Anforderungen und das Reagieren auf Veränderungen sind wesentlicher Bestandteil agiler Vorgehensweisen. Aus diesem Grund sind agile Methoden per se gut geeignet, Fehler früh zu erkennen und die Wünsche und Anregungen der Endanwender effizient in den Prozess zurückzuführen. Hierbei stellt sich aber die Frage, wie dieses Feedback möglichst direkt an die Entwicklung zurückfließen kann. Aus der Erfahrung der Autoren heraus haben die wenigsten Unternehmen eine befriedigende Antwort auf diese Frage. Dies ist umso bemerkenswerter, da doch mittlerweile ein Großteil dieser Apps und Applikationen von agil arbeitenden Teams entwickelt wird, die somit genau von dieser Rückmeldung abhängig sind.

Die oftmals naheliegende Antwort „mit dem bestehenden Fehlerbehandlungs-Prozess“ löst das eigentliche Problem aus unserer Ansicht nicht zufriedenstellend. Da Feedback in immer kürzeren Abständen und von immer mehr Quellen kommt (siehe Abbildung 4), ist es schlicht und einfach notwendig, dieses zu filtern, zu kategorisieren und zu kanalisieren. Andernfalls wäre ein Entwicklungsteam nicht mehr in der Lage, die wertvolle Information effektiv zu nutzen. Großer Zeitverlust durch aufwendige Backlog-Pflege und mühsame Priorisierungs-Diskussionen wäre die Folge.

Es bedarf also mehr als nur eines Fehlerbehandlungs-Systems, um zusätzlich zur Organisation der eigenen Arbeit (via Backlog) die Einbindung in unternehmensweite Strukturen und die Zusammenarbeit mit den anderen Abteilungen zu unterstützen. In den Konzepten, die sich mit der Skalierung agiler Methoden auf das gesamte Unternehmen beschäftigen, werden dazu Ideen wie „Work Item Stack“ oder „Work Item Pool“ als Ergänzung zum agilen Backlog eingeführt [Amb12].

Letztendlich ist also das „geordnete“ Einsammeln des Feedbacks aus den unterschiedlichen Quellen ein wichtiger Bestandteil unternehmensweiter Agilität. Insofern gehört zu einer Continuous Delivery Pipeline immer auch eine integrierte Werkzeugkette, um somit zu einem Continuous Feedback zu gelangen; letztendlich also eine Basis an Werkzeugen, mit denen alle Bereiche die hohe Taktzahl in agil bearbeiteten Unternehmensprojekten bewältigen können.

Hauptaugenmerk beim Aufbau einer integrierten Lösung sollte sein, die Anzahl der benötigten Komponenten in dieser Werkzeugkette auf ein Mindestmaß zu beschränken und den Werkzeugeinsatz eher unter dem Aspekt der Integration und weniger unter dem Aspekt der Spezialisierung zu bewerten. Ein gutes Beispiel ist hier das Fehler- und Änderungsmanagement. Für jede Disziplin, sei es Entwicklung, Testen oder Betrieb, finden sich spezielle Anwendungen zur Aufnahme von, allgemein gesprochen, Änderungswünschen. Aus der Erfahrung der Autoren heraus findet man auch umso mehr dieser Spezialwerkzeuge, je größer das jeweilige Unternehmen ist. Das sind aber die denkbar schlechtesten Voraussetzungen, um Agile Continuous Delivery zu gewährleisten. Besser ist hier, einheitliche Werkzeuge zur Zusammenarbeit über alle Disziplinen

hinweg zu verwenden. Jedes Teammitglied, sei es nun Anforderungsmanager, Entwickler, Betrieb oder Nutzer, bekommt eine rollenspezifische Sicht auf den gemeinsamen Datenbestand der Änderungswünsche und Fehler.

Mit der auf der Jazz-Architektur basierenden IBM Collaborative Lifecycle Management Plattform™ (CLM), und hier speziell mit IBM Rational Team Concert™ als Lösung zur Verfolgung und Bearbeitung von Änderungswünschen, lässt sich eine solche Anlaufstelle für alle Projektbeteiligten schaffen. Über die Funktionalität der CLM-Plattform und im Besonderen von IBM Rational Team Concert wurde bereits an anderer Stelle berichtet [Pro12].

Fazit

Die Unternehmensziele für Develoment und Operations sind, wie auch Ziele der agilen Methoden und der DevOps-Bewegung, identisch: Es geht allen darum, die Entwicklungszeit insgesamt zu verkürzen, schneller und flexibler auf veränderte Anforderungen zu reagieren, Feedback des Kunden und der eigenen Marketingabteilung schneller und konsequenter zu integrieren.

Agilität zu Ende denken bedeutet dann, die Möglichkeiten der Automation in den Bereichen Build, Test und Deployment zu nutzen und durch die Einführung einer Delivery Pipeline die vorgegebene Schlagzahl der agilen Entwicklung aufzugreifen. Zwangsläufig bekommt man dadurch auch die gewünschte, beschleunigte und umfangreichere Rückkopplung von den Nutzern und anderen Beteiligten (Continuous Feedback).

Damit der Zyklus der Entwicklung und Veröffentlichung nicht durch zu viel Feedback ins Stocken gerät, ist es aber auch unerlässlich, für eine Aufbereitung des Feedbacks zu sorgen. Neben der Delivery Pipeline an sich ist deshalb auch im agilen Kontext ein für alle Beteiligten nutzbares, integriertes Change Management unabdingbar. ■

Literatur & Links

[Amb12] Sc. W. Ambler, M. Lines, Disciplined Agile Delivery – A Practitioner's Guide to Agile Software Delivery in the Enterprise, IBM Press 2012

[Kom14] A. Komus, Status Quo Agile; Zweite Studie zu Verbreitung und Nutzen agiler Methoden, Hochschule Koblenz, siehe: http://www.hs-koblenz.de/rmc/fachbereiche/wirtschaft/forschung-projekte/forschungsprojekte/status-quo-agile/

[Lef14] D. Leffingwell, et.al., Scaled Agile Framework, siehe: http://scaledagileframework.com

[oUP10] The openUP process, siehe: http://epf.eclipse.org/wikis/openup

[Pro12] H.-J. Pross, W. Schoepe, Von Agilisten für Agilisten: das Jazz-Projekt, Advertorial im Online-Themenspecial Agility 2012, siehe: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2012/Agility/pross_schoepe_OS_Agility_2012_gdp0.pdf

[Sin14] A. Singh, Do we really need a defect tracking system in agile?, siehe: https://www.scrumalliance.org/community/articles/2014/january/do-we-really-need-a-defect-tracking-system-in-agil.aspx

[SwQ14] SwissQ Consulting AG, Trends & Benchmarks in Software Development 2014

[Wes11] D. West, Analyst Watch: Water-Scrum-fall is the reality of agile, siehe: http://sdtimes.com/analyst-watch-water-scrum-fall-is-the-reality-of-agile

[Wikia14] Wikipedia-Artikel „Lean IT" (englisch), siehe: http://en.wikipedia.org/wiki/Lean_IT

[Wikib2014] Wikipedia Artikel „Kanban (development)" (englisch), http://en.wikipedia.org/wiki/Kanban_(development) und http://de.wikipedia.org/wiki/Kanban_(Softwareentwicklung)