Thomas Rümmler
(thomas.ruemmler@aitgmbh.de)
ist Senior Software Consultant und Projektleiter bei der AIT GmbH & Co. KG. Sein Arbeitsschwerpunkt liegt auf Application Lifecylce Management. Er hilft Unternehmen, ihre Software unter Einsatz des Microsoft Visual Studio Team Foundation Servers effizienter zu entwickeln. Seine Erfahrung gibt er als Autor des TFS-Blogs und Sprecher im Microsoft ALM-Umfeld weiter.

Christian Schlag
(christian.schlag@aitgmbh.de)
ist Consultant bei der AIT GmbH & Co. KG. Er berät und unterstützt Unternehmen bei der Einführung und Anpassung der Microsoft ALM-Plattform Team Foundation Server. Seine Erfahrung gibt er als Autor und Sprecher im Microsoft ALM-Umfeld weiter.

DevOps und Continuous Delivery: sich gemeinsam kontinuierlich verbessern

Softwareentwicklung ist Teamarbeit. Die sich immer mehr verbreitenden agilen Vorgehensmodelle sind das beste Beispiel dafür. Dabei wird beispielsweise ein Sprint bzw. eine Iteration gemeinsam geplant, sich täglich über den Fortschritt ausgetauscht und abschließend gemeinsam reflektiert. Am Ende des Sprints steht ein Softwareprodukt zur Verfügung, welches neue Features und Verbesserungen enthält und von den Kunden genutzt werden kann. Soweit die Theorie. Häufig ist die automatische Auslieferung neuer Softwareversionen mit dem Kopieren der Ergebnisse eines zentralen Server Builds auf ein Netzlaufwerk oder einen Webserver für Entwicklungsabteilungen beendet. Bis dahin ist diese Vorgehensweise als *Continuous Integration* in vielen Entwicklungsteams implementiert. Danach sind oft diverse manuelle Schritte notwendig, um anschließend die Qualitätssicherung durchführen zu können, welche die Codeänderungen mit manuellen und automatischen Tests auf unterschiedlichen Systemen und Umgebungen verifiziert. *Continuous Delivery* erweitert *Continuous Integration* um diese bisher zu wenig beachteten Aktivitäten. Neben diesen vorwiegend technischen Aspekten wird die notwendige Kommunikation zwischen Entwicklungsabteilungen und IT-Administration bzw. Konfigurationsmanagement allzu oft vernachlässigt oder findet überhaupt nicht statt. Hierfür hat sich in den letzten Jahren unter dem Codenamen *DevOps* die gegenseitige Annäherung von Entwicklung und Betrieb etabliert, wodurch *Continuous Delivery* tatsächlich erst realisierbar wird. Der Artikel arbeitet die wesentlichen Bestandteile von *Continuous Delivery* heraus, die damit einhergehende, notwendige Kooperation zwischen Entwicklung und Betrieb und gibt einen Überblick über die zur Realisierung notwendigen Schritte. Der Fokus liegt hierbei auf der konzeptionellen Ebene, technische Beispiele erfolgen unter Verwendung des Microsoft Team Foundation Servers.

Cloud Cadence oder Auslieferung schneller als man klicken kann

Wie waren noch die guten alten Zeiten, in denen sich das Entwicklungsteam nach einer ausgiebigen Entwurfsphase an die Entwicklung machte und nach monatelanger Arbeit seine Resultate an die Testabteilung gegeben hat. Nach einigem Pingpong zwischen Testern und Entwicklern hatte man die Software auch einigermaßen fehlerfrei und das Installationsteam konnte sich um die Erstellung der Installationsmedien kümmern. Gedanken daran, dass man extrem schnell auf äußere Umstände reagieren muss, waren in weiter Ferne, denn die Wettbewerber haben es ja auch nicht besser gemacht.

Was manch einem noch aus eigener Erfahrung als „die gute alte Zeit“ vorkommt, ist für andere unvorstellbar und hört sich an wie eine Beschreibung des Softwaremittelalters. Der Wandel hat jedoch in der IT keine Jahrhunderte gedauert. Vielmehr haben sich die Geschäftsmodelle vieler Softwarehersteller innerhalb weniger Jahre komplett verändert.

Wer früher seine Software noch per DVD vertrieb, betreibt heute eventuell die gleiche Anwendung selbst bei einem der großen Infrastructure as a Service (IaaS) -Provider (eine gute Übersicht der Anbieter bietet GARTNER [Gar13]). Einen Schritt weiter geht noch die komplette Orientierung in Richtung Cloud, wenn man beispielsweise seine Anwendung über einen Platform as a Service (PaaS) -Anbieter komplett als Dienstleistung anbietet (siehe z. B. [RaS12, Seiten 22, 26]).

Ein sehr prominentes Beispiel für diesen Wandel ist Microsoft (vgl. **Abbildung 1**). Zunächst wurde der gesamte Entwicklungsprozess reflektiert und stetig angepasst. Dann wurde der Lebenszyklus der Entwicklungstools stückchenweise verkürzt, sodass das Kundenfeedback immer besser und schneller integriert werden konnte. BRIAN HARRY beschreibt in sei-

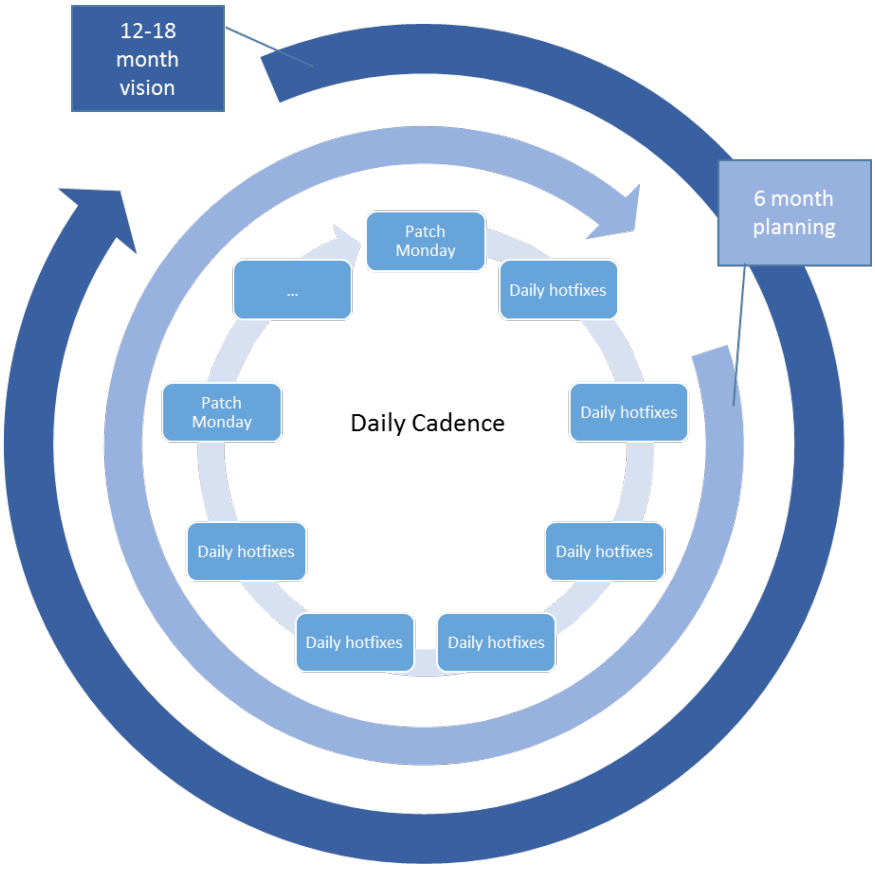

Abb. 1: Cadence bei Microsoft Visual Studio Online (eigene Darstellung in Anlehnung an die Daten aus [Har12])

nem Blog (siehe [Har12]) die Taktung der Entwicklung für Visual Studio Online, einem sehr jungen Produkt von Microsoft, welches in dem Blogbeitrag noch unter dem damaligen Namen Team Foundation Service rangiert.

Der Screenshot der Features Timeline für Visual Studio Online (vgl. Abbildung 2) zeigt, dass die angekündigte Taktung, insbesondere die Auslieferung nach jedem Sprint, umgesetzt ist. In den meisten Monaten gibt es zwei Update-Daten, resultierend aus dreiwöchigen Sprints.

Interessant an dieser Tabelle ist auch die letzte Spalte. Hier wird dargestellt, in welches On-Premise-Release des Produktes die einzelnen Features aufgenommen werden. Dazu jedoch später mehr.

Wenn man nun die eigenen Erfahrungen gegen solch eine Schnelllebigkeit reflektiert, kann man schnell zu der Überlegungen kommen, wie es denn überhaupt möglich ist, so schnell zu reagieren. In vielen unserer Kundenprojekten sehen wir viele manuelle Schritte, verteilte Aufgaben oder verschiedene Skripte, die erforderlich sind, um eine neue Version zu veröffentlichen. Ganz davon abgesehen, dass erst noch aufwendige Tests bestanden und verschiedene Freigaben erlangt werden müssen.

Wie kann man diesem Ziel näher kommen? Welche Rolle spielen technische Unterstützung, Kommunikation und Teamarbeit auf dem Weg, die schnellen Zyklen stabil durchhalten zu können? Die Software muss doch ständig in den Startlöchern für eine Veröffentlichung stehen. Man könnte sagen, sie muss „*continuously available*“ sein, was automatisch zu den Begriffen *Continuous Integration* und *Continuous Delivery* führt.

Continuous Integration und Continuous Delivery

Viele Entwicklungsabteilungen setzen bereits *Continuous Integration* (CI) in ihrem Softwareentwicklungsprozess ein, sodass die Softwarequalität sichergestellt werden kann und jederzeit eine lieferbare Version gewährleistet ist. Dabei werden permanent alle Codeänderungen vor dem Eincheckvorgang auf ihre Korrektheit geprüft. Das heißt, es wird ein Server Build mit dem neuen Quellcode ausgeführt und dabei werden verschiedenste Qualitätsprüfungen durchgeführt. Ist dieser zentrale Build erfolgreich, werden die Codeänderungen im Versionsverwaltungssystem abgelegt.

Neben der reinen Kompilier- und Baubarkeit des Quellcodes werden zugleich automatisch Unit-Tests und Codeanalysen durchgeführt. Somit bekommt der Ent-

Service	Feature	Server
28 Feb 2014	Build support for Java code managed in Git	Future update
	Java JDK, Ant, and Maven libraries preinstalled in hosted build	--
	Maven support for TF version control projects	2013
10 Feb 2014	Exporting test artifacts	Future update
	New "create tags" permission	Future update
22 Jan 2014	Querying tags	Future update
	Removing weekends from the Burndown	Future update
	Configurable CFD dates	Future update

Abb. 2: Ausschnitt der Features Timeline für Visual Studio Online (Screenshot erstellt von [Mic])

wickler nach abgeschlossenem Sever Build direkt Feedback in Form eines positiven oder negativen Build-Ergebnisses. Im ersten Fall werden die Codeänderungen ohne weiteres Zutun auf dem Branch eingecheckt. Im Negativfall muss der Entwickler nacharbeiten bis der Build erfolgreich ist.

Darauf aufbauend erweitert *Continuous Delivery* (CD) dieses Konzept. Es werden nicht nur die interne Integration und schnelles Feedback innerhalb des Entwicklungsteams berücksichtigt, sondern zugleich weitere Rollen (wie zum Beispiel Tester, Administratoren und Release-Manager) sowie Stakeholder einbezogen.

Es ergibt sich ein Zusammenspiel zwischen den verschiedenen Bereichen (→ Teamarbeit). Abbildung 3 zeigt die verschiedenen Bereiche vereinfacht in Form von zwei verschiedenen Farben.

Der neuralgische Punkt bei diesem Zusammenspiel ist die Schnittstelle zwischen Entwicklung (Construct, blau) und Betrieb (Operate, orange). Innerhalb der einzelnen Entwicklungssprints auf der linken Seite wird durch *Continuous Integration* die Qualität gewährleistet. Doch erst durch *Continuous Deployment* wird sichergestellt, dass man stets und unter Verwendung eines hohen Grades an Automatisierung an den Release-Manager bzw. an den Betrieb abgeben kann.

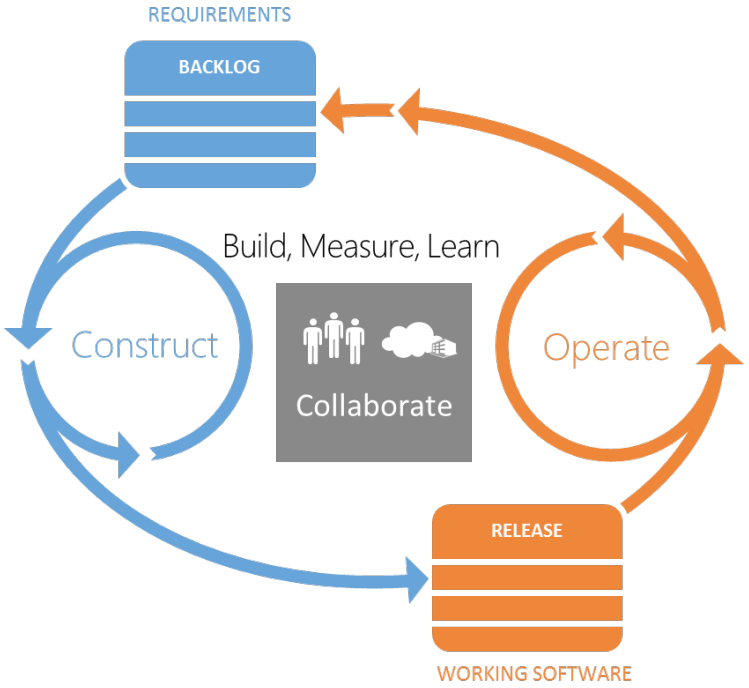

Abb. 3: Build, Measure, Learn-Kreislauf

Konfusion allerorts

Wie MARTIN FOWLER in seinem Blog beschreibt, besteht die Herausforderung in der Softwareentwicklung darin, dass neue Funktionen eines Softwareprodukts nicht nur auf dem PC des Entwicklers funktionieren müssen, sondern die Software stabil und live im Produktivsystem des Kunden in Betrieb sein muss, damit Geld verdient wird [Fow].

Häufig ergeben sich bei der Umsetzung dieser „letzten Meile" aus Unwissenheit oder Missverständnissen aller Beteiligten vermeidbare Verzögerungen. Am gesamten Release-Prozess sind neben Entwicklung und Betrieb weitere Abteilungen involviert, wie z. B. Test- und Release-Management. Um die Problematik zu verdeutlichen wird das Verhältnis zwischen Entwicklung und Administration skizziert.

Beide Abteilungen haben auf den ersten Blick verschiedene Interessen. Denn die Entwickler müssen auf Basis der Stakeholder und deren Anforderungen neue Funktionen und Technologien implementieren, sodass das Softwareprodukt einem stetigen Wandel unterliegt. Die Administratoren, die die verschiedenen Umgebungen betreiben, müssen einen reibungslosen Betrieb aller Systeme gewährleisten und halten an dem Credo „never change a running System" fest. Ein sich ständig änderndes Softwareprodukt torpediert diese Philosophie permanent.

Beide Abteilungen für sich sind in gewisser Weise im Recht. Dennoch müssen sie gemäß des Wertschöpfungsprozesses zusammenarbeiten und ihre Positionen an einer gemeinsamen Strategie ausrichten. Denn wie bereits eingangs erwähnt, ist Softwareentwicklung Teamarbeit und zwar über Abteilungsgrenzen hinaus.

Auf Basis der grundverschiedenen Auffassungen von Entwicklung und Betrieb entsteht bei einem neuen Release mit nicht definiertem Release-Prozess folgende Situation: Am Ende eines Sprints steht eine neue Softwareversion mit zahlreichen Änderungen zur Verfügung. Die Entwicklungsabteilung gibt die Version zum Release frei, vielleicht nur mittels automatisch generierter E-Mail. Die Administratoren sind damit auf sich allein gestellt.

Es ist unklar, welche Anpassungen an Konfigurationsdateien und Skripten (sofern vorhanden) für einen fehlerfreien Betrieb notwendig sind. Eventuell fehlende Ansprechpartner und Dokumentationen erschweren die Situation darüber hinaus. Dass die neue Version umgehend auf den verschiedenen Testsystemen laufen soll, ist dabei selbstverständlich. Mit der Zunahme von verteilten Teams verschärft sich dieses Szenario zusätzlich. In [Edw10] wird dieses implizite Wissen zutreffend mit „Wall of Confusion" bezeichnet (siehe Abbildung 4).

Ein solches „über den Zaun werfen" von Aufgaben hat sehr wenig mit strukturierter Zusammenarbeit oder kontrollierter Prozessschnittstelle zu tun. Es führt regelmäßig zu Verwirrung (Confusion) beim „Paketempfänger" und bietet keinerlei Sicherstellung von Qualitätsmerkmalen. Da dies schnell in einer Katastrophe endet, hat sich in den letzten Jahren unter dem Codenamen *DevOps* die gegenseitige Annäherung von Entwicklung und Betrieb etabliert, wodurch *Continuous Delivery* tatsächlich erst realisier-

Abb. 4: Wall of Confusion (entnommen aus [Edw10])

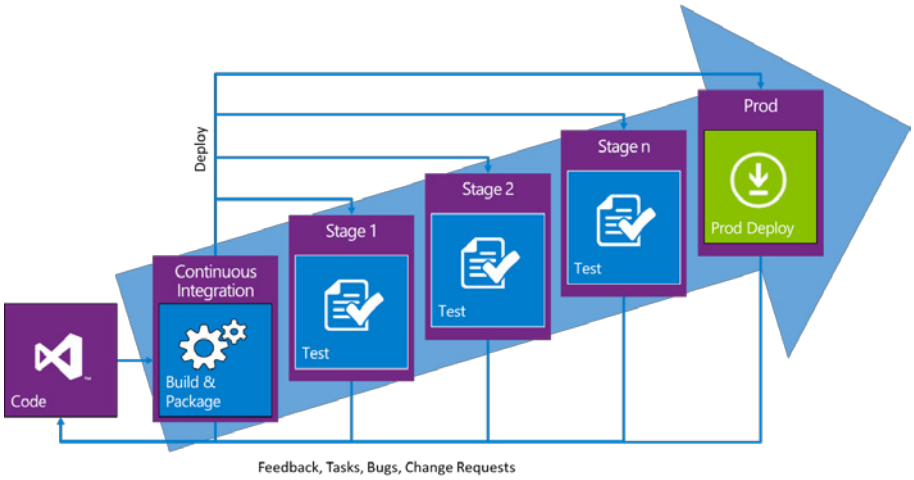

Abb. 5: Deployment-Pipeline

bar wird und die „Wall of Confusion“ stetig minimiert und schlussendlich gänzlich eliminiert wird.

Development + Operations + X = DevOps

Wie nähert man sich nun der „Wall of Confusion“? Welche Maßnahmen haben sich bereits etabliert?

Ähnlich wie bei der Automatisierung von Tests müssen die betroffenen Abteilungen miteinander kooperieren. Es muss ein gemeinsames Bewusstsein dafür geschaffen werden, dass die Inbetriebnahme neuer Softwareversionen nur gelingt, wenn die Entwicklungsabteilung die notwendigen Schritte für den Betrieb der Software kennt. Andererseits muss die Administration ein Verständnis dafür entwickeln können, warum eine Komponente der Software verändert bzw. eine neue Technologie zum Einsatz kommt.

Für Unternehmen, die bereits agile Vorgehensmodelle einsetzen, bieten sich die Daily Standup-Meetings regelrecht an. Denn das Ziel dieser Meetings ist es, den Informationsaustausch unter den Teammitgliedern zu gewährleisten und etwaige Störungen zu identifizieren und anschließend zu beseitigen. Hierzu erscheint die Einbindung eines Mitarbeiters aus dem Betrieb durchaus richtig.

Umgekehrt ist es ebenso sinnvoll, einen Entwickler in den täglichen Meetings der Betriebsabteilung einzubinden. Sollte während des Daily Standup-Meetings weiterer Klärungsdarf bezüglich des Deployments notwendig sein, kann dies nachgelagert erfolgen. Anhand des nun täglich stattfindenden Informationsaustausches halten sich beide Abteilungen auf dem Laufenden und es können notwendige Aufgaben frühzeitig definiert und abgearbeitet werden.

Man könnte es auch so formulieren: Prozesse sind notwendig aber nicht hinreichend. Kommunikation ist wie so oft unabdingbar. Demnach folgt:

$$X = Communication$$

$$DevOps = Development + Operations + Communication$$

Stück für Stück entlang der Deployment-Pipeline

In der vorher dargestellten Zusammenarbeit wird entlang einer Deployment-Pipeline gearbeitet. Diese ist der definierte Prozess, den das gesamte Team einhält. Durch die einheitliche Pipeline werden die notwendigen Qualitätsstufen definiert, die nacheinander passiert werden müssen (siehe Abbildung 5).

Wie in Abbildung 5 dargestellt steht am Anfang der Code. Dieser wird an ein Build System weitergegeben und bei erfolgreichem Durchlaufen dieses Quality Gates in das Source Control Repository eingestellt (*Continuous Integration*).

Die darauffolgenden Stufen bilden den Bereich *Continuous Deployment*. Dabei wird das Ergebnis des zuvor erfolgreich durchgeführten CI-Builds nacheinander an die einzelnen Systeme weitergegeben (deployed). Wie Abbildung 5 verdeutlicht, wird stets dasselbe CI-Build-Ergebnis verteilt. Auf den einzelnen Stufen wird das Deployment-Paket nicht mehr verändert. Das „Weiterreichen“ über die Stufen ist ein rein logischer Schritt.

Physisch wird stets das Build-Ergebnis aus der Stufe *Continuous Integration* auf eine weitere Stufe ausgeliefert. Dies kann je nach Implementierung und Erfordernissen der Anwendung von einem einfachen Kopieren von Dateien bis hin zu komplexen Installationen mit Anpassung von Registry, Webservereinstellungen oder ähnlichem reichen.

Der Aufbau der *Deployment-Pipeline* ist abhängig vom Anwendungsfall, die Anzahl und die Parameter der Qualitätsstufen können variieren. Die hier in Abbildung 5 gezeigten Stufen stellen lediglich Platzhalter dar. Nachfolgend werden exemplarisch ein paar mögliche Stufen erklärt. Dafür wird eine Anwendungssuite als Beispiel verwendet, so wie es Microsoft Office darstellt.

Stufe 1: Code

- Die „Code-Stage“ repräsentiert die Arbeit der Entwickler in ihren lokalen Umgebungen. Es werden lokale Unit-Tests und Builds ausgeführt, jedoch ohne zentral gesteuerte Qualitätskontrollen.

Stufe 2: Continuous Integration

- In dieser Phase wird der Code einzelner Entwickler an ein zentrales Build System abgegeben. Bei diesem Schritt wird die Codeänderung auf Kompatibilität zu eventuell in der Zwischenzeit durch andere Entwickler durchgeführte Änderungen geprüft. Die eingecheckten Änderungen beziehen sich auf eine Komponente, im Beispiel auf Änderungen an MS Word oder MS Excel.

Stufe 3: Test

- Die Summe aller Änderungen an einem Teil der Software-Suite wird getestet. Dies kann manuell oder auch automatisiert (Details zu Testautomatisierung unter [Ors13]), zum Beispiel manuelle UI-Test oder sogenannte Coded-UI-Tests geschehen.

Stufe 4: Integration

- In dieser Stufe wird das Zusammenspiel verschiedener Komponenten überprüft. Die Änderungen an allen Bestandteilen der Software-Suite werden berücksichtigt. Im Beispiel werden Word, Excel, PowerPoint und Co. nicht mehr isoliert betrachtet, sondern in ihrem Zusammenspiel validiert.

Stufe 5: Staging

- Eine mögliche letzte Stufe, bevor ein System in den Produktivbetrieb geht. Je nach Szenario wird hier bereits mit „echten“ Anwendern getestet, das System in seinen wichtigsten Szenarien untersucht, unabhängig von den tatsächlich durchgeführten Änderungen oder auch das Produkt beim Kunden in einer

Parallelumgebung geprüft. Diese Stufe kann also noch beim Softwarehersteller oder auch beim Kunden angesiedelt sein. Diese Stufe ist oft projekt- bzw. softwarespezfisch.

Stufe 6: Prod

- Diese Stufe repräsentiert die finale Produktivumgebung, ist aber nicht zwingend ein Bestandteil der *Deployment-Pipeline*. Diese Stufe repräsentiert jedoch spätestens das Ende der Pipeline.

Zwischen *Continuous Integration* und *Prod* können 0 – n Stufen liegen. Je strenger der Freigabeprozess ist oder je mehr regulativen Vorgaben ein Hersteller unterliegt, desto höher ist tendenziell die Anzahl der Stages dazwischen. Mit steigender Anzahl ist in Konsequenz auch eine längere Durchlaufzeit durch die Pipeline verbunden.

Jede einzelne Stufe liefert Rückmeldungen, die sich wiederum auf den Code am Anfang der Pipeline auswirken. Wird beispielsweise ein Bug gefunden, so wird der Fluss durch die Pipeline typischerweise unterbrochen, der Bug im Code beseitigt und die Pipeline erneut durchlaufen. Dies kann jedoch auch abhängig von einer Bewertung des Bugs sein.

Auf die gleiche Art können allgemeines Feedback, z. B. zur User Experience, Aufgaben oder auch Change Requests in jeder einzelnen Stufe entstehen. Diese haben nicht unbedingt zur Folge, dass die Pipeline unterbrochen wird. Dieses Vorgehen stellt jedoch sicher, dass diese Informationen strukturiert erfasst werden und nicht verloren gehen sowie Entscheidungspunkte eingehalten werden.

Durch die vorgegebene Definition der Pipeline ist ein gleichartiger Weg entlang der Pipeline sichergestellt. Dadurch wird eine weitgehende Automatisierung erst ermöglicht. Ein relativ junges Werkzeug für diese Automatisierung und geführte Unterstützung entlang der *Deployment-Pipeline* bietet Microsoft mit Release Management für Visual Studio. Dieses ermöglicht die Definition der Pipeline und der workflowgestützten Abarbeitung der zu erledigenden Aufgaben. Nähere Informationen sind auf der Produktwebseite zu finden [MicRM].

Der Einsatz eines solchen Werkzeuges unterstützt bei der Organisation, Wartung und natürlich beim Durchlaufen der *Deployment-Pipeline*. Durch die enge Verzahnung mit dem Team Foundation Server [MicTFS] als Hilfsmittel zur Arbeitssteuerung (wie Rückmeldungen aus der Deployement-Pipeline), als Quellcodeverwaltung sowie als Build-Management-System sind Einhaltung des Ablaufs sowie Datenkonsistenz gewährleistet.

Fazit

Das gezeigte Zusammenspiel zwischen den verschiedenen Bereichen, die zum erfolgreichen Erstellen und Betreiben von Anwendungen erforderlich sind, lässt sich sehr gut durch die Definition eines Prozesses unterstützen. Solch eine *Deployment-Pipeline* ermöglicht die kontinuierliche Einhaltung der Qualitätsstufen sowie deren ständige Erweiterung und Verbesserung.

In jeder einzelnen Stufe der *Deployment-Pipeline* ist es möglich, Ergebnisse und Feedback in das Produkt zurückfließen zu lassen, die wiederum zur kontinuierlichen Verbesserung der eigentlichen Anwendung beitragen. Hier spiegelt sich der Titel der Abbildung 3 wieder: Build, Measure, Learn.

Das dargelegte Vorgehen hat jedoch auch seine Grenzen. Je näher die Anwendung an einem in der Cloud betriebenen Dienst ist, desto besser lässt sich die Pipeline bis in die Produktivumgebung hinein durchziehen. Handelt es sich jedoch um ein eher traditionelles Geschäftsmodell, bei dem der Independent Software Vendor (ISV) gar keinen Zugriff auf die Produktivumgebung hat oder in der Anzahl der Releases beschränkt ist, weil z. B. mit jeder Aktualisierung aufwendige Validierungsschritte notwendig sind, so kann man von der Pipeline lediglich bis zur internen Qualitätssicherung profitieren.

Eine gute Kombination beider Welten lebt Microsoft vor. Das eingangs erwähnte Beispiel Visual Studio Online erfährt, so wie in Abbildung 2 dargestellt, Aktualisierungen in sehr kurzen Zyklen. Diese Plattform wird zugleich genutzt, um die Reaktion der Nutzer auszuwerten (Measure) und die Features in ein späteres Release der On-Premise-Variante des Produktes, dem Team Foundation Server, einfließen zu lassen (siehe letzte Spalte in Abbildung 2).

Die teilweise unterschätzte Schnittstelle zwischen Entwicklung und Betrieb gewinnt also an Bedeutung. Wer mit der hohen Geschwindigkeit, insbesondere im Bereich der Online-Applikationen, mithalten möchte, sollte dieses Thema nicht außer Acht lassen und diese Schnittstelle optimieren. ■

Quellen

[Gar13] L. Leong, D. Toombs, B. Gill, G. Petri, and T. Haynes, "Magic Quadrant for Cloud Infrastructure as a Service," Gartner.com, 2013. [Online]. Available: http://www.gartner.com/technology/reprints.do?id=1-1IMDMZ5&ct=130819&st=sb. [Accessed: 16-Mar-2014].

[Ras12] T. Rümmler and R. Stropek, "Cloud Computing - was bringt's?," in Software-Architektur 2. Symposium, 2012, pp. 21–28.

[Har12] B. Harry, "Brian Harry's blog," MSDN Blogs, 2012. [Online]. Available: http://blogs.msdn.com/b/bharry/archive/2012/08/28/tfs-shipping-cadence.aspx. [Accessed: 14-Mar-2014].

[Mic] Microsoft, "Team Foundation Service Features Timeline," Visual Studio News. [Online]. Available: http://www.visualstudio.com/en-us/news/release-archive-vso. [Accessed: 14-Mar-2014].

[Fow] M. Fowler, "Delivery Guide." [Online]. Available: http://martinfowler.com/delivery.html. [Accessed: 16-Mar-2014].

[Edw10] D. Edwards, "What is DevOps ?," 2010. [Online]. Available: http://dev2ops.org/2010/02/what-is-devops/. [Accessed: 16-Mar-2014].

[Ors13] N. Orschel, "Ein Dialog unter Fremden : Testautomatisierung in der Praxis," OBJEKTspektrum Online-Themenspecial Test., pp. 1–5, 2013.

[MicRM] Microsoft, "Release Management." [Online]. Available: http://www.visualstudio.com/en-us/explore/release-management-vs.aspx. [Accessed: 16-Mar-2014].

[MicTFS] Microsoft, "Team Foundation Server." [Online]. Available: http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx. [Accessed: 16-Mar-2014].