DevOps light:
DevOps in regulierten Umgebungen

Insbesondere bei Banken und Versicherungen, aber auch im Bereich der Pharma-Software, gibt es weitgehende Vorgaben und Regulierungen für den Umgang mit Software, mit der Entwicklung und mit dem Betrieb, um Risiken sowohl durch Unachtsamkeit und Fehler als auch durch gewollte Angriffe zu minimieren. Dies hat in der Regel getrennte Bereiche für Entwicklung und Betrieb mit geringer Kommunikation und hohem Aufwand für jede Lieferung einer Software zur Folge. Die damit verbundenen Kosten sind ein klares Problem. Dieser Artikel zeigt, wie sich auch in diesen Umgebungen die Vorteile des DevOps-Ansatzes nutzen lassen, um den Aufwand für Lieferungen zu senken und die Software in einer Form zu entwickeln, die für den Betrieb hilfreich ist und die in der Produktionsumgebung die gewünschten nicht-funktionalen Anforderungen erfüllt.

Ein Thema, das in der Systementwicklung im Allgemeinen und besonders im agilen Umfeld gerne übersehen wird, ist die IT-Governance. Spätestens in der täglichen Verwendung und im Betrieb der Software wird dieses Thema sehr relevant und ist in den Strukturen und Prozessen der IT von Organisationen fest etabliert und notwendig. IT-Governance zielt unter anderem darauf ab, den Wertbeitrag – und damit die Effizienz und Effektivität der IT – zu erhöhen, Risiken zu verwalten und die Geschäftsseite mit der IT abzustimmen (das häufig zitierte Business/IT-Alignment). Um dies zu erreichen, gilt es, Strukturen und Prozesse für die IT zu definieren, Verantwortlichkeiten und Rechenschaftspflichten eindeutig und transparent festzulegen sowie Risiken zu entdecken und diese angemessen zu berücksichtigen (vgl. [Joh11]).

In Branchen wie Versicherungen, Banken, Medizintechnik oder der Pharmaindustrie muss die IT-Governance zusätzlich noch dafür sorgen, dass externe Vorgaben eingehalten werden (*Compliance*). Ein Beispiel hierfür ist die Vorgabe der Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin), nach der Entwicklung und Test vom Produktionsbetrieb getrennt sein müssen (vgl. [BaF12], Kapitel AT7.2) sowie der IT-Grundschutzkatalog des BSI eingehalten werden soll (vgl. [BSI]). Dies endet nicht mit den Ländergrenzen, insbesondere internationale Konzerne müssen in vielen Fällen auch weiteren Vorgaben wie denen des amerikanischen Sarbanes-Oxley-Gesetzes (vgl. [Wik]) folgen.

Derartige Vorgaben führen im Regelfall zu getrennten Bereichen für Entwicklung und Betrieb, die stark widersprüchliche Ziele haben und wenig miteinander kommunizieren. Lieferungen von Softwareprodukten aus der Entwicklung in den Betrieb sind sehr selten und bedeuten hohe Risiken, die mit entsprechend hohem Aufwand entschärft werden müssen. Jeder, der in diesen Branchen in der Softwareentwicklung oder im Betrieb gearbeitet hat, kennt die Wochenendtermine, zu denen ein großer Teil der Entwicklungs- und Betriebsmannschaft vor Ort ist, um auf unvorhergesehene Zwischenfälle zeitnah reagieren zu können.

Der Status quo

Jede Firma ist durch ihren Geschäftszweck getrieben – daher müssen auch alle Einheiten der Firma auf diesen Geschäftszweck hinarbeiten. Dies beinhaltet nicht nur den Fachbereich, sondern die Entwicklung genauso wie den Betrieb und die Infrastruktur (meist ein Teil des Betriebsbereichs). Da der Fachbereich als Vertreter des Geschäftszwecks über das Budget verfügt und die IT-Abteilungen mit diesem Budget arbeiten, entsteht innerhalb der Firma eine Kunde-Dienstleister-Beziehung. Der folgerichtige Ansatz an dieser Stelle ist es, die IT als Kostenstelle zu betrachten. Diese Sicht passt sehr gut zu dem allgemeinen Trend der letzten 20 Jahre, Kosteneffizienz durch die Einführung von klaren Prozessen zu erzielen (**siehe Abbildung 1**).

Aus dieser Betrachtungsweise und aus den daraus entstehenden Prozessen entwickelt sich automatisch eine starke Trennung der verschiedenen Bereiche der IT mit unterschiedlichen Zielen. So möchte ein Entwicklungsbereich möglichst schnell neue Produkte und Funktionalitäten für die Kunden zur Verfügung stellen, da erst mit dem Produktionsgang die Geschäfts- oder Unternehmenswerte realisiert werden. Der Betrieb auf der anderen Seite möchte kein Risiko eingehen und die betriebliche Stabilität der Software in keiner Weise gefährden. Der Infrastruktur schließlich möchte man auch kein zusätzliches Budget zukommen lassen, da hierdurch (normalerweise) keine zusätzlichen Geschäftswerte realisiert werden. Und falls doch Investitionen getätigt werden müssen, sollen diese möglichst zukunftsfähig und durch zusätzlichen Geschäftswert direkt finanzierbar sein.

Diese widersprüchlichen Ziele fragmentieren den IT-Bereich häufig auf verschiedensten Ebenen in organisatorische Inseln. Da

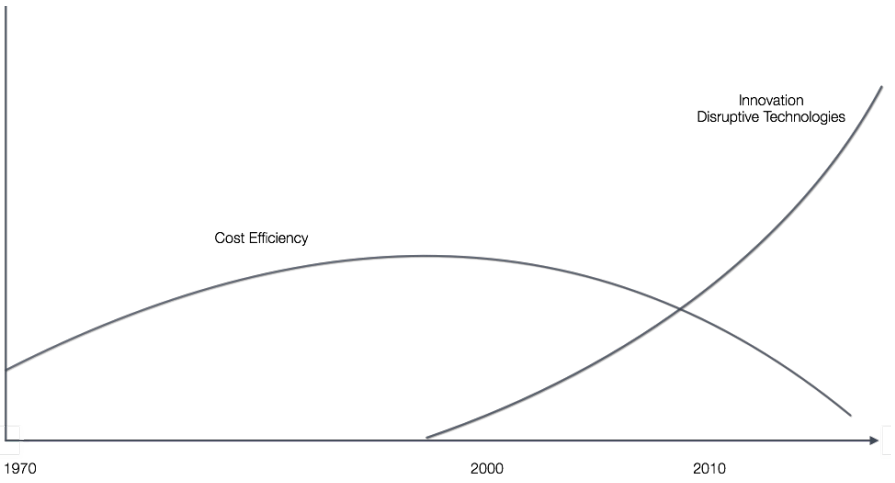

Abb. 1: Kosteneffizienz und Innovation.

der variable Gehaltsanteil der Mitarbeiter und Führungskräfte von dem Erreichen der gegenläufigen Ziele abhängt, entsteht häufig ein Feindbild zwischen den Inseln. Diesem liegt die implizite Annahme zu Grunde, dass wir ein Nullsummen-Spiel haben, bei dem das bessere Erreichen der Ziele des einen Bereichs automatisch das schlechtere Erreichen der Ziele des anderen Bereichs bedeutet. Dies führt wiederum zu sehr rigiden Prozessen, starren Schnittstellen, wenig Kommunikation und sehr hohem zeitlichem Aufwand, um diesen Übergängen zu folgen, zum Beispiel um eine Software in Produktion zu bringen. Häufig wird diese Rigidität und Starrheit noch mit einem Verweis auf gesetzliche Vorgaben zementiert, auch wenn diese nicht in dem Maße einschränken, das propagiert wird.

Der Übergang vom Fachbereich zur IT wird durch ähnlich rigide Prozesse und starre Schnittstellen bestimmt. Diese Rigidität entsteht durch den weit verbreiteten Wasserfall-Ansatz und seine Varianten. Dieses ursprünglich von Winston Royce erfundene Vorgehen (vgl. [Roy70) definiert einen sequenziellen Prozess von der Spezifikation bis zur Auslieferung. Interessanterweise hielt Royce bereits in der Originalveröffentlichung fest, dass ein strikt sequenzieller Prozess sehr hohe Risiken birgt und ein iterativer Prozess mit starker Kommunikation zwischen den einzelnen Schritten eine sehr viel höhere Wahrscheinlichkeit für einen Erfolg bietet. Diese Einsicht wurde aber bis zum Erfolg der agilen Methoden weitgehend ignoriert. Aus Fachbereichssicht ist es vielmehr immer noch häufig so, dass eine Spezifikation „über den Zaun geworfen wird“, dann von der IT umgesetzt und zur Prüfung sowie zum Test vorgelegt wird. In der Zwischenzeit ist der Fachbereich bereits vollständig mit dem Schreiben der nächsten Spezifikation ausgelastet und hat keine oder nur wenig Zeit für Rückfragen. Damit haben wir das nächste Feindbild zwischen Fachbereich und IT, wobei der Fachbereich nicht verstehen kann, wieso die IT unverschämt viel Budget für einfache Funktionalität haben möchte, und die IT nicht nachvollziehen kann, warum der Fachbereich die einfachsten technischen Einschränkungen nicht akzeptieren kann.

Das Problem

Die letzten 20 Jahre in der IT waren geprägt von Effizienzsteigerungen und Prozessoptimierungen. Die hierdurch erzielte Kosteneffizienz konnte am Markt verwendet werden, um sich gegen den Mitbewerber durchzusetzen. Wir sind allerdings inzwischen an einem Punkt angelangt, an dem weitere Effizienzsteigerungen nur mit sehr hohen Investitionen möglich sind (insbesondere in regulierten Umgebungen). Damit ist eine Differenzierung im Markt über Preissenkungen nicht mehr möglich und eine weitere Marktdurchdringung über die Effizienz nicht erreichbar.

Deshalb erfolgt in den letzten Jahren mehr und mehr ein Umdenken in Richtung Innovation und Innovationsgeschwindigkeit, um am Markt ein Alleinstellungsmerkmal zu haben. Dies lässt sich auch sehr deutlich in vielen Märkten beobachten, die mehr und mehr von Produkten mit einem kürzeren Lebenszyklus geprägt werden.

Natürlich ließe sich argumentieren, dass dieses Problem zumindest in regulierten Märkten nicht auftritt, aber das ist ein Trugschluss. Auch hier ist Innovationsgeschwindigkeit ein Alleinstellungsmerkmal, das sehr viel interessanter ist als reine Kosteneffizienz. Damit werden immer mehr Mitbewerber am Markt auf Innovation und neue Produkte setzen. Das führt dazu, dass bei zu langen Entwicklungszyklen ein Produkt noch während seiner Entwicklungszeit obsolet werden kann. Ein kleines Beispiel sind die aktuellen Überlegungen der Deutschen Bank, die Postbank zu verkaufen, während die Integration der IT-Systeme der beiden Häuser noch in vollem Gange ist.

In nicht regulierten Märkten ist die Situation noch volatiler. Durch das immer stärkere Sinken der Einstiegskosten für Softwareentwicklung ist ein Eintritt in einen Mark mit immer niedrigeren Investitionen verbunden. Dies führt zu einer zunehmenden Zahl von disruptiven Innovationen, die – zunächst nur in einer Marktnische erfolgreich – früher oder später einen Markt überrollen können (vgl. [Chr97]) und damit auch etablierte Softwarefirmen gefährden.

Wir haben also auf der einen Seite unsere starre Fachbereichs- und IT-Organisation mit ihren Inseln (oder Silos), die sich über rigide Prozesse und Schnittstellen abschotten, auf der anderen Seite haben wir den massiven Bedarf an Innovationsgeschwindigkeit, um den Firmenzweck in einem schnell veränderlichen Markt weiterhin positionieren zu können. Gleichzeitig haben wir aber Compliance-Vorgaben und gesetzliche Regelungen, die diese Inseln zementieren.

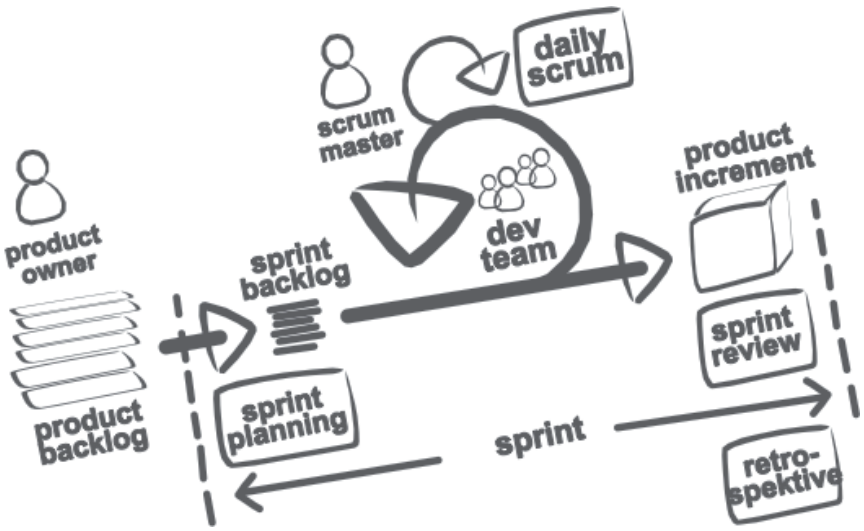

Abb 2: SCRUM im Überblick.

Agilität als erster Baustein

Agile Methoden sind aktuell in aller Munde, aber bereits in den letzten 15 Jahren zunehmend für Entwicklung und Projektmanagement auf dem Vormarsch (vgl. [Bec01]). Scrum ist beim Projektmanagement inzwischen der klare Gewinner, mit Kanban als der Alternative für Wartung und Weiterentwicklung. Insbesondere Scrum wird heutzutage mit Agilität gleichgesetzt und mit mythischen Qualitäten besetzt (**siehe Abbildung 2**). So soll die Software gleichzeitig schneller, billiger und qualitativ hochwertiger sein und sich täglich an die Wünsche des Fachbereichs anpassen.

Die Wirklichkeit ist nicht ganz so strahlend – vor allem sind agile Ansätze durch die deutlich erhöhte Kommunikation und die intensive Einbeziehung des Fachbereichs im ersten Schritt teurer als klassische Wasserfall-Methoden. Wenn wir also eine vollständige Spezifikation haben und eine klare Vorstellung der verwendeten Technologien und wenn sich beide im Verlauf des Pro-

jekts nicht ändern, dann ist der Wasserfall-Ansatz deutlich effizienter.
Der bei agilem Projektmanagement zusätzlich benötigte Kommunikationsaufwand kann in Teilen durch die Automatisierung aller Entwicklungsprozesse – Test, Build, Integration und Deployment – kompensiert werden und in Teilen durch das höhere Engagement eines agilen Teams und des integrierten Fachbereichs ausgeglichen werden. Massive Vorteile können agile Vorgehensweisen aber erst dann bringen, wenn die Spezifikation unvollständig ist oder sich im Verlauf des Projekts ändert. Denn dann spielt der ständige Informationsaustausch zwischen Fachbereich und Entwicklungsteam seine Stärken aus und das Ergebnis des agilen Projekts ist deutlich effektiver. Das Wasserfall-Modell ist hierbei auch weiterhin das effizientere Modell, liefert aber nicht das benötigte Ergebnis.
Durch agile Ansätze können wir also prinzipiell die Mauern zwischen Fachbereich und Entwicklung einreißen und dafür sorgen, dass eine intensive und produktive Zusammenarbeit beginnt. Natürlich muss sich der Fachbereich umstellen und sich auf die neue und intensive Kommunikation mit der IT einlassen, aber das Argument der zeitnahen und flexiblen Umsetzung von aktuell gewünschter Funktionalität ist in fast allen Fällen Vorteil genug, um die Mauern, die historisch bedingt und durch den Wasserfall-Ansatz begünstigt entstanden sind, fallen zu lassen. Auch aus Entwicklungssicht ist ein agiles Vorgehen vorteilhaft, denn plötzlich entsteht ein Dialog darüber, wie eine Funktionalität auszusehen hat, Unklarheiten können viel schneller beseitigt werden und auch ein Feedback zu einer Umsetzung kann man viel schneller erhalten.
Wir haben aber weiterhin das Problem, dass das letzte Glied in der Kette – der Betrieb – nicht involviert und weiterhin abgeschottet ist.

DevOps als zweiter Baustein

Im Rahmen agiler Vorgehensweise mit den typischen Zykluszeiten von zwei bis vier Wochen hat sich in vielen Projekten eine möglichst weitgehende Automatisierung der Deployment-Pipeline vom Entwickler bis in die Produktion als sehr hilfreich erwiesen. Durch diese Automatisierung werden Unsicherheiten bei Releases beseitigt, die entsprechenden Skripte sind hundertfach getestet und – richtig aufgesetzt – ist auch ein Rollback problemlos möglich. Das nimmt dem Release-Vorgang einen großen Teil des Risikos und der Kosten.
Wenn wir aber in der Lage sind, unsere Software sehr schnell und mit geringem Risiko zu liefern, dann macht es plötzlich Sinn, die Zyklen weiter zu verkürzen und auch kleine Funktionen sofort nach Abnahme durch den Fachbereich in Produktion zu bringen. Das führt zu einer grundsätzlichen Änderung des Entwicklungsprozesses mit geringeren Kosten, geringerem Risiko und geringerer Wartezeit bis zur Umsetzung. Dieser geänderte Prozess wird als kontinuierliche Lieferung oder *Continuous Delivery* bezeichnet (vgl. [Bau13] und **siehe Abbildung 3**).
Das Problem ist allerdings, dass wir einen Verantwortungsübergang von der Entwicklung zum Betrieb haben, der die Einbettung einer solchen Continuous-Deployment-Pipeline in vielen Fällen schwierig macht. Insbesondere in regulierten Umgebungen ist die simple Umsetzung von Continuous Delivery zum Scheitern verurteilt.
Um hier erfolgreich zu sein, müssen wir einen Schritt weiter gehen und uns dem Thema DevOps zuwenden. Die naive Vorstellung ist, dass DevOps einfach Entwicklung und Betrieb zusammenlegt, aber diese Vorstellung ist falsch und wäre in den meisten Firmen auch nicht sinnvoll.
Vielmehr ist DevOps ein Ansatz, der auf hohe Lieferungsfrequenz, niedrige Fehlerraten und geringe Zeit für die Fehlerreparatur fokussiert. Hierzu gehören handelnde Personen und ihre Kultur, Prozesse und ihre Einhaltung sowie gemeinsame Werkzeuge und Werkzeugketten (*Tool Chains*). Gerade die Werkzeugketten sind unter dem Thema *Continuous Deployment* sehr bekannt geworden und werden inzwischen weitgehend beherrscht.
Hingegen sind sowohl die Personen, Kulturen als auch die Prozesse in den meisten großen Firmen und insbesondere in regulierten Umgebungen sehr stark zementiert und es scheint unwahrscheinlich, dass die Barrieren zwischen Entwicklung und Betrieb aufgehoben werden können, um kurzfristig eine gemeinsame Kultur, übergreifende Prozesse und Werkzeuge zu etablieren.
Aber es ist möglich, einige Ideen von DevOps zu verwenden und in den beteiligten Bereichen unabhängig voneinander zu etablieren, während parallel durch die Etablierung von spezifischen Prozessen die Kommunikation zwischen den Bereichen gefördert wird.

Notwendige Änderungen

Der Betrieb und seine Belange müssen stärker in die früheren Schritte involviert werden:

- Im ersten Schritt muss bereits in der ersten groben Spezifikationsphase, die jedem Projekt vorausgeht, der Betrieb mit involviert sein. Dies sorgt dafür, dass die betriebsspezifischen Anforderungen und Einschränkungen, an die weder Fachbereich noch Entwicklung denken, mit berücksichtigt werden.
- Im zweiten Schritt wird der Betrieb auch in die Planung einer jeden Iteration der Entwicklung mit einbezogen. Damit kann auch in diesem Teil der Planung auf den Betrieb Rücksicht genommen werden.
- Im dritten Schritt bekommt der Betrieb einen Teil der Arbeitsleistung in Form von Story-Points für jeden Sprint zur eigenen Verfügung, um im Projekt Betriebsanforderungen umsetzen zu können. Für diese Story-Points gibt es einen

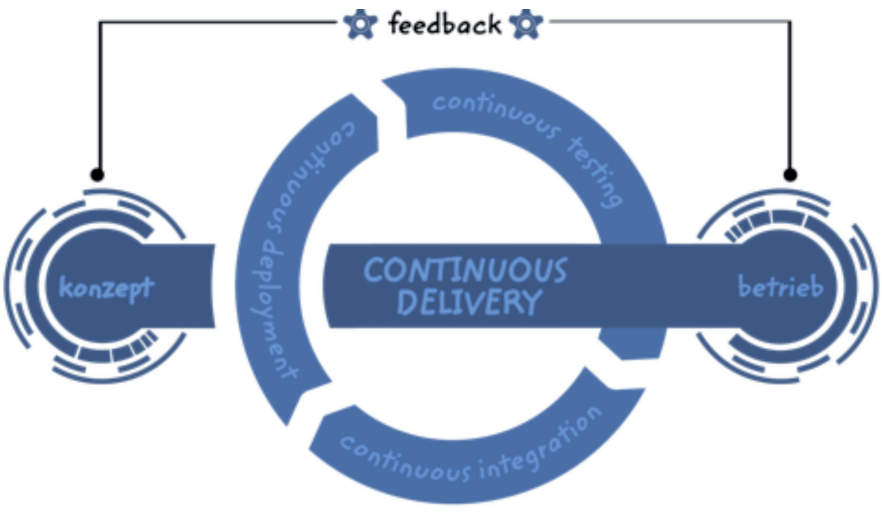

Abb. 3: Continuous Delivery als Integration von Fachbereich, IT und Betrieb.

eigenen Verantwortlichen (gewissermaßen einen Product Owner für Betrieb).

Im Gegenzug stellt der Betrieb die automatische Provisionierung sämtlicher Entwicklungs- und Testsysteme zur Verfügung und ist für die Continuous-Deployment-Pipeline verantwortlich. Bereits zu Beginn des Projekts stehen die Umgebungen zur Verfügung, die sich gemeinsam mit den Anforderungen des Projekts und des Betriebs weiterentwickeln und sich mehr und mehr an die sich gleichfalls ändernde Produktionsumgebung anpassen. Damit beginnt die Integration von Software und Infrastruktur bereits am Anfang des Projekts und sorgt nicht zum Schluss in der Transitionsphase für Probleme.

Gemeinsame Werkzeuge zwischen Entwicklung und Betrieb zu etablieren, ist nicht immer ganz einfach, aber möglich und wird an verschiedenen Stellen bereits durchgeführt. Interessant wird es, hier zusätzlich den Fachbereich mit einzubeziehen, sodass Testwerkzeuge nicht nur in der IT, sondern auch durch die Experten verwendet werden können – und dies sowohl für funktionale als auch für nicht-funktionale Anforderungen.

Eine Continuous-Deployment-Pipeline, die nicht nur den Build durchführt, sondern die Testsysteme eigenständig provisioniert, Datenbanken installiert, mit Testdaten versieht und sämtliche Tests der jeweiligen Stages automatisiert durchführt, lässt sich gleichfalls implementieren und ist für diesen Ansatz besonders wichtig. Hier besteht ein sehr hoher Aufwand darin, zuerst einen hohen Grad an Testautomatisierung auf den verschiedenen Ebenen der Testpyramide zu erreichen, dann vernünftige Testdaten zur Verfügung zu stellen und die im Laufe einer jeden Entwicklung stattfindenden Schemaevolutionen der beteiligten Datenbanken abzubilden. Alle diese Informationen, in einem Repository untergebracht, erlauben im besten Fall das Installieren eines beliebigen Versionsstandes eines Projekts ohne Probleme. Dies lässt sich für Testzwecke sehr gut nutzen, für den Wechsel der Produktivsysteme zu einer älteren Version ist dies aber im Normalfall nicht geeignet, da es in der Regel keine Skripte gibt, die eine Rückwärtsmigration neuer Echtdaten erlauben.

Ergebnis

Wir haben damit klare Verantwortungsbereiche auf der einen und Einflussmöglichkeiten auf der anderen Seite. Fachbereich, Entwicklung und Betrieb sprechen im Rahmen der Planungstreffen regelmäßig miteinander. Die Entwicklung setzt – dem zugeordneten Kontingent entsprechend – die für den Betrieb notwendige Funktionalität um und kann diese auf den durch den Betrieb zur Verfügung gestellten Umgebungen testen mit der Sicherheit, dass diese in den relevanten Punkten der Produktionsumgebung entsprechen. Um allen Beteiligten die notwendige Planungssicherheit zu geben, sollten alle gegenseitigen Verpflichtungen explizit in einem Projektmanifest festgehalten werden. Hierbei ist es wichtig, dass sich der Betrieb genau wie der Fachbereich als Teil des Projekts versteht.

Mit den entsprechenden Werkzeugen und Werkzeugketten sind diese Projekte viel transparenter und es ist zu jedem Zeitpunkt nachvollziehbar, welche Änderungen an Quelltext, Infrastruktur und Daten stattgefunden haben. Lieferungen sind zu 100 Prozent automatisiert und revisionssicher dokumentiert. Gleichzeitig erhöht sich die mögliche Geschwindigkeit der Lieferungen, was das Risiko der einzelnen Lieferung reduziert und gleichzeitig die Kosten für nicht in Betrieb genommene Software senkt. Natürlich können immer noch Fehler passieren, aber jetzt lässt sich die Ursache nachvollziehen und zielgerichtet beheben.

Dies funktioniert auch in Fällen, in denen der Betrieb an einen Dienstleister ausgelagert ist. Dort ist der Fokus auf den Verantwortungsübergang und die reibungslose Übergabe der Ergebnisse noch wichtiger als bei internen Bereichen, da hier die Nichteinhaltung von *Service Level Agreements (SLAs)* bares Geld bedeutet.

Dieses Vorgehen ließe sich als DevOps Light bezeichnen, weil viele der Bereichsgrenzen nicht aufgehoben werden und ganz explizit keine gemeinsame Kultur etabliert werden soll. Natürlich besteht die Hoffnung, dass sich diese entwickelt, aber für ein Funktionieren des Ansatzes ist dies nicht notwendig, und zumindest von externen Dienstleistern vielleicht auch nicht gewünscht. ||

Literatur & Links

[BaF12] Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin), Rundschreiben 10/2012 (BA) – Mindestanforderungen an das Risikomanagement (MaRisk), 2012, siehe: `http://www.bafin.de/SharedDocs/Veroeffentlichungen/DE/Rundschreiben/rs_1210_marisk_ba.html`

[Bau13] J. Baumann, B. Spanneberg, R. Hötschl, Continuous Delivery – Ein Überblick. dpunkt.verlag 2013

[Bec01] K. Beck et al., Manifest für Agile Softwareentwicklung, 2001, siehe: `www.agilemanifesto.org/iso/de/`

[BSI] Bundesamt für Sicherheit in der Informationstechnik (BSI), IT-Grundschutz-Kataloge, siehe: `https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/itgrundschutzkataloge_node.html`

[Chr97] C. Christensen, The innovator's dilemma: when new technologies cause great firms to fail, Harvard Business School Press 1997

[Joh11] W. Johannsen, M. Goeken, Referenzmodelle der IT-Governance (2. aktual. u. erw. Aufl.), dpunkt.verlag 2011

[Roy70] W.W. Royce, Managing the Development of Large Software Systems, in: Proceedings of IEEE WESCON 26, 1970

[Sch13] K. Schwaber, J. Sutherland, Der Scrum-Guide, 2013, siehe: `http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf`

[Wik] Wikipedia, Sarbanes-Oxley Act, siehe: `http://de.wikipedia.org/wiki/Sarbanes-Oxley_Act`

Der Autor

|| Dr. Joachim Baumann
(joachim.baumann@codecentric.de)
ist Geschäftsführer der codecentric AG in Frankfurt/Main, Autor von Büchern und Artikeln zu Themen im Bereich Build Management, Continuous Delivery und Agilität und regelmäßiger Sprecher auf Konferenzen.