Zusammenspiel von Agilität, DevOps, Microservices und Cloud

Maximale Agilität mit Softwareentwicklung 4.0

Es gibt kaum ein Unternehmen, das nicht auf agile Methoden setzt. Trotzdem können sich die wenigsten in puncto „Time-To-Market" mit Online-Händlern wie OTTO [OTTOblog], Zalando und Netflix [NetflixBlog] messen. Zalando spricht sogar von „Radical Agility" [Bis15]. Solch radikale Ausprägungen verdeutlichen, dass die übliche „Standard-Agilität" oft nicht mehr ausreichend ist. Es bedarf der Softwareentwicklung 4.0, die sich durch das Zusammenspiel von Agilität, DevOps, Cloud und Microservices auszeichnet. Der Beitrag zeigt, warum nur das perfekte Zusammenspiel dieser vier Disziplinen die Voraussetzung schafft, in puncto Zeit, Reaktionsfähigkeit und Flexibilität zu den agilsten Unternehmen zu zählen.

Viele Unternehmen befinden sich aktuell in einem Transformationsprozess zu mehr Agilität. Die Gründe sind vielfältig, wie die häufigsten Ziele laut dem 13. jährlichen, internationalen Bericht zum Status von Agilität [SoAR13] zeigen:

- eine beschleunigte Softwareauslieferung (74 Prozent der Befragten),
- das schnelle Reagieren auf sich ändernde Prioritäten (62 Prozent),
- eine höhere Produktivität, beziehungsweise geringere Projektkosten (51 Prozent),
- eine bessere Abstimmung zwischen Fachbereich und IT (50 Prozent),
- eine verbesserte Softwarequalität (43 Prozent).

Mit dem Ziel der beschleunigten Softwareauslieferung ist der Wunsch verbunden, in sehr kurzer Zeit neue Ideen umzusetzen und sich einen Wettbewerbsvorteil zu verschaffen. Solche Ideen betreffen das Kerngeschäft und sind zumeist in unternehmenskritischen Kernsystemen umzusetzen.

Reifegrade Agilität

Anhand der genannten Aspekte zeigt sich, dass es zweckmäßig ist, den Reifegrad von Projekten anhand der Umsetzungszeit geschäftskritischer Ideen zu bestimmen (**siehe Abbildung 1**). Das erste im Produktivbetrieb eingesetzte Release eines neu entwickelten und komplexen IT-Systems dauert auch heute noch zwischen sechs und 15 Monaten. Danach können Release-Zyklen und damit die Umsetzungszeit neuer Ideen deutlich kürzer ausfallen. Anhand der Umsetzungszeit lassen sich drei Reifegrade von Softwareentwicklungsprojekten klassifizieren – *agil*, *agiler* und *agilissimo*:

Das Standardprojekt ist agil, wenn man 6 bis 12 Monate benötigt, um eine Idee im Rahmen eines Softwareentwicklungsprojekts in Produktion zu bringen. Voraussetzung hierfür sind mindestens 3 bis 4 Releases der IT-Systeme pro Jahr, wobei kleine Änderungen oder Fehlerbehebungen nicht gezählt werden. Eine schnellere Umsetzung ist kaum möglich, da Ideen auch noch beschlossen und in das Backlog für das nächste Release aufgenommen werden müssen. In den meisten IT-Projekten – schätzungsweise etwa 80 Prozent – ist das für unternehmenskritische Softwarelösungen heute noch Standard. Dies gilt, obwohl nahezu alle angeben, agile Methoden wie Scrum oder Kanban einzusetzen.

Das Leuchtturmprojekt ist agiler und hat meist einen hohen Time-to-Market-Druck. Sie machen circa 15 Prozent aller Projekte aus. Dort schafft man 10 bis 12 Releases der Software pro Jahr und eine Idee kann bestenfalls in 2 bis 3 Monaten umgesetzt werden. Schneller geht dies nicht, da der Backlog für ein gestartetes Release in der Regel fix ist. Dauern die einzelnen Releases deutlich länger als einen Monat, ist circa ein Release pro Monat möglich, indem an 2 bis 3 parallel gearbeitet wird. Dieses Vorgehen ist meist dem Zeitdruck geschuldet und damit „aus der Not geboren". Hinzu kommen auch hier je nach Bedarf simple Updates, um Fehler zu beheben.

Das Ausnahmeprojekt ist agilissimo. Dies betrifft höchstens 5 Prozent der Projekte. Man findet sie nur in wenigen Unternehmen, insbesondere im Online-Handel. Diese Ausnahmeprojekte schaffen es, Ideen in 1 bis 2 Wochen komplett umzusetzen.

Nicht jedes Projekt muss agilissimo werden, wie auch Diskussionen um bimodale IT [GartnerBimodal] zeigen. Im Online-Handel ist agilissimo erfolgsentscheidend, während für die Umsetzung von neuen Gesetzen im öffentlichen Bereich meist genügend Vorlaufzeit besteht. Trotzdem kennt der Autor niemanden, der nicht dennoch agiler werden möchte. Daher lohnt eine Analyse anhand der folgenden Punkte, was notwendig ist, um den nächsten Schritt hinzu agilissimo zu machen.

Softwareentwicklung 4.0

Das Erfolgsrezept von Projekten bei OTTO, Zalando und Netflix, die sich agilissimo nennen dürfen, ist, dass sie alle sehr konsequent auf vier Bausteine und deren Integration (untereinander) setzen (**siehe Abbildung 2**):

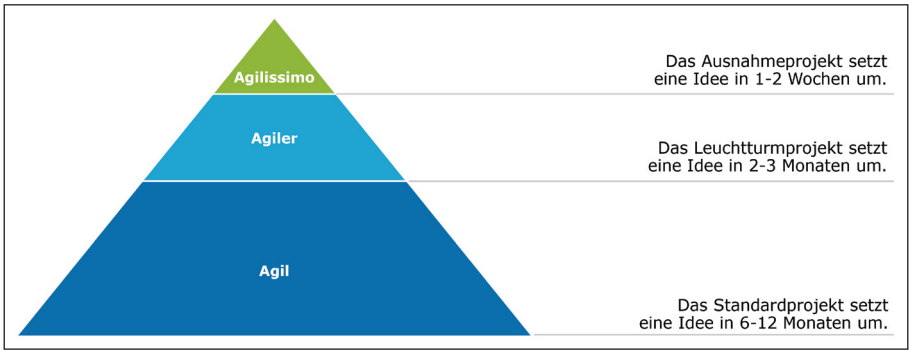

Abb. 1: Die agile Reifegrad-Pyramide

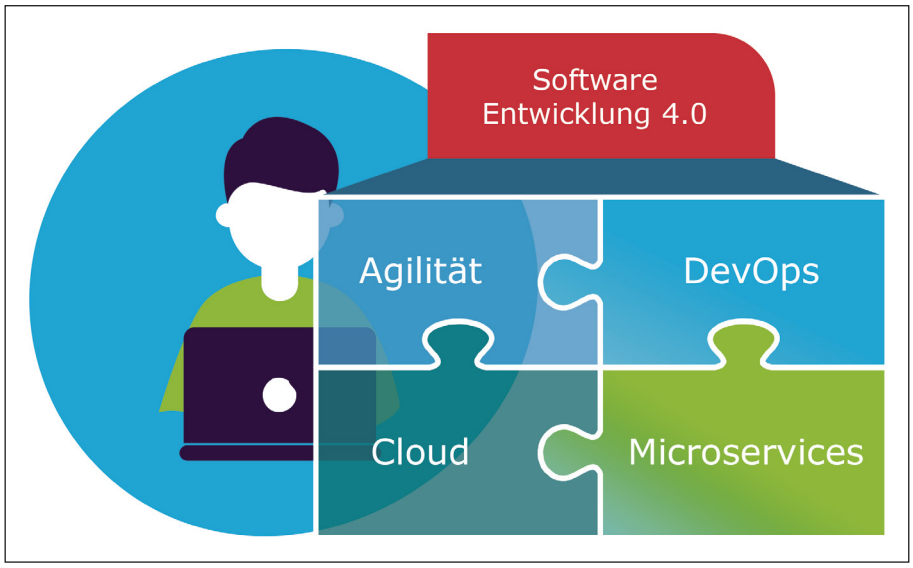

Abb. 2: Softwareentwicklung 4.0 = Agilität + DevOps + Microservices + Cloud

- agile Vorgehensmethoden, Prozesse und Organisationen,
- DevOps mit vollständiger Automatisierung und Auflösung der Mauer zwischen Entwicklung und Betrieb,
- Microservices und weitere leichtgewichtige Architekturen,
- Einsatz von flexiblen Cloud-Infrastrukturen.

Die Kombination dieser vier Bausteine führt zur vierten Entwicklungsstufe der Projektvorgehensmethoden (**siehe Abbildung 3**): die Softwareentwicklung 4.0. Sie ist nach dem klassischen Wasserfall (1.0), schwergewichtigen Modellen wie RUP oder V-Modell XT (2.0) sowie den agilen Vorgehensmethoden wie Scrum und Kanban gemäß des agilen Manifests von 2001 (3.0) seit circa 2015 – zumindest nach Ansicht des Autors – die Antwort auf die Frage „Was kommt nach Scrum und Kanban?“.
Softwareentwicklung 4.0 bedeutet, dass alle vier Bausteine nicht isoliert sind, sondern perfekt ineinandergreifen müssen. Agilissimo kann darüber hinaus nur werden, wer in allen vier Disziplinen den höchsten Reifegrad in der Umsetzung der „Best Practices“ hat, die im Folgenden beschrieben werden (**siehe Abbildung 4**).

Agilität hat den Reifegrad agilissimo

Konsequentes agiles Vorgehen bedeutet mehr als die Einführung von Scrum oder Kanban. Zumindest sollten folgende Empfehlungen umgesetzt werden:

- *Agile Vorgehensmethoden:* An erster Stelle steht, dass man agile Vorgehensmethoden wie Scrum, Kanban oder SAFe (Scaled Agile Framework) einsetzt.
- *EIN Team mit Entscheidungskompetenz:* Notwendig sind außerdem Organisationsänderungen, die zu kurzen Entscheidungswegen führen. Wer nur zwei Wochen für die Ideenumsetzung hat, kann nicht Tage oder Monate auf Entscheidungen warten. Deshalb müssen alle am Projekt beteiligten Bereiche, und zwar Fachbereich, Entwicklung, Test und Betrieb, im Projektteam mit Entscheidungskompetenz vertreten sein: Beispielsweise trifft der Product Owner die fachlichen Entscheidungen, die Entwickler entscheiden über die technische Umsetzung, die Tester verantworten die Qualitätssicherung und ein Betriebsexperte trifft Betriebsentscheidungen. Diese enge Zusammenarbeit der beteiligten Bereiche als EIN Team nennt man auch *BizDevOps*.
- *Produktorientierung:* Empfehlenswert ist, dass Unternehmen sich von einer Projektorganisation hin zu einer Produktorientierung transformieren. Dabei steht das Produkt stets im Mittelpunkt und das Team bleibt von der Idee bis zum Produktivbetrieb dafür verantwortlich. Das schließt nicht aus, dass das Produktteam von Teams mit Querschnittsfunktionen, zum Beispiel für den 24/7-Support, unterstützt wird.
- *Management 3.0:* Um agilissimo zu sein, bedarf es nach Ansicht des Autors moderner Führungsmethoden, wie sie oft unter dem Schlagwort Management 3.0 propagiert werden. Dazu zählt das Vertrauen in die Mitarbeiter, von denen jeder einen Teil der Verantwortung übernimmt. Ein einzelner Architekt bestimmt und verantwortet damit nicht mehr alle technischen Entscheidungen, sondern tritt als Coach des Teams auf und entwickelt eine gemeinsame Vision. Ein „Product Owner“ überzeugt beispielsweise in seiner Rolle als Business-Architekt das Team von einer Produktversion. Der erfahrene Entwickler überzeugt als technischer Architekt sein Team wiederum von der optimalen technischen Architektur für das Produkt.

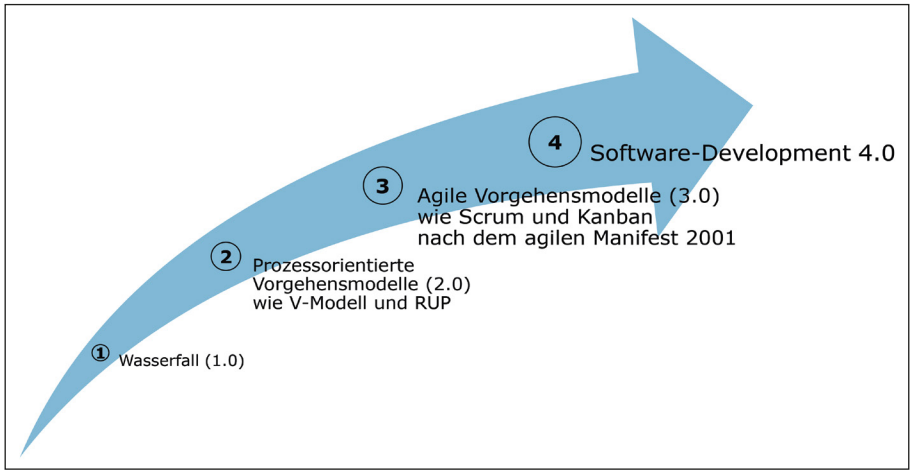

Abb. 3: Die vier Entwicklungsstufen der IT-Projektvorgehensmodelle

Als Beispiel lässt sich hier ein dem Autor bekanntes Großprojekt aus dem Automobilsektor heranziehen. Das Team besteht dort aus mehreren hundert Mitgliedern und mehrere Fachbereiche sind beteiligt. Die Umsetzungsgeschwindigkeit von Ideen wurde auch dort hinterfragt. Eine Analyse machte deutlich, wo wie viel Zeit verloren geht. Überraschenderweise wurde am meisten Zeit dafür benötigt, die Entscheidung zu treffen, dass die Idee überhaupt umgesetzt wird. Der Grund war, dass viele Stakeholder und Entscheidungsgremien zustimmen mussten. So mussten beispielsweise Budgets gewährt und Ideen zuungunsten anderer Vorschläge priorisiert werden, die dann nicht um-

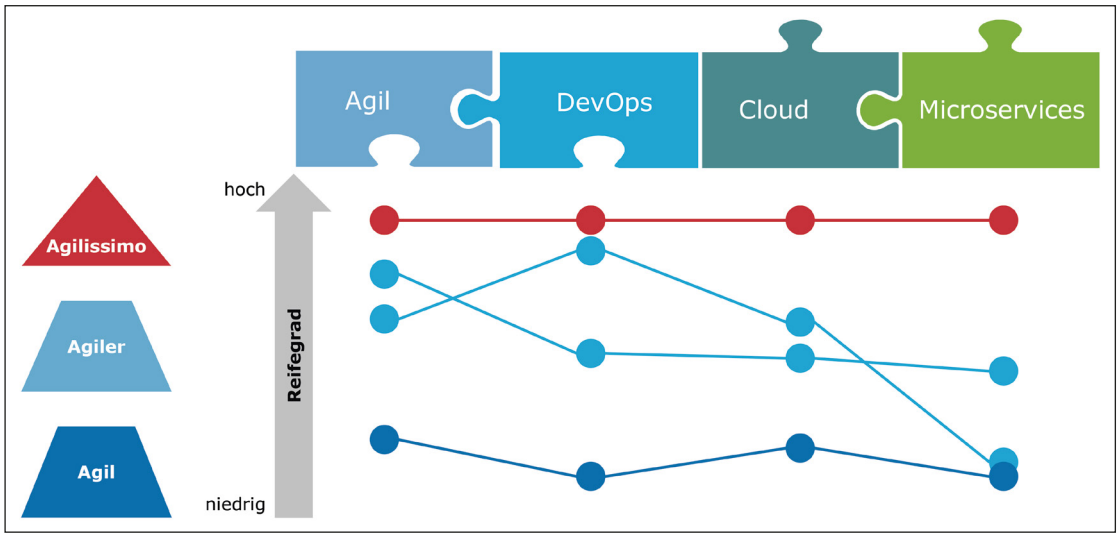

Abb. 4: Agilissimo Projekte setzen den höchsten Reifegrad in allen vier Disziplinen voraus!

gesetzt werden konnten. Der größte Hebel zur Umsetzungsbeschleunigung von Ideen war demnach *EIN Team mit Entscheidungskompetenz*.

DevOps und agilissimo

DevOps besitzt zwei wesentliche Merkmale. Eines ist die Zusammenarbeit von Entwicklung und Betrieb, für die organisatorische Veränderungen im Unternehmen unabdingbar sind. Dies ist in großen Unternehmen schwieriger umzusetzen als das zweite Merkmal: die vollständige Automatisierung. Sie bezieht sich auf die Prozesse zum Bauen, Installieren, Testen und Produktivsetzen der IT-Systeme. Für mehrtägige Abnahmetests ist bei „agilissimo“ keine Zeit mehr, wenn eine Idee in zwei Wochen umgesetzt sein soll. Daher dürfen keine Releases mehr existieren, vielmehr geht man kontinuierlich in Produktion, gemäß des Prinzips des „Continuous Deployment“ ([freecodecampCP], [Sha17]).

Beim „Continuous Deployment“ geht ein gecheckter Quellcode sofort in Produktion, sofern innerhalb der automatisierten Teststufen kein Fehler entdeckt wurde, der die Produktivsetzung verhindert. „Continuous Deployment“ setzt Vertrauen und einen hohen Reifegrad der Testautomatisierung voraus, der nur durch eine kontinuierliche Verbesserung zu erreichen ist. So genannte „Feature Toggles“ [FeatureFlags] stellen sicher, dass keine halbfertigen Features produktiv sichtbar werden.

„Continuous Deployment“ ist eine Fähigkeit, von der die meisten Unternehmen noch weit entfernt sind. Vorstufen auf der Reifegradskala für DevOps sind „Continuous Integration“ sowie „Continuous Delivery“, das zum ersten Mal im gleichnamigen Buch-Klassiker von Humble und Farley [Hum10] definiert wurde. Die Unterschiede zu „Continuous Integration“ und „Continuous Deployment“ werden in [freecodecampCP] aufschlussreich erläutert. Aus Sicht des Autors setzen fast alle Projekte zumindest eine „Continuous Integration“ um. Einige erzielen bereits ein „Continuous Delivery“, während „Continuous Deployment“ momentan eher die Ausnahme ist.

DevOps und Agilität sind wie Yin und Yang, da die Umsetzung von DevOps immer mit einer agilen Vorgehensweise einhergeht. Bei Capgemini sind daher die entsprechenden Kompetenzen in einem „Center of Excellence für Agile & DevOps“ konzentriert. Für häufig in Projekten eingesetzte Technologiestacks existieren außerdem vorkonfigurierte, automatisierte Werkzeugketten, mit denen DevOps-Experten in kürzester Zeit neue Projekte mit einer automatisierten Build-Deploy-Test-Pipeline ausstatten können. So lässt sich schnell der erste Schritt in Richtung „Continuous Deployment“ umsetzen.

Deswegen ist jedem Unternehmen zu empfehlen, die eingesetzte Technologievielfalt zu steuern und nicht nach Belieben der Entwickler zuzulassen. So profitiert man leichter von den Erfahrungen aus anderen Projekten und erzielt einen deutlich höheren Reifegrad.

Cloud und agilissimo

Microservices und voll automatisierte Teststufen können durch ihren Ressourcenbedarf schnell zum Kostentreiber werden. Eine Cloud-Infrastruktur, die ihren Namen verdient, macht „IT wie aus der Steckdose“ verfügbar, indem sie flexibel, schnell und automatisiert Ressourcen bereitstellt. Der Vergleich bezieht sich auch auf die Elastizität, um bedarfsgerecht und exakt nach Verbrauch abrechnen zu können (Stichwort „Pay per Use“). Ryan Shuttleworth hat dieses Elastizitätsprinzip in [Shu12-a] und [Shu12-b] dargestellt. Im Gegensatz dazu existieren in vielen unternehmenseigenen Rechenzentren nicht genügend freie Hardware-Ressourcen, um nur ansatzweise eine Elastizität wie die großen Cloud-Anbieter liefern zu können.

Mit einer elastischen Cloud werden Überlastsituationen und Ausfälle unmittelbar kompensiert und unzufriedene Kunden vermieden. Insgesamt lassen sich dank dieser Elastizität sogar die Kosten senken und trotzdem die Kundenzufriedenheit erhöhen.

Der wesentliche Unterschied zwischen privater und öffentlicher Cloud wird an diesem Praxisbeispiel deutlich: Der Betrieb für ein in mehreren Microservices neu entwickeltes, unternehmenskritisches System sollte in einer privaten Cloud betrieben werden. Bereits mehr als ein Jahr vor Inbetriebnahme sollte der Autor eine Einschätzung geben, wie viele virtuelle Maschinen mit wie viel RAM und CPU-Kapazität denn eingekauft werden müssten. Wie schwer belastbare Aussagen für ein neues, noch in der Entwicklung befindliches System zu machen sind, weiß jeder Architekt, der bereits für vergleichbare Systeme verantwortlich war. Es ging um circa 200 VMs mit 16 bis 64 GB RAM je nach Microservice und entsprechender CPU-Kapazität. Vom Prinzip „IT aus der Steckdose“ war diese Anfrage weit entfernt.

In Form von Konfigurationsdateien – Stichwort „Infrastructure as Code“ – können virtuelle Maschinen so eindeutig definiert werden, dass man exakt dieselbe Infrastrukturkonfiguration auf den verschiedenen Umgebungen – von Entwicklung über verschiedene Teststufen bis zur Produktion – sicherstellen kann. Das erspart böse Überraschungen aufgrund unterschiedlicher Konfigurationen und Ressourcen können dank „Infrastructure as Code“ bedarfsgerecht und automatisiert hochgefahren werden, etwa für Last- und Performanztests. Damit liefern Cloud-Infrastrukturen einen wichtigen Baustein für die notwendige Testautomatisierung von

Literatur & Links

[12FACT17] The Twelve Factor App, siehe: **https://12factor.net/**

[Bis15] K. Bishogo, Radical Agility with Autonomous Teams and Microservices, Zalando Techblog, 2015, siehe: **https://jobs.zalando.com/tech/blog/video-radical-agility-with-autonomous-teams-and-microservices/**

[ENT18] What is the Difference between Cloud-Ready and Cloud-Native?, siehe: **https://www.entando.com/page/en/what_is_the_difference_between_cloudready_and_cloudnative?contentId=BLG4585**

[FeatureFlags] The Hub for Feature Flag Driven Development, siehe: **https://featureflags.io/literature/**

[freecodecampCP] https://medium.com/free-code-camp/how-to-set-up-continuous-deployment-in-your-home-project-the-easy-way-41b84a467eed

[GartnerBimodal] https://www.gartner.com/it-glossary/bimodal/

[Hum10] J. Humble, D. Farrley, Continuous Delivery, Addison-Wesley, 2010

[NetflixBlog] https://medium.com/netflix-techblog

[OTTOBlog] https://www.otto.de/jobs/technology/techblog/

[OTTO16] G. Steinacker, Why Microservices, Otto, 20.3.2016, siehe: **https://www.otto.de/jobs/technology/techblog/artikel/why-microservices_2016-03-20.php**

[ScaledAgile] https://www.scaledagile.com/enterprise-solutions/enterprise-challenges/

[Sha17] M. Shahin, M. A. Babar, M. Zahedi, L. Zhu, Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges, in: ACM/IEEE Int. Symposium on Empirical Software Engineering and Measurement (ESEM), 2017

[Shu12-a] R. Shuttleworth, The Lean Cloud for Startups with AWS: Introduction & AWS Overview, Amazon Web Services, 2012

[Shu12-b] Ryan Shuttleworth, Journey Through the AWS Cloud; Building Powerful Web Applications, Nov. 2012, siehe: **https://www.slideshare.net/AmazonWebServices/jouney-buildingpowerfulwebapplications**

[SoAR13] 13th annual State of Agile Report, COLLABNET VERSIONONE, siehe: **https://stateofagile.versionone.com/**

[TH18] Janakirma MSV, 10 key attributes of cloud-native applications, The New Stack, 19.7.2018, siehe: **https://thenewstack.io/10-key-attributes-of-cloud-native-applications/**

[Til19] St. Tilkov, E. Wolff, Microservices ersetzen den Monolithen - Ameisenprinzip, in: IX, April 2019

der Entwicklung bis in die Produktion. Letztlich ermöglicht nur die Kombination aus elastischer Cloud und „Infrastructure as Code“ die für ein „Continuous Deployment“ notwendige Testautomatisierung und die damit erforderlichen Ressourcen für automatisierte Last- und Performance-Tests.

Der Betrieb in der Cloud stellt weiterhin Anforderungen an die Anwendungsentwicklung, wobei gemäß der Definition in [ENT18] „cloud-native“ Anwendungen entwickelt werden sollten. Die zehn Schlüsseleigenschaften in [TH18] oder die zwölf Faktoren in [12FACT17] sind ein guter Startpunkt hierfür. Eine Empfehlung in [TH18] ist der Einsatz loser gekoppelter Microservices, welche die nächste Voraussetzung für „agilissimo“ sind.

Microservices und agilissimo

Das Risiko, ein neues Release eines vergleichsweise kleinen Microservice produktiv zu setzen, ist deutlich geringer als bei großen und komplexen Deployment-Monolithen oder -Modulithen. Voraussetzung dafür ist, lose gekoppelte und damit voneinander unabhängig installierbare Microservices zu entwickeln. Dann lösen Fehler keine Fehlerlawine über abhängige Microservices aus. Microservices sind daher ein Architekturstil, mit dem sich schrittweise die vollständige Automatisierung bis zur Produktivsetzung mit weniger Risiken umsetzen lässt.

Testverfahren für einzelne isolierte Microservices sind zudem weniger aufwendig als Regressionstests für komplexe Deployment-Modulithen. In Verbindung mit einer verlässlichen Abhängigkeitsanalyse der Microservices untereinander können somit deutlich kleinere Testsuiten zielgenau angestoßen und Laufzeiten und Ressourcenverbrauch minimiert werden.

Der Autor ist seit 2015 überzeugt, dass Microservices für unternehmenskritische IT-Systeme der Architekturstil der Zukunft sind. Das gilt insbesondere für größere Microservices, sogenannte „Self-Contained Systems“ (SCS), wie sie beispielsweise in [OTTO16] und [Til19], Seiten 42 ff. dargestellt werden. Trotzdem muss auch heute nicht jeder Bereich in solchen IT-Systemen mit Hunderten oder sogar Tausenden von Bearbeiterjahren agilissimo sein. Deshalb bietet es sich an, Teilsysteme mit hohem „Time-To-Market“-Druck mit Microservices zu bauen, während für andere Teilsysteme gut strukturierte Modulithen die erste Wahl bleiben.

Entsprechend zeigt sich ein eindeutiger Trend, dass die größten existierenden IT-Systeme nicht mehr als EIN Modulith, sondern in mehreren Modulithen und Microservices gebaut werden. Insgesamt stehen Microservices stellvertretend für den Trend zu leichtgewichtigen Architekturen, die ein sehr agiles Vorgehen unterstützen. Andere Trends sind event-gesteuerte Architekturen mit Streaming-Tools wie Kafka oder leichtgewichtige Prozess- oder Fallsteuerungen durch Produkte wie Camunda, Activiti und Drools, die schwergewichtige Produkte rund um einen selbstständigen Prozess-Server ablösen.

Wechselwirkungen der Disziplinen

Unter der Annahme, dass „agilissimo“ als Reifegrad und die Umsetzung einer Idee innerhalb von zwei Wochen angestrebt wird, lässt sich die Wechselwirkung der vier Disziplinen agiles Vorgehen, DevOps, Cloud und Microservices aufzeigen: Das agile Vorgehen ermöglicht die kurzen Entscheidungswege, während DevOps beziehungsweise BizDevOps alle Entscheidungsträger einbezieht.

Da innerhalb von zwei Wochen von der Idee bis zur Produktion keine Zeit für aufwendige manuelle Tests bleibt, ist die vollständige Automatisierung der „Build-Deploy-Test-Run-Pipeline“ in allen Umgebungen unabdingbar – und zwar von der Entwicklung über die verschiedenen Teststufen bis zur Produktion. Diese Automatisierung ist wiederum nur mit Cloud-Eigenschaften wie „Infrastructure as Code“ bis auf die Ebene der Hardware-

Ressourcen zuverlässig möglich. Die Elastizität der Cloud ermöglicht zugleich ein günstiges Kosten- und Nutzenverhältnis.
Zahlreiche Microservices wären wiederum ohne einen hohen Automatisierungsgrad und die beschriebenen DevOps- und Cloud-Eigenschaften nicht zu akzeptablen Kosten als Ersatz für große Deployment-Modulithen zu betreiben. Welches Unternehmen möchte schon Hunderte von Microservices ständig manuell neu installieren und überwachen?
Und Microservices tragen schließlich dazu bei, die Risiken bei der Entwicklung hin zum „Continuous Deployment" zu verringern. Durch den reduzierten Testaufwand bei jeder Produktivsetzung lässt sich immer mit genügend Sicherheit testen. Der Kreis der vier Disziplinen der Softwareentwicklung 4.0 schließt sich, weil unabhängige Microservices die beste Voraussetzung zur Bildung kleiner agiler Teams sind, die möglichst unabhängig voneinander entscheiden, entwickeln, testen und produktiv setzen sollen.

Softwareentwicklung 4.0: Mehr als die Summe ihrer Teile

Dem aufmerksamen Leser wird klar sein, dass es bei Softwareentwicklung 4.0 nicht um die Entwicklung von kleinen überschaubaren Apps oder in Cloud-Plattformen angebotenen Services geht. Diese sind meist von überschaubarer Komplexität. Vielmehr adressiert Softwareentwicklung 4.0 die Entwicklung sehr umfangreicher unternehmenskritischer Systeme. Doch auch dort lässt sich die Transformation zu mehr Agilität schrittweise innerhalb der vier Disziplinen vorantreiben, sodass die Risiken jederzeit beherrschbar bleiben.
Aus der Beobachtung zahlreicher Projekte in vielen Unternehmen schließt der Autor, dass sich heute alle Unternehmen und öffentlichen Institutionen mehr oder weniger mit den vier Disziplinen der Softwareentwicklung 4.0 beschäftigen. Jedes Projekt oder Produkt besitzt allerdings in jeder Disziplin einen unterschiedlichen Reifegrad. Entsprechend gibt es kaum ein Projekt, dass sich agilissimo nennen darf: Während der eine bei der Cloud weiter ist, punktet der andere mit einem höheren Reifegrad bei DevOps. Konsequent auf Microservices setzen eher diejenigen, die beim Thema Agilität weit fortgeschritten sind und in ihren Projekten bereits an Grenzen der Architektur stoßen, um noch agiler zu werden.
So kann man zusammenfassen, dass es mit Softwareentwicklung 4.0 ähnlich wie mit der Agilität ist. Jeder beschäftigt sich damit, aber die wenigsten haben bereits einen sehr hohen Reifegrad erreicht. So schaffen es vorerst nur die wenigsten, „agilissimo" zu sein und eine Idee innerhalb von zwei Wochen trotz umfangreicher Change Requests in Produktion zu bringen. ||

Der Autor

Dr. Karl Prott
(karl.prott@capgemini.com)
verantwortet seit über 20 Jahren als IT-Architekt bei Capgemini die Umsetzung unternehmenskritischer IT-Systeme. Seit einiger Zeit unterstützt er als Architekt die zahlreichen Angebote und hat dabei die Bausteine von Softwareentwicklung 4.0 fest im Blick.