Container im Agilen Entwicklungsprozess

Was Sie als Führungskraft wissen müssen

Version Juni 2017

Über diesen Artikel

In viele Unternehmen stoßen die agilen Teams nach einiger Zeit an eine technische Grenze. Sie benötigen schnelles Feedback auf allen Ebenen, doch zahlreiche Faktoren stehen dem im Weg: Entscheidende Tests laufen nicht auf den Entwicklerrechnern, der Buildserver ist von verschiedenen Drittsystemen abhängig, die Fachabteilung hat kein Testsystem und das Produktivsystem wird nur selten aktualisiert.

Nicht an jeder Stelle erreicht das Team einhundertprozentige Automatisierung – das behindert, weil die manuelle Arbeit Feedback-Möglichkeiten reduziert oder Zeit von der Feature-Entwicklung nimmt. Und nicht an jeder Stelle erreicht das Team exakte Übereinstimmung. Man möchte Dev-Prod-Parity erreichen - die Übereinstimmung zwischen Entwicklungssystem und Produktionssystem. Nur so verhalten sich Produkt- und Entwicklungssystem identisch. Denn es ist enorm wichtig, an allen Stellen kurzfristig verlässliches Feedback zu bekommen:

Jede Feedback-Schleife spart bares Geld in der Entwicklung. Je kürzer die Schleife, umso wertvoller ist das Feedback.

Dev-Prod-Parity ist wichtig, um eine hohe Aussagekraft des Feedbacks sicherzustellen.

Die Einführung von Container-Technologie muss passend zum Ausbildungsstand des Entwicklungsteams und der IT-Mannschaft geschehen. Dieser Artikel verdeutlicht Ihnen als Führungskraft, was es bedeutet, Container-Technologie in Entwicklungs- projekten einzusetzen und was Sie tun müssen, um mit ihren Projekten erfolgreich zu sein. Sie werden lernen, dass Container deutlich mehr bedeuten als eine Form der Virtualisierung oder ein technischer Hype (Kapitel 1). Der Einsatz von Containern erschließt für den agilen Prozess Abgeschlossenheit, Verlässlichkeit, Geschwindigkeit und Reproduzierbarkeit. Sie werden verstehen, welchen Nutzen Sie für die Entwicklung daraus ziehen können (Kapitel 2).

Geschickt eingesetzt beschleunigt die Container-Technologie ihren agilen Entwicklungsprozess (Kapitel 3) und wird dem Entwicklungsteam helfen, Ihre Kunden zu begeistern.

Mit Container-Technologie können Sie mächtige und zugleich leichtgewichtige Praktiken zur Softwareentwicklung etablieren, indem sie von der aktuellen technischen Situation ausgehen und sie schrittweise verbessern.

Sie sollten in diesem Zusammenhang verstehen, welche Möglichkeiten Sie sich als Führungskraft erschließen und welche Ebenen Sie besser den Technikern überlassen (Kapitel 5).

Und nicht zuletzt werden Sie eine Vorstellung davon entwickeln, welche Rolle technische Beratung und Coaching für den Erfolg Ihrer Entwicklungsprojekte spielt (Kapitel 5).

Wir bei it-agile wenden seit 1999 agile Verfahren an und haben dabei gelernt, was funktioniert und was nicht. Heute versammeln wir das Wissen und die Erfahrung von 40 festangestellten Mitarbeitern mit zusammen mehreren hundert Jahren Erfahrung in agiler Entwicklung, modernen Arbeitsprozessen und zeitgemäßer Führung. Wir stellen Ihnen als Führungskraft mit diesem Artikel das Kondensat aus unserer Erfahrung zur Verfügung. Der Artikel hilft Ihnen dabei, die Einsatzgebiete der Technologie jenseits der rein technischen Aspekte zu verstehen.

Wir freuen uns darauf, Sie beim Technologie-Einsatz in ihren agilen Projekten zu unterstützen.

Zögern Sie nicht, mich oder meine Kollegen darauf anzusprechen.

Dr. Wolf-Gideon Bleek
Agile Engineering Evangelist bei it-agile

wgb@it-agile.de, Tel. 0175 522 63 44

1. Containerisierung ein kurzer Blick auf die Historie

Vor-Virtualisierungs-Zeit

In der Zeit vor der Verbreitung von Virtualisierung gab es unterschiedliche Bezugssysteme im professionellen Software-Entwicklungsprozess: der Entwicklungsrechner, der Buildserver, das Test-, das Staging-System und das Produktionssystem (nicht immer sind alle Systeme vorhanden).

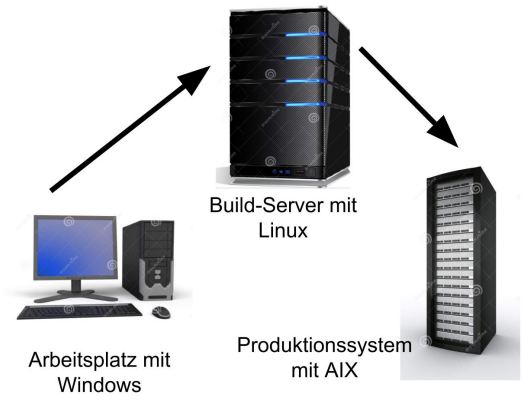

Abbildung: Beispiel für unterschiedliche Bezugssysteme

Im Allgemeinen bestand ein spürbarer technischer Unterschied zwischen den Systemen, der sich darüber ausdrückt, dass “works on my machine“ zu einem geflügelten Wort wurde. Die Systeme wurden überwiegend von unterschiedlichen Gruppen aufgesetzt (Entwicklungsrechner von den Entwicklern selbst, Produktionssysteme von den Administratoren der IT-Abteilung).

Die einzige Möglichkeit um Vergleichbarkeit und damit Sicherheit herzustellen war es, identische Systeme (Hard- und Software) in den vorhandenen Umgebungen einzusetzen. Häufig musste sich das Entwicklungsteam damit begnügen, dass es ein ähnliches System zu bestimmten Zeiten als Testsystem zur Verfügung bekam.

Selbst bei identischer Hardware blieb das Problem, dass die Installationen nicht 100% identisch waren und sehr genau geschaut werden musste, ob kleinere Versions-Unterschiede bestanden, die Auswirkungen hatten.

Virtualisierung

Mit dem Aufkommen von Virtualisierungslösungen [1] (VMware, VirtualBox etc.) entstand ein neuer Freiheitsgrad und es war möglich, die Unterschiede zu reduzieren. Bspw. konnten auf dem Windows-Entwicklungsrechner virtuelle Maschinen laufen, die dem Unix-Betriebssystem der Produktion entsprachen. Oder der Build-Server nutzte virtuelle Maschinen, um die Tests in einer produktionsähnlichen Umgebung auszuführen.

Dadurch konnte wertvolles Feedback direkt an den Entwicklerarbeitsplatz und direkt auf den Build-Server geholt werden. Die Ergebnisse wurden aussagekräftiger und schneller verfügbar.

Es blieb aber das Problem, dass virtuelle Maschinen schwergewichtig sind: sie benötigten jeweils ein vollständiges Betriebssystem, die Einrichtung war langwierig und der Ressourcenverbrauch groß ([8], Seite 5).

Configuration Management

Der nächste logische Schritt war die automatische Installation und die automatisierte Konfiguration von Software in der virtualisierten Umgebung [2].

Werkzeuge wie Vagrant, Chef, cfEngine, Puppet oder Ansible ermöglichten es, das Installieren und Konfigurieren von Hard-

ware, Betriebssystemen und Software automatisch durchzuführen. Das war dringend notwendig, denn nur in den seltensten Fällen war das Aufsetzen der VM mit wenigen Schritten getan. Alle notwendigen Schritte wurden in einem Skript aufgeschrieben, das automatisiert ausgeführt werden konnte. Außerdem stieg die Anzahl der zu betreuenden Systeme.

Wesentliche Vorteile neben der Massenhandhabung waren Dokumentation und Reproduzierbarkeit. Das Installations- und Konfigurations-Script wurde zu einem weiteren Artefakt im Software-Entwicklungsprozess. Es erlaubte es, an jedem beliebigen Ort eine VM entsprechend der Spezifikation aufzusetzen.

Der Gewinn für den agilen Prozess lag darin, implizites Wissen zu explizieren und Wissen über das Aufsetzen im Entwicklungsteam zu verteilen. Die Konfiguration wurde so ebenfalls testbar und das Feedback wiederum beschleunigt.

Wurde die virtuelle Maschine für die Tests jedes Mal neu erzeugt, wenn ein Build erfolgreich durchgelaufen war, dann war dies gleichzeitig der Test dafür, ob das aktuelle System mit dem vorliegenden Skript korrekt aufgesetzt werden konnte.

Container

Heute sind Container Stand der Technik: Technologien, wie bspw. Docker, fahren einen deutlich leichtgewichtigeren Ansatz. Docker legt read-only Dateisysteme geschickt übereinander und erzeugt so ein Schichtsystem, in dem nur die oberste Schicht schreibbar ist. Verschiedene Systeme nutzen dieselben Schichten, existieren aber trotzdem getrennt voneinander [3].

Beispiel: Das Betriebssystem Ubuntu 16.04 wird in der Schicht A installiert. In einer darüber liegenden Schicht B1 wird der Webserver installiert und in einer unabhängigen Schicht B2 wird eine Datenbank installiert. Aus der Perspektive des Webservers sieht es wie ein Webserver-Installation auf Ubuntu aus und aus der Perspektive der Datenbank sieht es wie eine Datenbank-Installation auf Ubuntu aus. Die Schicht A kann beliebig häufig wiederverwendet werden, da alle Anpassungen in unabhängigen Schichten hinterlegt werden.

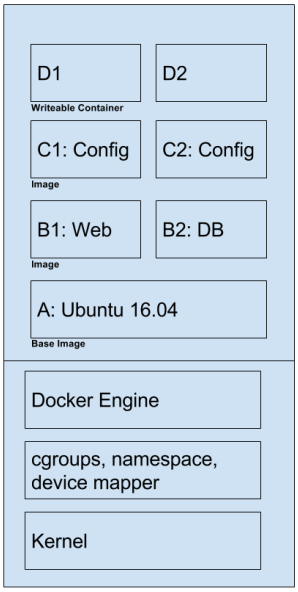

Abbildung: Schichten bei Docker

Wesentlicher Vorteil dieses Verfahren ist, dass die Container leichtgewichtig existieren und deren Beschreibung standardisiert ist. Eine Vielzahl von Containern und Container- Beschreibungen ist öffentlich verfügbar. Es wird somit sehr einfach, auf vorhandenen Containern aufzubauen. Für die gängigen Applikationen (wie z.B. Datenbanken und Webserver) kann der jeweilige Entwickler die eigene Konfiguration ergänzen und das System ist mit wenigen Zeilen konfiguriert und lauffähig.

Der Gewinn für den agilen Prozess liegt

darin, dass die Beschreibung des Containers vollständig Teil des Entwicklungsprozesses ist und damit kleinste Einheiten separat konfektioniert werden können. Gleichzeitig ist die Beschreibung universell einsetzbar, d.h. auf jedem der Bezugssysteme kann dasselbe Script ausgeführt werden, um ein identisch konfiguriertes System zu erstellen.

Container-Einsatz bedeutet Umdenken

Obwohl Container eine Form der Virtualisierung sind, erschließen sie viel mehr also nur die offensichtliche Möglichkeit, ein Gastsystem auf einem Rechner auszuführen. Insbesondere die Trennung von Installation und Konfiguration sowie die Möglichkeit Gruppen von Containern als neue Einheiten zu betrachten, können genutzt werden, Systeme anders zu schneiden, Verantwortlichkeiten zu klären und Entwicklungsprozesse zu agilisieren.

Der Einsatz von Containern ohne Anpassung der Architektur, der Verantwortlichkeiten und der Prozesse bringt allerdings nur marginale Verbesserungen.

„Do One Thing and Do It Well“

Jeder Container sollte dazu genutzt werden, genau eine Sache umzusetzen und dies mit der am besten dafür geeigneten Technologie.

2. Eine Muster-Architektur für Container

Die Leichtgewichtigkeit der Container legt es nahe, für jede Zuständigkeit einen separaten Container zu nehmen. Bei einer Webanwendung sind dies beispielsweise ein Container für den Webserver, ein Container für den Application-Server und ein Container für die Datenbank. Hinzu kommen ggf. Container für Benutzer-Verwaltung, Caching, Queuing und andere Dienste.

Es ist sinnvoll separate Container zu verwenden, weil es *Separation of Concerns* unterstützt und insbesondere Update-Kaskaden vermeidet. Muss für eine Applikation ein neueres Betriebssystem oder eine andere Bibliothek installiert werden, bleiben alle anderen unberührt. Früher ergab sich häufig die Situation, dass das Update einer Komponente Veränderung an einer Vielzahl von Systembestandteilen bedeutete, die unbeabsichtigt “eingekauft” wurden.

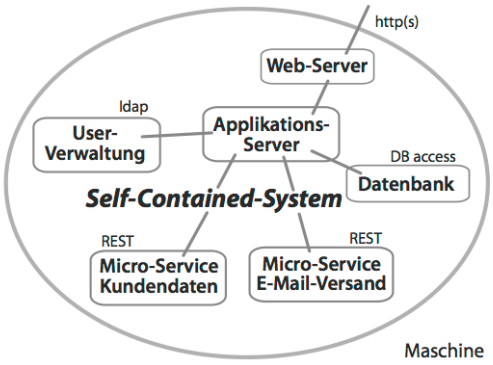

Abbildung: So könnte eine Container-Zelle zusammengestellt werden

Mikro-Ebene

Container sind die Einheiten der Mikro-Ebene. Das Software-Entwicklungs-Team gestaltet auf dieser Ebene die Architektur eines einzelnen Services. Da jeder Container in einem abgeschlossenen System läuft können hier für jeden Service unterschiedliche Technologien eingesetzt wer-

den. Das Entwicklungsteam kann z.B. die Programmiersprache frei wählen und so die produktivsten Werkzeuge einsetzen.

Makro-Ebene

Zellen sind aus Containern zusammengesetzt und Einheiten der Makro-Ebene. Die Systemlandschaft wird aus mehreren Zellen bestehen. Das Software-Entwicklungs-Team gestaltet auf dieser Ebene die Architektur eines Self-Contained-Systems [11] im Sinne einer Modul-Architektur aus.

Die nach außen veröffentlichte Schnittstelle wird von anderen benutzt, die zu Vertragspartnern werden. Änderungen an der Schnittstelle erfordern Absprachen über die Teamgrenze hinweg.

Vorteile

Wir gewinnen Unabhängigkeit. Die höhere Entwicklungsgeschwindigkeit überwiegt als Vorteil gegenüber den redundanten Maschinen.

Die Zusammenstellung verschiedener Container zu einem lauffähigen System kann man als Zelle oder Self-contained-System bezeichnen und als Musterarchitektur formulieren [4]. Die Zelle ist für sich abgeschlossen und liefert eine vollständige Dienstleistung (hier z.B. dynamische Webseiten). Die Zelle ist unabhängig, d.h. sie funktioniert ohne weitere Dienste.

Wir erreichen Isolation. Die Vorteile des reduzierten Abstimmungsaufwands überwiegen gegenüber ggf. redundant vorhandenen Daten.

Jeder einzelne Container wird in seinem Aufbau beschrieben und auch die Zusammenstellung der Container wird beschrieben. Die Beschreibungen sind weitere Artefakte im Software-Entwicklungsprozess. Damit unterziehen sie sich den üblichen Praktiken (Versionieren, Testen usw.).

Wir gewinnen Nachvollziehbarkeit. Die Container-Beschreibung machen alle Schritte zur Installation nachvollziehbar. Der deswegen entstehende Aufwand für die Entwickler ist durch die enormen Vorteile aufgewogen.

Ausgehend von einer Zelle kann ich Umgebungen im Entwicklungsprozess entsprechend gestalten. Auf dem Entwicklungsrechner kann ich ein Exemplar laufen haben, um direkt meine Handlungen zu prüfen. Auf dem Continuous-Integration-Rechner kann ein Exemplar erzeugt werden, um die Container-Beschreibungen und die Zusammenstellung zu testen. Diese und weitere Instanzen können für automatisierte Tests verwendet werden. Auf einer eventuell vorhandenen Staging-Umgebung, kann regelmäßig eine neue Instanz erzeugt werden, um einer Testgruppe von Anwendern Einblick in die neue Version zu geben. Weitere Möglichkeiten sind Last- und Performance-Tests, Integrationstest usw. Und natürlich kann das Produktionssystem mit einer aktualisierten Zelle bedient werden.

Wir gewinnen schnelles Feedback. Die Steigerung der Softwarequalität durch kurzfristiges Feedback macht den Aufwand für die vielen Exemplare des Systems wett.

Technisch betrachtet haben Zellen eine innere und eine äußere Schnittstelle. Über die innere Schnittstelle wird geklärt, wie die Container miteinander sprechen. Auf welchen Kommunikationskanälen unterhalten sich die verschiedenen Container – was offenbart ein Container den anderen Mitgliedern der Zelle. Über die äußere Schnittstelle wird geklärt, mit welchem Kanal die Zelle nach Außen kommuniziert.

Das ist eine wichtige Betrachtung, denn nur für die äußere Schnittstelle müssen sich größere Organisationen, in denen es mehrere Software-entwickelnde Teams gibt, abstimmen.

Diese äußere Schnittstelle sollte nach dem REST-Paradigma [9] gestaltet werden. Technisch wird dann häufig ein Aufruf im JSON-Format erwartet. Für Webbasierte Systeme können auch HTML-Komponenten geliefert werden.

3. Neues Potential

Unbestritten eröffnet sich durch die Container-Technologie Potential in Bezug auf technische Verteilung, Maschinen-Auslastung und einige andere Aspekte. Diese stehen allerdings nicht im Fokus dieses Artikels.

Von einem agilen Standpunkt aus betrachtet wollen wir die Vorteile für die Organisationsentwicklung und den Software-Entwicklungsprozess herausstellen.

Für die Organisation

Der Feedback-Prozess in der Organisation wird beschleunigt. Entwickler bekommen schneller Rückmeldung zu ihrem Arbeitsergebnis: sei es lokal, auf dem Build-Server oder durch automatisierte Testsysteme. Gleichzeitig können Product-Owner und Kunden früher in den Genuss eines lauffähigen Prototyps kommen.

Die Verantwortlichkeiten sind klar geregelt. Sie brauchen keine Verantwortungskette, da ein Entwicklungs-Team alle Container eigenverantwortlich betreibt, die notwendig sind, um seine Software zu nutzen. Jedes Team übernimmt die Gesamtverantwortung für eine Zelle.

Abhängigkeiten werden entschärft, indem alle notwendigen Bestandteile in einer Zelle zusammengefasst werden. Jedes Team betreibt eine Kopie von anderen Systemkomponenten und entkoppelt sich darüber. Das reduziert Abstimmungsbedarf zwischen Teams oder Abteilungen und ermöglicht es, mit Querbezügen entspannter umzugehen.

Für die Entwicklung

Insgesamt bieten mir Zellen von Containern ein neues Potential. Ich kann die Möglichkeit nutzen, zwei verschiedene Versionen einer Zelle gegeneinander zu vergleichen. Dies verwendet man z.B. für Variantentests oder im Bereich der Continuous Delivery.

Beispiel: Variantentests

Mit Variantentests bezeichnet man das systematische Testen von zwei oder mehr Varianten einer Dienstleistung oder eines Produkts im direkten Vergleich gegenüber den Kunden [5]. Online-Shops spielen beispielsweise mehrere Varianten einer Produktseite aus und untersuchen Unterschiede im Kaufverhalten. Die Hälfte der Nutzer bekommt Variante A, die andere Hälfte Variante B. Nach einem gewählten Zeitraum vergleicht man den Umsatz oder andere Indikatoren.

Um dies mit Zellen umzusetzen reicht es häufig aus, Zellen mit unterschiedlichen Versionen zu erzeugen und die Aufrufe der Nutzer auf die Zellen aufzuteilen.

Beispiel: Continuous Delivery

Continuous Delivery [6] ist eine Weiterentwicklung des Continuous-Integration-Ansatzes [7], bei dem es darum geht, ein Feature vollständig bis in Produktion zu bringen bevor die Entwicklung des nächs-

ten beginnt. Der Continuous-Delivery-Ansatz macht die Feature-Entwickler dafür verantwortlich, das neue Stück Software in Produktion zu betreiben.

Continuous Delivery führt dazu, dass die Entwickler:

- lernen, dass ein Feature nur dann fertig ist, wenn es genutzt werden kann.
- verstehen, welche Formen des Logging, Monitoring und Alerting in Produktion notwendig ist.
- immer nur an einem Feature zur Zeit arbeiten

Für das Unternehmen/die Kundschaft bietet es außerdem folgende Vorteile:

- neue Funktionalitäten sind schnell verfügbar.
- Investitionen erwirtschaften frühzeitig eine Rendite.
- Annahmen können kurzfristig überprüft werden.

Skalierung, Lastverteilung, Standortwechsel

Container bieten für die Entwicklung einfache Möglichkeiten, um Software für größere Nutzergruppen zu skalieren. Container können dieselbe Maschine effizienter ausnutzen und bei Bedarf auf verschiedene Maschinen verteilt werden.

Für die Lastverteilung werden mehrere Exemplare eines Containers hinter einen Load-Balancer gehängt. Dazu muss nur ein Load-Balancer als Schnittstelle nach außen in die Zelle eingeführt werden.

Ist ein Standortwechsel notwendig oder sollen die Antwortzeiten verbessert werden, kann die Container-Zelle in einem anderen Rechenzentrum erzeugt werden. Der Aufwand dazu ist gering, da die Beschreibung zur Erzeugung bereits existiert und getestet ist.

4. Pro und Contra verstehen

Der Einsatz von Containern in der Entwicklung fördert agile Praktiken und damit kürzere Feedback-Zyklen. Trotzdem gibt es nicht nur Vorteile und Container können nicht ausnahmslos empfohlen werden. Hier wollen wir kurz und knapp auf die Nach- und Vorteile eingehen.

Nachteile beim Einsatz von Containern

Container erfordern erfahrene Entwickler. Je unerfahrener Ihr Team ist, umso eher raten wir von Containern ab.

Der Schnitt von Software, sodass sie eine angemessene Aufteilung auf Container ermöglicht, erfordert Erfahrung und Augenmaß.

Container erfordern, dass sich Entwickler mit ihrer technischen Plattform stärker auseinandersetzen. Hierzu müssen die Entwickler bereit sein. Sie müssen das angemessene Betriebssystem auswählen und ggf. konfigurieren, sie müssen z.B. den Application-Server auswählen, sie müssen ihren persistenten Datenspeicher bestimmen. Das alles erschließt neue Freiheitsgerade, erfordert aber auch mehr Auseinandersetzung mit der Materie. Hier treten häufig Widerstände auf.

Vorteile durch den Container-Einsatz

Container verbessern die Rückkopplung im agilen Entwicklungsprozess stark. In der Entwicklung, beim automatisierten Testen, in der Diskussion mit Product Owner oder Auftraggeber und bei der Li-

vestellung ist die Wirkung deutlich.

Container ermöglichen kleine, klar abgegrenzte Systeme, die helfen einen Microservice-Architektur umzusetzen. Für den Entwicklungsprozess entstehen unabhängige Module.

Mithilfe von Zellen (Self-Contained-Systems) werden Entwicklungsteams von anderen Entwicklungsprozessen unabhängig. Ergebnisse können mit weniger Abstimmungsaufwand geliefert werden.

Container tragen Verantwortung in das Team, indem sie alle Elemente in die Hand des Teams legen. Sie entkräften ein für allemal die Ausrede “da müssen wir andere bitten, für uns etwas zu tun”. Damit beseitigen sie eine wichtige Behinderung von schnellem Feedback.

5. Unterstützung durch it-agile

Unsere Mitarbeiter arbeiten auf verschiedenen Positionen in Entwicklungsprojekten mit. Wir beherrschen das notwendige Handwerkszeug, um technische Prozesse anzupassen und Automatisierung voran zu treiben. Durch die umfangreiche Erfahrung zu agilen Prozessen können wir die richtige technische Unterstützung wohldosiert in Entwicklungsprojekte einbringen.

Bei unserem Einsatz zielen wir darauf ab, ihr Team zu befähigen, aus eigener Kraft die technischen Herausforderungen zu stemmen. D.h. wir werden nicht die fertige Lösung für ihr Team bauen, sondern mit dem Team daran arbeiten, die fehlenden Qualifikationen aufzubauen, um eine container-basierte Lösung selbständig zu entwickeln.

Wir freuen uns, wenn Sie unsere Unterstützung in Anspruch nehmen. Unsere festangestellten Berater kumuliert mehrere hundert Jahre Erfahrung mit agilen Techniken: Wir haben kleinen Unternehmen mit nur einem Team geholfen, agiler zu werden und Transitionen mit mehreren tausend Mitarbeitern begleitet (wir stellen Ihnen konkrete Fallbeispiele gerne im persönlichen Gespräch vor).

Dabei beschränkt sich unser Unterstützungsangebot nicht auf die IT oder einzelne Teams. Um den größten Vorteil aus agilen Ansätzen zu ziehen, sollte die gesamte Wertschöpfung betrachtet und ganzheitlich optimiert werden. Wir beziehen daher die verschiedenen Facetten des Unternehmens und der Wertschöpfung in unsere Beratung und unser Coaching mit ein.

Nehmen Sie gerne Kontakt mit uns auf:

Dr. Wolf-Gideon Bleek

Agile Engineering Evangelist

Tel. 0175/522 63 44

wgb@it-agile de

http://www.it-agile.de

6. Referenzen

1. Virtualisierung https://en.wikipedia.org/wiki/Virtualization
2. Configuration Management https://en.wikipedia.org/wiki/Vagrant_(software)
3. Container https://en.wikipedia.org/wiki/Docker_(software)

4. Musterarchitektur https://de.wikipedia.org/wiki/Architekturmuster
5. AB-Testing https://en.wikipedia.org/wiki/A/B_testing
6. Continuous Delivery https://en.wikipedia.org/wiki/Continuous_delivery
7. Continuous Integration https://en.wikipedia.org/wiki/Continuous_integration
8. Hashimoto, Mitchell: Vagrant – Up and Running
9. REST-Paradigma: https://de.wikipedia.org/wiki/Representational_State_Transfer
10. 10 things to avoid in docker container: https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/
11. InfoQ „Self Contained Systems (SCS): Microservices Done Right“, Eberhard Wolff https://www.infoq.com/articles/scs-microservices-done-right

Danksagung

Vielen Dank an Urs Reupke, Volker Machon und das Launch-Team für die Rückmeldung zu diesem Dokument.