Information Security

DevOps und klassisches IT Service Management

Am Beispiel von OpenStack

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

eingereicht von

Dominik Böckle
is161501

im Rahmen des
Studiengangs Information Security an der Fachhochschule St. Pölten

Betreuung
Betreuer: Dipl.-Ing. Gabor Österreicher, BSc
Mitwirkung: Dipl.-Ing. Dipl.-Ing. Christoph Lang-Muhr, BSc

St. Pölten, 18. Jänner 2019

(Unterschrift Autor/Autorin) (Unterschrift Betreuer/Betreuerin)

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Der Studierende/Absolvent räumt der FH St. Pölten das Recht ein, die Diplomarbeit für Lehre- und Forschungstätigkeiten zu verwenden und damit zu werben (z.B. bei der Projektevernissage, in Publikationen, auf der Homepage), wobei der Absolvent als Urheber zu nennen ist. Jegliche kommerzielle Verwertung/Nutzung bedarf einer weiteren Vereinbarung zwischen dem Studierenden/Absolventen und der FH St. Pölten.

St. Pölten, 18. Jänner 2019

(Unterschrift Autor/Autorin)

Zusammenfassung

ITIL (IT-Infrastructure Library) als Best-Practices-Sammlung unterstützt IT-Serviceprovider im IT Service Management bei der Konzeption, Steuerung und Optimierung, um IT Services zu planen, zu entwickeln und einzuführen. Eine Methode, um Softwareprojekte umzusetzen, ist ITIL hingegen nicht.
Dafür kommen agile Methoden zur Softwareentwicklung zum Einsatz, die sich Ende der 1990er Jahre als Nachfolger von Vorgehensmodellen, wie z.B. dem Wasserfallmodell, etabliert haben. Agile Methoden haben oft einen inkrementellen und iterativen Ansatz, um der Kundin bzw. dem Kunden möglichst schnell erste Teilergebnisse präsentieren zu können. Dabei besteht das Ziel darin, möglichst früh im Entwicklungsprozess Feedback zu bekommen.
Im Bereich der Softwareentwicklung haben unter anderem die Cloud-Technologien und Konzepte wie Continuous Deployment neue Möglichkeiten mit sich gebracht. Bei letzterem werden Änderungen im Source Code vollautomatisch über eine Deployment Pipeline bis in die Produktion übertragen. Die Schritte Build, Test und Deploy des Softwareauslieferungsprozesses sind dabei vollständig automatisiert.
Dadurch haben sich auch die Vorgehensmodelle weiterentwickelt und es entstand der DevOps-Ansatz, bei dem Development und Operations zusammengeführt werden. Dies hat durchaus starke Auswirkungen auf die Teamzusammenstellung und prägt eine eigene Kultur. DevOps umfasst dabei agile Methoden, wie z.B. Scrum, ein schlankes Management, wie z.B. Kanban, und IT Service Management basierend auf dem ITIL Framework.
DevOps und ITIL lassen sich demnach auch gut kombinieren. Es sind dabei aber speziell in den Prozessen Change Management, Release and Deployment Management, Service Asset and Configuration Management sowie Service Validation and Testing einige Besonderheiten zu beachten. So müssen etliche Schritte automatisiert durchführbar sein.
OpenStack bietet in diesem Zusammenhang als Cloud-Plattform eine gute Basis, um automatisiert standardisierte Infrastruktur-Ressourcen über eine Schnittstelle zu provisionieren. Dies ist für DevOps und für die damit einhergehende Deployment Pipeline notwendig.

Abstract

ITIL (IT Infrastructure Library) can support IT service providers in designing, controlling and optimizing their IT service management. But ITIL is not an approach for developing software.
This is where agile software development methods come to the fore. They became established at the end of the 1990s as the successor to approaches such as the water fall method. Agile methods are the better choice for software development. They often take an incremental and iterative approach in order to be able to present first partial results to the customer as quickly as possible. The developers thus get feedback at a very early stage of the development process.
From the perspective of software developers, further opportunities arise from technologies such as the cloud and strategies such as continuous deployment. Here changes in the source code are automatically transferred via a deployment pipeline to production. The Build, Test and Deploy steps of the software delivery process are completely automated.
Procedural models thus evolved, leading to the DevOps approach. The underlying idea is that of merging development with operations. This has a strong impact on the team composition and shapes a separate culture. DevOps groups agile methods such as scrum, lean management such as kanban and IT service management based on the ITIL framework.
DevOps and ITIL can therefore also be combined well. However, there are some special features to note in the change management process, the release and deployment management process, the service asset and configuration management process, and the service validation and testing process. Thus, several steps must be automated.
In this context, OpenStack as a cloud platform offers a good basis for automatically provisioning standardized infrastructure resources via an interface. This is essential for DevOps and the associated deployment pipeline.

Inhaltsverzeichnis

1 EINLEITUNG 7

1.1 MOTIVATION 7

1.2 FORSCHUNGSFRAGE 7

2 KLASSISCHES IT SERVICE MANAGEMENT MIT ITIL 8

2.1 SERVICE STRATEGY 9

2.2 SERVICE DESIGN 11

2.3 SERVICE TRANSITION 13

2.4 SERVICE OPERATION 18

2.5 CONTINUAL SERVICE IMPROVEMENT 19

3 AGILE METHODEN 20

3.1 SCRUM 21

3.2 KANBAN 23

3.3 SCRUM VS. KANBAN 25

3.4 AGILE TECHNIKEN 26

4 GEÄNDERTE ANFORDERUNGEN AN DEN IT-BETRIEB DURCH CLOUD-TECHNOLOGIEN 27

4.1 CLOUD COMPUTING 28

4.2 MICROSERVICES UND CONTAINER 31

4.3 OPENSTACK 35

5 DEVOPS 39

5.1 DEVOPS-KULTUR 40

5.2 DEVOPS LIFE CYCLE 44

5.3 SOFTWAREAUSLIEFERUNGSPROZESS 46

5.3.1 Continuous Integration 47

5.3.2 Continuous Delivery 49

5.3.3 Continuous Deployment 51

5.4 AUTOMATION UND INFRASTRUCTURE AS CODE 51

6 IT SERVICE MANAGEMENT UND DEVOPS 52

6.1 DEVOPS UND ITIL SERVICE STRATEGY 53

6.2 DEVOPS UND ITIL SERVICE DESIGN 53

6.3 DEVOPS UND ITIL SERVICE TRANSITION 53

6.4 DEVOPS UND ITIL SERVICE OPERATION 55

6.5 DevOps und ITIL Continual Service Improvement (CSI) 55
6.6 Zusammenfassung 55

7 USE CASE 56

7.1 DevOps-Phase Plan 59
7.2 DevOps-Phase Develop 61
7.3 DevOps-Phase Test 63
7.4 DevOps-Phase Release 63
7.5 DevOps-Phase Operate 64

8 CONCLUSIO 65

ABBILDUNGSVERZEICHNIS 66

TABELLENVERZEICHNIS 67

LITERATURVERZEICHNIS 68

1 Einleitung

Die vorliegende Arbeit repräsentiert eine theoretische Analyse über die Themen DevOps, ITIL (IT-Infrastructure Library) und OpenStack. ITIL als Sammlung von Best Practices, setzt sich mit der Umsetzung von IT Service Management auseinander und wird mit Edition 2011 in dieser Arbeit berücksichtigt.
DevOps hingegen entstand aus einer Reihe von Bewegungen bzw. Techniken, wie Infrastructure as Code, agile Methoden, Lean Thinking, etc. Die Idee ist hierbei, die Bereiche Development und Operations zusammenzuführen. Das Ziel des DevOps-Ansatzes ist es für eine kontinuierliche Softwarebereitstellung zu sorgen, um schnell auf Marktveränderungen bzw. Kundenwünsche reagieren zu können. Dabei spielt Automatisierung eine wesentliche Rolle.
Somit liegt der Zusammenhang zwischen ITIL und DevOps darin, dass sich der nach ITIL organisierte Systembetrieb und die nach DevOps vorgehenden Entwicklerinnen und Entwickler überlegen müssen, wie sie ihre Prozessmodelle in Einklang bringen können.
OpenStack dient als technische Plattform in dieser Arbeit. Auf Basis von OpenStack wird ein Use Case beschrieben, der die technische Grundidee des DevOps-Ansatzes veranschaulicht. Zuvor werden in einem eigenen Kapitel die prozessrelevanten Aspekte in Bezug auf ITIL und DevOps erläutert.

1.1 Motivation

IT-Abteilungen sehen sich immer mehr damit konfrontiert, ihre IT Services stärker an den Business-Anforderungen auszurichten. Dies führt dazu, dass die IT-Organisationen nicht mehr nur als reine Kostenstellen betrachtet werden, sondern als Business Enabler. In weiterer Folge führt dies aber auch dazu, dass IT-Organisationen immer mehr mit ihren Service-Delivery-Zeiten unter Druck geraten. Um diese zu reduzieren, müssen neue Methodiken und Technologien angewandt bzw. eingesetzt werden.
Die Motivation dieser Arbeit liegt darin, zu untersuchen, wie klassische Modelle im IT Service Management und jüngere agilere Modelle wie DevOps miteinander kombiniert werden können.

1.2 Forschungsfrage

Die dieser Arbeit zu Grunde liegende Forschungsfrage lautet: „Lässt sich ITIL, als klassisches Modell für das IT Service Management mit dem agileren DevOps-Ansatz aus der Softwareentwicklung kombinieren?“ Dies soll an Hand eines Use Case untersucht werden, dabei sollen die technischen und organisatorischen Aspekte aufgezeigt werden.
Die Arbeit soll darüber hinaus ein Grundverständnis für ITIL vermitteln. Außerdem soll, beginnend mit den agilen Methoden in der Softwareentwicklung, die Entstehung des DevOps-Ansatzes erläutert werden. Dabei werden auch die relevanten Technologien wie Cloud Computing, Microservice-Architekturen und die technischen Methoden wie Continuous Integration, Continuous Delivery und Continuous Deployment beschrieben.

2 Klassisches IT Service Management mit ITIL

ITIL (IT-Infrastructure Library) ist eine Sammlung von Best Practices und besteht aus fünf Büchern. Der prozessbasierte Ansatz ist, wie in Abbildung 1 dargestellt, in fünf Phasen eines Service-Lebenszyklus aufgeteilt, wobei jede Phase einem Buch entspricht: Service Strategy, Service Design, Service Transition, Service Operation und Continual Service Improvement. Die Bücher enthalten Informationen zu den jeweiligen Themen und bilden einen großen Erfahrungsschatz aus in der Praxis bewährten Herangehensweisen. Diese Sammlung ist über Jahrzehnte gewachsen und berücksichtigt auch Normen wie den IT-Grundschutz, die ISO 9001 und weitere in der Branche verbreitete Normen. [1, pp. 1-2]

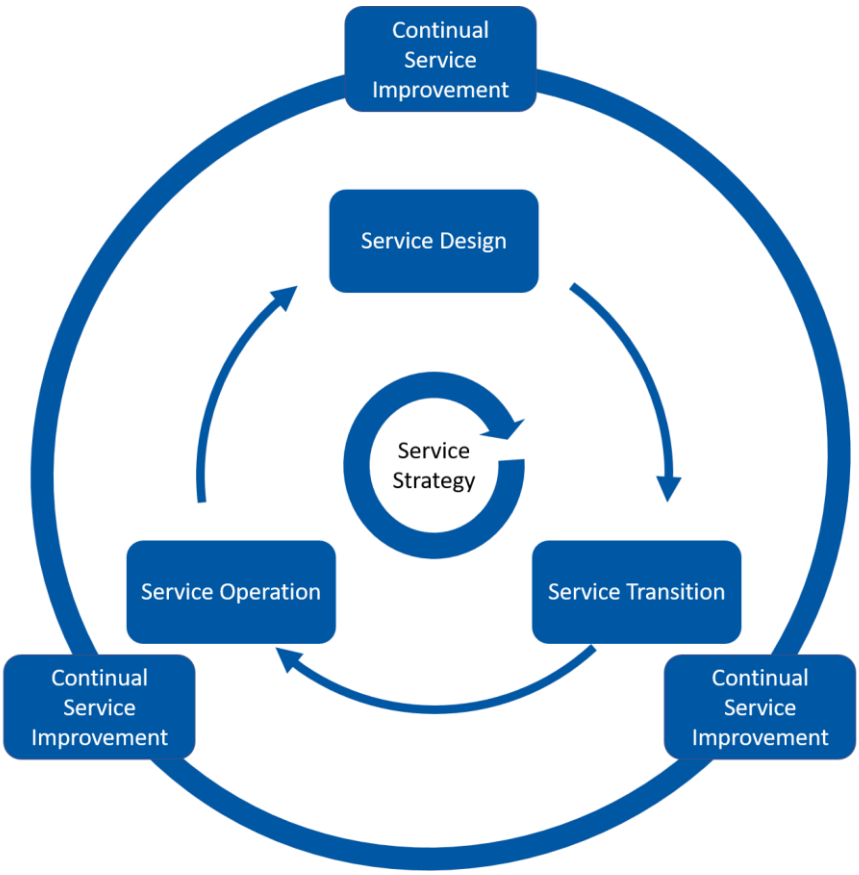

Abbildung 1: ITIL-Service-Lebenszyklus [1, p. 2]

ITIL ist eine sehr verbreitete Möglichkeit, um IT Service Management umzusetzen, und befasst sich vornehmlich mit der Konzeption, Steuerung und Optimierung von IT-Prozessen. Inhaltlich werden außerdem noch Themen wie die Planung, Entwicklung, Einführung und Steuerung von IT Services behandelt. Unternehmen, die ITIL umgesetzt haben, haben oft ein klareres Auftreten gegenüber ihren Kundinnen bzw. Kunden. Sie besitzen ein klares Angebot über die IT Services, die sie anbieten. Darüber hinaus existiert eine umfangreichere Kenntnis

darüber, welche Fachabteilungen welche IT Services genau benötigen. Ein weiterer Vorteil ist, dass die Abhängigkeiten von IT-Infrastrukturkomponenten, die für die Serviceerbringung notwendig sind, bekannt sind. Darüber hinaus wird der Service Level gemessen und sichergestellt, dass die Kundin bzw. der Kunde mit der gelieferten Leistung zufrieden ist. [1, p. 3]

In den nachfolgenden Unterkapiteln werden die einzelnen ITIL-Phasen beschrieben. Inhaltlich wurden nur jene Prozesse detaillierter beschrieben, die für die vorliegende Arbeit relevant sind. Alle anderen wurden nur der Vollständigkeit halber erwähnt, um ein Gesamtbild zu vermitteln.

2.1 Service Strategy

Die Service-Strategy-Phase ist für den Entwurf bis hin zur Implementierung des IT Service Management als Ressource an sich zuständig. Inhaltlich werden hier die Ziele und Richtlinien erstellt, die für alle weiteren Phasen als Vorgabe dienen. [1, pp. 133-134]
Die Strategie enthält dabei sämtliche Maßnahmen, die zur Erreichung der gesetzten Ziele notwendig sind. [1, p. 136]
Die Service Strategy liegt nach ITIL eindeutig im Verantwortungsbereich der Geschäftsführung. Herausfordernd ist hierbei die Erzeugung eines gemeinsamen Verständnisses über die Serviceprioritäten. Bei in Silos organisierten Organisationen liegt die Schwierigkeit dabei oft in der Kommunikation, die darüber hinaus oft auch mit zu viel Bürokratie ausgestattet ist. [1, p. 137]
Ebenso tragen die exakt ausgearbeiteten und abgestimmten Richtlinien und Prozesse, die verstärkt von der Kernaufgabe eines Serviceproviders[1] ablenken, ihren Teil zur herausfordernden Kommunikation zwischen den Silos bei. [1, p. 137]

Die Kernaufgabe eines Serviceproviders:

> *„[…] einen Service bei jedem einzelnen Abruf durch einen Verbraucher verzugs-, naht- und reibungslos in der vereinbarten Qualität zu erbringen. […]“ [1, p. 137]*

Inhaltlich sind in diesem Zusammenhang noch zwei wichtige Begriffe relevant. Zum einen die IT-Strategie und zum anderen die Servicestrategie, die im Folgenden kurz erklärt sind:

Die IT-Strategie beschreibt hierbei, wie ein Unternehmen seine Geschäftsziele durch den Einsatz von Technologie erreichen möchte. [1, pp. 170-171]
Die Servicestrategie beschreibt in weiterer Folge, wie der Serviceprovider die Services nutzt, um die Geschäftsziele seiner Kundinnen bzw. Kunden zu unterstützen. [1, pp. 170-171]

[1] Der Serviceprovider kann als IT-Dienstleister gesehen werden [1, p. 3].

Die nachfolgenden Prozesse sind Teil der Service-Strategy-Phase:

- Strategy Management for IT Services:

  Dieser Prozess ist für die Entwicklung und Pflege der Servicestrategie verantwortlich. Die Servicestrategie muss die übergeordnete Geschäftsstrategie unterstützen. [1, p. 177]

- Service Portfolio Management:

  Das Service Portfolio enthält grundsätzlich alle geplanten, aktiven und außer Dienst gestellten Services. Das Service Portfolio Management verwaltet als strategischer Prozess die Services über ihren gesamten Lifecycle hinweg. Somit kümmert sich dieser Prozess auch um die Fragestellung, welche Services angeboten werden, um das Business bestmöglich zu unterstützen. [1, p. 191]

- Financial Management for IT Services:

  Gegenstand dieses Prozesses ist die Kostenrechnung, Finanzplanung und Leistungsverrechnung. Dafür ist ein enger Kontakt mit dem Finanzmanagement notwendig. [1, p. 211]

- Demand Management:

  Kern dieses Prozesses ist die Kapazitätsplanung, um rechtzeitig und wirtschaftlich Kapazitäten zur Verfügung stellen zu können. Dabei sollen auch Kapazitätsspitzen und Performance-Lücken geeignet abgefedert werden können. [1, p. 231]

- Business Relationship Management:

  Dieser Prozess stellt eine Verbindung zwischen Kundin bzw. Kunde und Serviceprovider her, mit dem Ziel das Verständnis über die Businessanforderungen der Kundin bzw. des Kunden zu verbessern. So können die Services besser an diese Anforderungen ausgerichtet werden. [1, p. 240]

Zusammenfassend gesprochen befasst sich diese Phase mit der Definition der grundlegenden Prozesse im Service Management und mit den jeweiligen Definitionen der IT-Strategie und der Servicestrategie. [1, pp. 170-171]

2.2 Service Design

Die aus der Service-Strategy-Phase resultierenden Services müssen nun umgesetzt werden. Zuerst wird die Umsetzung noch geprüft, anschließend werden die Services designt und die notwendigen Bestandteile beschafft und getestet. Im Anschluss an diese Phase wird das zur Umsetzung anstehende Paket der Transition-Phase übergeben. Ein Servicedesign umfasst den Service, die Prozesse, die Architektur, etwaige Richtlinien und die Dokumentation. [1, p. 264]

Die nachfolgenden Prozesse sind Teil der Service-Design-Phase:

- Design Coordination:

  Gegenstand dieses Prozesses ist die zentrale Koordinierung und Steuerung der Service-Design-Prozesse. [1, p. 309]

- Service Catalogue Management:

  Der Servicekatalog enthält die operativen und somit bestellbaren Services. Darin enthalten sind Informationen je Service wie z.B. Preis, Kontaktperson, Bestell- und Lieferprozeduren. [1, pp. 194-195]
  Das Service Catalogue Management ist für die Pflege des Servicekatalogs zuständig, um stets aktuelle, korrekte und vollständige Informationen über den jeweiligen Service anzubieten. [1, p. 318]

- Service Level Management:

  Beim Service Level Management werden die Kundenanforderungen für einen Service vertraglich vereinbart. Dafür müssen die Services ständig überwacht werden. [1, pp. 325-326]

- Availability Management:

  Dieser Prozess stellt sicher, dass die definierten Verfügbarkeitslevel der Services eingehalten werden. [1, p. 345]

- Capacity Management:

  Das Capacity Management kümmert sich um die Kapazitäts- und Performanceanforderungen der Services. [1, p. 375]

- IT Service Continuity Management:

  Dieser Prozess beschäftigt sich mit der Planung und Erstellung von Notfallplänen, die zur raschen Wiederherstellung der Services nach Katastrophen, wie z.B. Blitzeinschlag, Brand etc., notwendig sind. Er ist somit die technische Unterstützung des Business Continuity Managements. [1, pp. 392-393] Letzteres beschäftigt sich mit Analyse und Management von Risiken. [1, p. 394]

- Information Security Management:

  Dieser Prozesses ist für die Gewährleistung von Vertraulichkeit, Integrität und Verfügbarkeit der Services, samt den von ihnen verarbeiteten Daten, zuständig. Dadurch soll ein Grad an Informationssicherheit erreicht werden, der die Businessanforderungen erfüllt. [1, p. 408]

- Supplier Management:

  Durch diesen Service wird sichergestellt, dass alle Verträge mit Lieferanten und Dienstleistern entsprechend eingehalten werden. [1, p. 425]

Es fällt auch die Gestaltung der ITIL-Prozesse selbst in die Service-Design-Phase, da sie die Basis für die IT-Service-Erbringung sind. Alle weiteren Phasen basieren auf jenen Standards und Outputs, die sich aus den Servicedefinitionen ergeben. Dabei ist das Feedback aus dem Continual Service Improvement für die ständige Serviceverbesserung entscheidend. Deshalb besteht zwischen diese beiden Phasen (Continual Service Improvement und Service Design) auch ein enger Kontakt. [1, p. 264]

Ob ein Service erfolgreich designt ist, lässt sich am besten daran beurteilen, ob er die vom Business definierten Ergebnisse erzielt und gleichzeitig die Vorgaben und Randbedingungen der Service-Strategy-Phase einhält. [1, p. 266]

2.3 Service Transition

Die in den vorherigen Phasen gestalteten Services müssen nun in den Betrieb überführt werden. Damit und auch mit der Stilllegung von Services beschäftigt sich diese Phase der Service Transition. [1, p. 448], [1, p. 452]
Inhaltlich geht es konkret um die effiziente und effektive Planung, Steuerung und Produktivsetzung von neuen oder geänderten Services. Generell werden Änderungen (an der Architektur, den Prozessen, den Tools, den Messgrößen, der Dokumentation, den IT Services, etc.), die Auswirkungen auf die IT Services haben könnten, Service Changes genannt. [1, p. 450]
Releases werden durch standardisierte Prozessaktivitäten möglichst ohne negative Auswirkungen auf die IT Services implementiert. Ein Release kann dabei aus mehreren Changes bestehen. [1, p. 450]
Darüber hinaus ist ein weiteres Ziel dieser Phase auch die Sicherstellung, dass die Changes den gewünschten Nutzen erbringen und alle Compliance-Anforderungen erfüllt werden. Dies kann durch Messungen der operativen Nutzung, Reviews und Audits überprüft werden. [1, p. 450]
Weitere Aufgabengebiete dieser Phase sind die Umsetzung des Riskmanagements und die Bereitstellung von Wissen und Informationen für die geänderten oder hinzugefügten Services. Dabei dienen das Wissen und die Informationen dem Change, Release und Deployment Management als Entscheidungsgrundlage um Releases aus der Testumgebung ausrollen zu können. [1, pp. 450-451]

Die nachfolgenden Prozesse sind Teil der Service Transition Phase:

- Transition Planning and Support:

  Hier passieren übergreifende Planungsarbeiten, um den designten Service effektiv in den Betrieb zu überführen. Das heißt, hier werden Aktivitäten mit allen beteiligten Projekten, Lieferanten und internen Serviceteams abgestimmt. [1, p. 470], [1, p. 472]
  Außerdem werden alle anderen Bereiche und Stakeholder beim Umgang mit den Planungsergebnissen aus diesem Prozess unterstützt. Dabei erfolgt die Unterstützung in Form von Beratung, Administration, Kommunikation, Fortschrittsüberwachung und Reporting. [1, p. 473]

  Für die Service Transition gibt es nach N. Ebel [1, p. 475] die folgenden Lebenszyklusphasen. Dabei beschreibt jede Phase, wie der Übergang zur nächsten formal zu erfolgen hat, und definiert entsprechende Eintritts- und Austrittskriterien sowie die notwendigen Ergebnisse in Form von Dokumenten:

  - Beschaffen und Testen neuer Komponenten und Configuration Items (CI)[2]
  - Build und Test
  - Service Release Test
  - Test für die Betriebsbereitschaft des Service (Service Operational Readiness Test)

[2] Vgl. „Change Management“ auf Seite 14.

- Deployment
- Early Life Support
- Review und Abschluss der Service Transition

- Change Management:

Verschiedenste Umstände können Änderungen an der IT-Infrastruktur notwendig machen. Zu den Auslösern zählen unter anderem geänderte Geschäftsprozesse, die Behebung von Störungen oder die Verbesserung von bestehenden Services. Änderungen an der Umgebung sind demnach alltäglicher Bestandteil eines Serviceproviders. Gelegentlich kommt es aufgrund von Änderungen zu Störungen der IT Services. Dies kann an mangelnder Kommunikation, unzureichender Dokumentation, zu geringen Ressourcen, etc. liegen. Das Change Management hat zum Ziel, die Unterbrechungen und die darauf folgenden Auswirkungen aufgrund von Änderungen zu minimieren. [1, p. 481]

Das Change Management evaluiert, autorisiert und implementiert Changes für Services, die z.B. durch das Hinzufügen neuer Komponenten oder durch das Ändern von bestehenden Komponenten verursacht werden. Der Prozess dient dazu, die Folgen von Changes zu beurteilen, das Risiko zu reduzieren und die Kommunikation mit den verschiedenen Stakeholdern zu verbessern. Das Change Advisory Board (CAB) ist das verantwortliche Gremium, welches für die Genehmigung von Changes zuständig ist. Ein effektives Change Management wägt immer zwischen dem Risiko einer Änderung und dem Nutzen, den diese Änderung dem Unternehmen bringt, ab. [2]

Den Begriff des Changes hat jedes Unternehmen für sich zu definieren. Darüber hinaus muss sich auch jedes Unternehmen darüber Gedanken machen, welche Änderungen nicht unter die Kontrolle des Change Management fallen, sodass keine unnötige Bürokratie für einfache operative Tätigkeiten entsteht. In jedem Fall sind alle Changes zu steuern, zu verwalten und zu dokumentieren. Für manche Unternehmen ist dies auch zur Nachweiserbringung für die Einhaltung von rechtlichen Rahmenbedingungen erforderlich. [1, p. 483]
Im Change-Management-Prozess sind die beiden Repositorys Configuration Management System (CMS) und Serviceportfolio wichtig. [1, p. 484]
Das Serviceportfolio beinhaltet Informationen zu allen aktuellen, stillgelegten und geplanten Services. Diese Informationen werden zur Bewertung der Auswirkungen von anstehenden Changes auf die bestehenden Services herangezogen. [1, p. 484]

Ein Configuration Item (CI) ist ein Service Asset, welches gemanagt wird, damit ein Service erbracht werden kann. Es stellt also ein einzelnes Konfigurationselement dar. [1, p. 513]
Beispiele für CIs sind Router, Laptop, Dokumentation, Firewall, Datenbank, Applikationsserver, etc. [1, p. 514]

Informationen zu den einzelnen CIs werden als Configuration Record (CR) in der Configuration Management Database (CMDB) abgelegt und über das Configuration Management System (CMS) verwaltet. Ein CR beschreibt ein CI und enthält jene Informationen die für andere Prozesse wie dem Change oder Incident Management wichtig sind. ITIL empfiehlt grundsätzlich für die Pflege der CMDB automatisierte Prozesse zu nutzen, um Kosten zu sparen und die Fehleranfälligkeit zu reduzieren. Dafür können Discovery, Inventory oder Audit Tools an das CMS angeschlossen werden. [1, pp. 513-514]

Einem Change geht immer ein Request for Change (RfC) voraus, dieser stellt sozusagen den Antrag für die Veränderung an einem bestimmten CI dar. Wird der RfC genehmigt, so wird er zum Change. [1, p. 485]

ITIL definiert die folgenden Change-Typen [1, p. 487]:

- Normale Changes, stellen Änderungen an einem oder mehreren CIs oder einem Service dar. Diese werden formell beantragt und müssen genehmigt werden.
- Standard Changes sind vorab genehmigte Standardabläufe mit geringem Risiko. Das Genehmigungsverfahren wird hier nur einmal durchlaufen.
- Emergency Changes werden so rasch als möglich mit nur einem Mindestmaß an Vorabtests umgesetzt. Die Dokumentation und die Überprüfung werden in der Regel erst nach Abschluss des Changes aufgearbeitet.

- Service Asset and Configuration Management (SACM):

  Ziel dieses Prozesses ist es, ein vollständiges und aktuelles CMS, welches alle notwendigen Informationen über die CIs und ihre Abhängigkeiten zu einander enthält, zur Verfügung zu stellen. [1, pp. 510-511]

- Release and Deployment Management:

  Gemäß ITIL bedeutet der Begriff Release die Zusammenfassung von mehreren Changes, die gemeinsam ausgerollt werden sollen. Die Aufgabe dieses Prozesses ist die Planung und Steuerung der Phasen Build, Test und Deployment eines Releases. Ziele dieses Prozesses sind unter anderem die Abstimmung der Release-und-Deployment-Pläne mit den beteiligten Stakeholdern und die sichere Speicherung der Release-Inhalte in einem Repository. [1, p. 533]

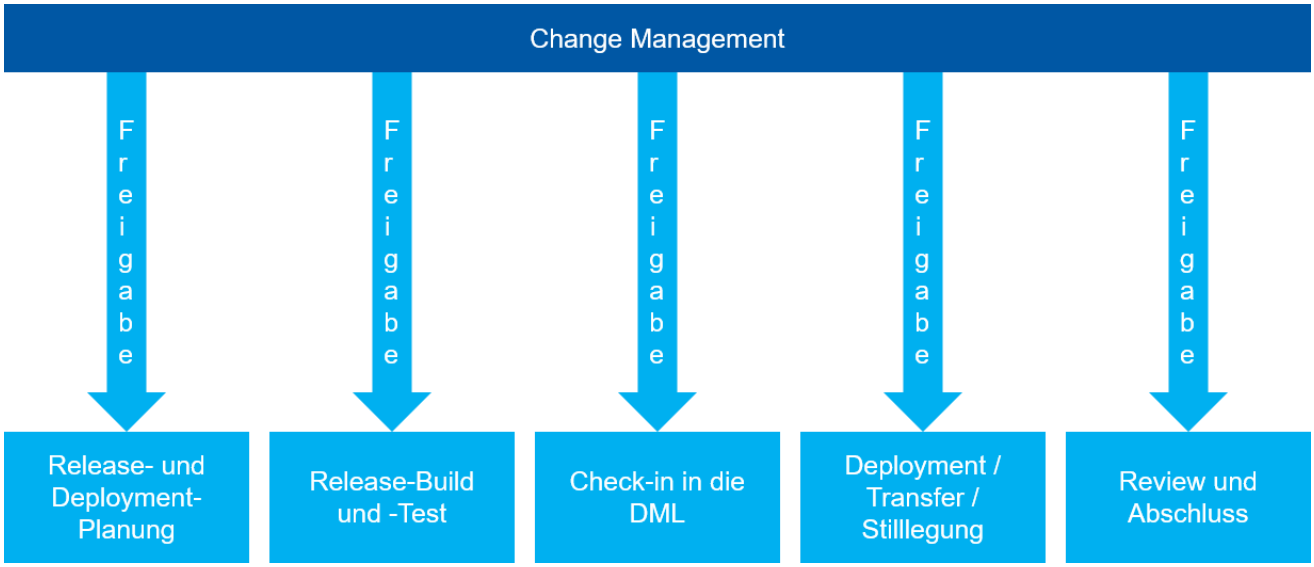

Abbildung 2: Phasen des Release and Deployment Management [1, p. 540]

Die Phase „Release- und Deployment-Planung“ in Abbildung 2 beschäftigt sich mit der Entwicklung der Pläne, die für die Umsetzung des Releases notwendig sind. [1, p. 541]

Inhaltlich werden hier folgenden Themen behandelt [1, pp. 541-544]:

- Kriterien für Erfolg und Misserfolg
- Build- und Testplanung
- Planung der Release-Paketierung und des Release-Build
- Vorbereitung für Build und Test
- Deployment-Planung
- Logistik und Planung der Lieferung und Ausbringung
- Planung von Pilottests
- Finanzplanung

Die Phase „Release-Build und -Test” in Abbildung 2 beschäftigt sich mit Standardisierungs- und Integrationsaspekten sowie mit dem Management der Konfigurationen. In dieser Phase spielen die unterschiedlichen Umgebungen wie Entwicklungs-, Build-, Test- und Produktivumgebung eine wichtige Rolle. [1, pp. 544-545]

Inhaltlich werden hier folgenden Themen behandelt [1, pp. 544-548]:

- Release- und Build-Dokumentation
- Beschaffung und Testen der CIs und Komponenten
- Release-Paketierung mit dazugehöriger Dokumentation

- Build und Management der Testumgebung
- Servicetests und Pilotierung
- Service-Generalprobe
- Pilottest

Als nächstes folgt in Abbildung 2 der „Check-in in die DML“. Die Definitive Media Library (DML) stellt ein Repository für das Speichern von Software-, Dokument- oder anderen elektronischen CIs einschließlich deren aktueller Lebenszyklusphase (Design, Build, Test, Deployment und Betrieb) dar. Darüber hinaus beschränkt die DML den Zugriff auf die Daten. Somit stellt sie das Fundament für das Release and Deployment Management dar und enthält somit alle qualitätsgeprüften Master-Kopien aller entwickelten oder gekauften Softwareprodukte inkl. Dokumentation und Lizenzen. [1, p. 518]

Die Phase „Deployment / Transfer / Stilllegung” in Abbildung 2 beschäftigt sich mit der Verteilung des Release Package aus der DML in die Produktivumgebung [1, p. 548].

Inhaltlich werden hier folgenden Themen behandelt [1, pp. 548-550]:

- Planung und Deployment-Vorbereitung
- Transfer, Deployment und Stilllegung
- Deployment-Verifizierung
- Early Life Support

Die Phase „Review und Abschluss” in Abbildung 2 beschäftigt mit dem Sammeln von Erfahrungen und Feedback um ggf. Maßnahmen abzuleiten [1, p. 550].

Inhaltlich werden hier folgenden Themen behandelt [1, pp. 550-551]:

- Review und Deployment-Abschluss
- Review und Transition-Abschluss

- Service Validation and Testing:

Dieser Prozess stellt sicher, dass die Release-Zusammenstellung und die Tests in wiederholbarerer und nachvollziehbarer Art und Weise stattfinden. Der qualitätssichernde Auftrag dieses Prozesses liegt außerdem darin, sicherzustellen, dass der geänderte Service allen Ansprüchen in Bezug auf Zuverlässigkeit und Funktionalität entspricht. [1, pp. 554-555]

- Change Evaluation:

  Change Evaluation prüft die Auswirkungen eines Change auf die Geschäftsergebnisse, Services und IT-Infrastruktur. Er unterstützt das Change Management bei den Freigabeentscheidungen und versorgt es so mit fundierten Informationen. Die Service Changes werden auf erwartete und unerwartete Effekte und die damit verbundenen Risiken hin untersucht. [1, pp. 570-571]

- Knowledge Management:

  Das Knowledge Management kümmert sich, grob umrissen, um den Austausch von Wissen und um dessen Verfügbarkeit am richtigen Ort zur richtigen Zeit. [1, p. 577]

2.4 Service Operation

Wie sich schon aus dem Namen dieser Phase vermuten lässt, geht es nun darum, die designten und gebauten Services zu betreiben, sodass die Kundin und der Kunde den durch den Service erhofften Mehrwert erhält. So ist es für die Mitarbeiterinnen und Mitarbeiter dieser Phase wichtig, den Service aus einer Gesamtsicht heraus zu betrachten, d.h. von den Anwendungen über die Prozesse bis hin zur Infrastruktur. Um diese Gesamtsicht erst zu ermöglichen, ist dabei auch der Einsatz von entsprechenden Tools notwendig. Primäre Ziele dieser Phase sind die Aufrechterhaltung von Anwender- und Kundenzufriedenheit, das Reduzieren von Ausfällen und die Sicherstellung, dass nur berechtigte Nutzer Zugriff auf die Services erhalten. [1, pp. 604-606]
Inhaltlich werden dabei alle Aktivitäten rund um den Service selbst behandelt, mit dem Ziel, das definierte Service Level zu erbringen. Dazu gehört auch das Managen der entsprechenden Technologien, auf die der Service basiert. Das sind z.B. die Server, Netzwerke, Datenbanken, Überwachungs-Werkzeuge etc. [1, pp. 606-607]
Durch den Einsatz dieser Best Practices kann insofern eine Kostenersparnis für das Unternehmen erzielt werden, als es zu weniger ungeplanten Ausfällen kommt und somit keine ungeplanten Kosten durch diese Arbeitseinsätze generiert werden. Darüber hinaus sorgt eine ausreichende Automatisierung dafür, dass Routineaufgaben effizienter und reproduzierbar abgearbeitet werden können. [1, p. 607]

Die nachfolgenden Prozesse sind Teil der Service Operation Phase:

- Event Management:

  Dieser Prozess kümmert sich um die Steuerung und das Management aller Statusänderungen eines CIs (Events). [1, pp. 623-624]
  Dabei ist das Aufgabenspektrum groß, denn es müssen nicht nur Infrastrukturkomponenten überwacht werden, ob sie online oder offline sind. Auch Lizenzen und Umgebungsbedingungen wie Rauch- oder Feuererkennung sind Teil des Event Managements, genauso wie IT-Security-relevante Aspekte. [1, p. 625]

- Incident Management:

  Das Incident Management kümmert sich um das Registrieren, Prüfen, Kategorisieren, Priorisieren und Verfolgen von Störungen an Services (Incidents). Incidents stellen demnach ungeplante Unterbrechungen an den Services dar. [1, p. 638]

- Request Fulfilment:

  Das Request Fulfilment kümmert sich um Service Requests. Service Requests sind ganz allgemeine Anfragen einer Anwenderin bzw. eines Anwenders. [1, p. 633]
  Beispiele für diesen Bereich sind das Rücksetzen eines Passwortes oder die Anforderung eines neuen Arbeitsplatzes für eine Anwenderin bzw. einen Anwender. Werden allerdings Störungen gemeldet, so werden sie an das Incident Management übergeben. [1, p. 664]

- Problem Management:

  Ein Problem definiert sich nach ITIL als Ursache von Incidents, dieses kann durch einen einzigen oder mehrere Incidents verursacht worden sein. Der Prozess Problem Management ist somit für das Managen dieser Probleme über ihren Lifecycle hinweg verantwortlich. [1, p. 677]
  Der Prozess sorgt dafür, dass die Auslöser gefunden und entsprechend behoben werden. Das Verfahren sorgt auch dafür, dass die notwendigen Dokumentationen erstellt werden. Die Behebung erfolgt dann über das Change Management und das Release and Deployment Management. [1, p. 678]

- Access Management:

  Der Access-Management-Prozess kümmert sich zusammengefasst um die Vergabe von Zugriffsrechten an autorisierte Anwenderinnen bzw. Anwender, dieses Aufgabengebiet wird auch unter dem Begriff Identity Management geführt. [1, p. 697]

2.5 Continual Service Improvement

Diese Phase zieht sich durch alle ITIL-Phasen. Der Fokus liegt dabei auf der Abstimmung der IT Services mit den sich ändernden Geschäftsanforderungen. Somit geht es um die Verbesserung der Service- und Prozesseffektivität sowie um die Optimierung der Wirtschaftlichkeit. Um den Grad der Verbesserung messbar zu machen, muss es Messverfahren und Zieldefinitionen geben. [1, p. 755]

3 Agile Methoden

Im Managementumfeld kam der Begriff der Agilität bereits in den 1970er Jahren vor. Dabei lag der Fokus immer auf der Fähigkeit von Unternehmen, möglichst flexibel und aktiv auf Marktveränderungen zu reagieren. [3, p. 6]

Somit lässt sich laut [4, p. 3] folgende Definition für Agilität finden:

„Agilität ist die Fähigkeit sich auf alle Arten von Veränderungen einzustellen und Ziele, Inhalte, Organisation und Prozesse zeitgerecht anzupassen.“ [4, p. 3]

Das Projektmanagement im Konkreten beschäftigte sich Ende der 1990er Jahren mit dem Thema Agilität, das damals im direkten Kontrast zu den klassischen Planungsmethoden (Wasserfallmodell, etc.) stand. [3, p. 7]
Über Alternativen zu klassischen Planungsmethoden im Softwareentwicklungsbereich wurde deshalb nachgedacht, da zu dieser Zeit bis zu 90% [3, p. 7] der IT-Projekte scheiterten. Die Ursache lag oft in einer zu unflexiblen und starren Planung, denn diese Art von Planung setzt voraus, dass die Auftraggeber genau wissen, was sie wollen, und dass sich diese Anforderungen im Laufe des Projektes nicht ändern. Diese Voraussetzungen sind bei Softwareprojekten oft schwierig zu erfüllen, weshalb viele Projekte scheiterten. [3, p. 7]
Die Agilität war folglich eine Antwort auf die gescheiterten Projekte im IT-Bereich und entstand aus einer Bewegung, die im Februar 2001 von einer Entwicklergruppe um Kent Beck ausging. Diese Gruppe veröffentlichte das agile Manifest. [5], [3, p. 7]

Das agile Manifest definiert im Wesentlichen die folgenden Werte:

„[…] • Individuen und Interaktionen mehr als Prozesse und Werkzeuge

• Funktionierende Software mehr als umfassende Dokumentation

• Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung

• Reagieren auf Veränderung mehr als das Befolgen eines Plans […]“ [3, p. 8], [5]

Diese Werte legen die Prioritäten fest, wie sich die genannten Gegensätze zueinander verhalten. Das Manifest entstand in einer Zeit, als der zweite Teil der Aussagen stark dominierte und darf keinesfalls so interpretiert werden, dass Prozesse und Werkzeuge, eine umfassende Dokumentation, Vertragsverhandlungen und die Befolgung eines Plans unwichtig oder nicht relevant wären. Vielmehr geht es darum, die Schwerpunkte richtig zu setzen. Unter heutigen Gesichtspunkten scheinen viele dieser Aussagen wenig revolutionär, da sie eben in einer Zeit entstanden sind, als die klassischen Planungsmethoden stark dominierten. [3, p. 8]

Um agile Methoden in der Softwareentwicklung oder im Führungsstil einfließen lassen zu können, bedingt dies auch ein respektvolles Arbeitsklima im Unternehmen, denn durch agile Methoden werden Arbeitsprozesse offengelegt und transparent gemacht. Dieses Offenlegen darf bei Mitarbeiterinnen und Mitarbeitern, die mit ihren Aufgaben in Verzug geraten sind, nicht ein Gefühl des An-den-Pranger-Stellens erwecken, da agile Methoden davon leben, dass Probleme und Schwierigkeiten sofort angesprochen werden, um so zeitnah Lösungen herbeizuführen. Darüber hinaus ist es auch wesentlich, aus Fehlern zu lernen. Dies bedingt ebenso eine entsprechende Fehlerkultur. [3, pp. 9-10]

In den nachfolgenden Kapiteln werden zwei sehr prominente agile Methoden näher beschrieben. Die Auswahl der Methoden erfolgte aufgrund ihrer Verbreitung:

> *„[…] Rund 86 % des Marktes agiler Methoden werden laut einer Umfrage der Fachhochschule Koblenz von Scrum dominiert, gefolgt von Kanban. […]“ [3, p. 11]*

3.1 Scrum

Scrum ist im Wesentlichen ein Prozessframework, das bei der Entwicklung von komplexen Produkten verwendet werden kann und auf der Theorie der empirischen Prozessteuerung basiert. Somit beruht das Wissen auf Erfahrungen, anhand derer zukünftige Entscheidungen getroffen werden. Dabei kommt ein iterativer und inkrementeller Ansatz zum Einsatz. [6, p. 3]

Dies bedeutet konkret, dass es sogenannte Sprints mit klaren Zieldefinitionen gibt. Sprints sind kurze, in der Regel unter einem Monat dauernde Entwicklungsiterationen. [4, p. 13]
Abbildung 3 zeigt den Scrum-Prozess, der nachfolgend detaillierter beschrieben wird.

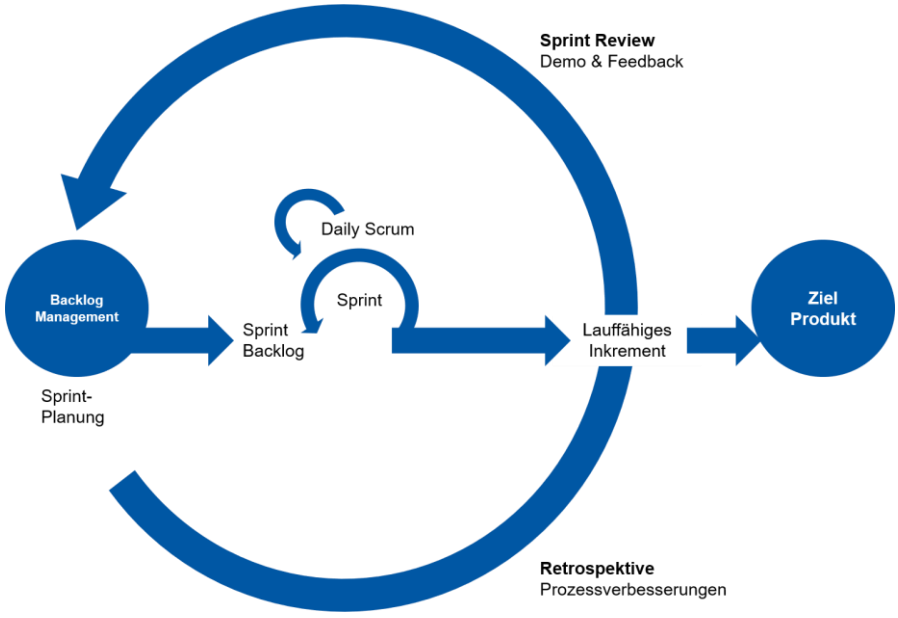

Abbildung 3: Scrum-Prozess [4, p. 15]

- Product Backlog:

  Das Product Backlog enthält alle Änderungsanforderungen und ist dynamisch, da es sich mit dem Produkt weiterentwickelt. Die Verantwortung über das Product Backlog hat der Product Owner.
  Darin aufgelistet sind Features, Funktionalitäten, Verbesserungen und Fehlerbehebungen, die in die neuen Releases des Produktes eingearbeitet werden sollen. [6, p. 14]

- Sprint Backlog und Daily Scrum:

  Das Sprint Backlog enthält die für den aktuellen Sprint ausgewählten Einträge aus dem Product Backlog. Darüber hinaus sind Informationen über die Lieferung des Produkts und der Sprint-Ziele enthalten. [6, p. 15]
  Die Fortschrittsverfolgung erfolgt dabei in einem Daily Scrum Meeting. Dieses Meeting ist sehr kurz gehalten (15 Minuten). Schwerpunkt ist die Abstimmung der Entwicklerinnen und Entwickler untereinander und die Planung der folgenden 24 Stunden. Augenmerk liegt auf der bereits durchgeführten Arbeit, der anstehenden Arbeit und auf etwaigen identifizierten Problemen. [6, p. 11]

- Sprint:

  Sprints sind abgeschlossene und auslieferbare Produkt-Inkremente. Die Zielvorgaben sind so gewählt, dass in etwa ein Arbeitsaufwand von maximal einem Monat entsteht. Während des Sprints werden keine Änderungen vorgenommen, durch die das Ziel des Sprints gefährdet werden würde. Nach Abschluss eines Sprints startet sofort die nächste Iteration mit dem nächsten abgeschlossenen Sprint. [6, p. 8]

- Sprint-Planung:

  Zu Beginn jedes Sprints wird in etwa ein Tag für die Erstellung der Arbeitspakete (Sprint Backlog) investiert. [4, p. 14]

- Sprint Review:

  Beim Sprint Review werden am Ende jedes Sprints dem Product Owner und interessierten Stakeholdern die Ergebnisse des aktuellen Sprints live am System präsentiert. Dies ermöglicht direktes Feedback über die neu entwickelten Features. Allfällige Nachbesserungsarbeiten fließen in die zukünftigen Sprintplanungen mit ein. [4, p. 15]

- Retrospektive:

  Hier wird durch die Teammitglieder die Qualität des Scrum-Prozesses bewertet, indem von jedem Mitglied ein Feedback über die positiven und negativen Erkenntnisse der vergangenen Iteration berichtet wird. Dies dient zur Verbesserung des Prozesses an sich, um aus der Vergangenheit lernen zu können. [4, p. 15]

Neben dem Prozess definiert Scrum auch die folgenden Rollen:

- Der Product Owner ist als fachlich Verantwortlicher für das Produkt die Brücke zwischen IT und Fachbereich. Er nimmt die Anforderungen der Kundin bzw. des Kunden entgegen und strukturiert, klassifiziert und priorisiert diese. Diese Anforderungen landen anschließend im Product Backlog. [4, p. 14]

- Der Scrum Master managt das Team, trägt Sorge dafür, dass das Team ungestört arbeiten kann, und hält Hindernisse von außen vom Team fern. Er ist nicht weisungsbefugt, hat aber dafür Sorge zu tragen, dass der Scrum-Prozess eingehalten wird. Darüber hinaus coacht er das Team und moderiert im Bedarfsfall die Meetings des Teams. [4, p. 14]

- Das Scrum-Team selbst repräsentiert die Entwicklerinnen und Entwickler, die sich in Selbstorganisation um die Erstellung der einzelnen Produkt-Inkremente kümmern. Dabei sind sie dazu autorisiert, alle zielführenden Maßnahmen zu ergreifen, um das Sprint-Ziel zu erreichen. [4, p. 14]

3.2 Kanban

Kanban stammt ursprünglich aus Japan und ist ein Visualisierungsframework, welches auf dem Ansatz der agilen Methoden basiert. Es zeigt im Wesentlichen, welche Arbeitspakete anstehen, wann diese fällig sind und wie viele davon abzuarbeiten sind. [7]
Das Wort selbst setzt sich aus den beiden japanischen Wörtern „KAN“ und „BAN“ zusammen. „KAN“ steht für das Visualisieren und „BAN“ steht für die Karte oder die Tafel. Die zugrundeliegende Idee ist simpel. Es wird die Anzahl der parallelen Arbeitspakete (Work-in-Progress) auf das tatsächlich Schaffbare reduziert. Dies basiert auf der Engpasstheorie, die besagt, dass der Engpass die Gesamtleistung eines Systems limitiert. [4, p. 22]
Aus diesem Ansatz resultiert der Vorteil, dass durch die Vermeidung von häufigem Kontextwechsel der Zeitaufwand für das Wiederhineindenken deutlich sinkt. [4, p. 24]
Die Visualisierung auf dem Kanban-Board und die Beschränkung der parallelen Arbeitspakete verstärkt die Transparentmachung von etwaigen Engpässen und Problemen, da dadurch sehr schnell sichtbar wird, bei wem sich die Arbeit staut und wer Unterstützung braucht. Abbildung 4 zeigt beispielhaft ein Kanban-Board. Darauf sind die einzelnen Phasen des Gesamtprozesses abgebildet (vom Backlog bis hin zum Zustand „Done“). Jedes Arbeitspaket ist in der entsprechenden Phase eingezeichnet, in der es sich aktuell befindet. [4, p. 22]

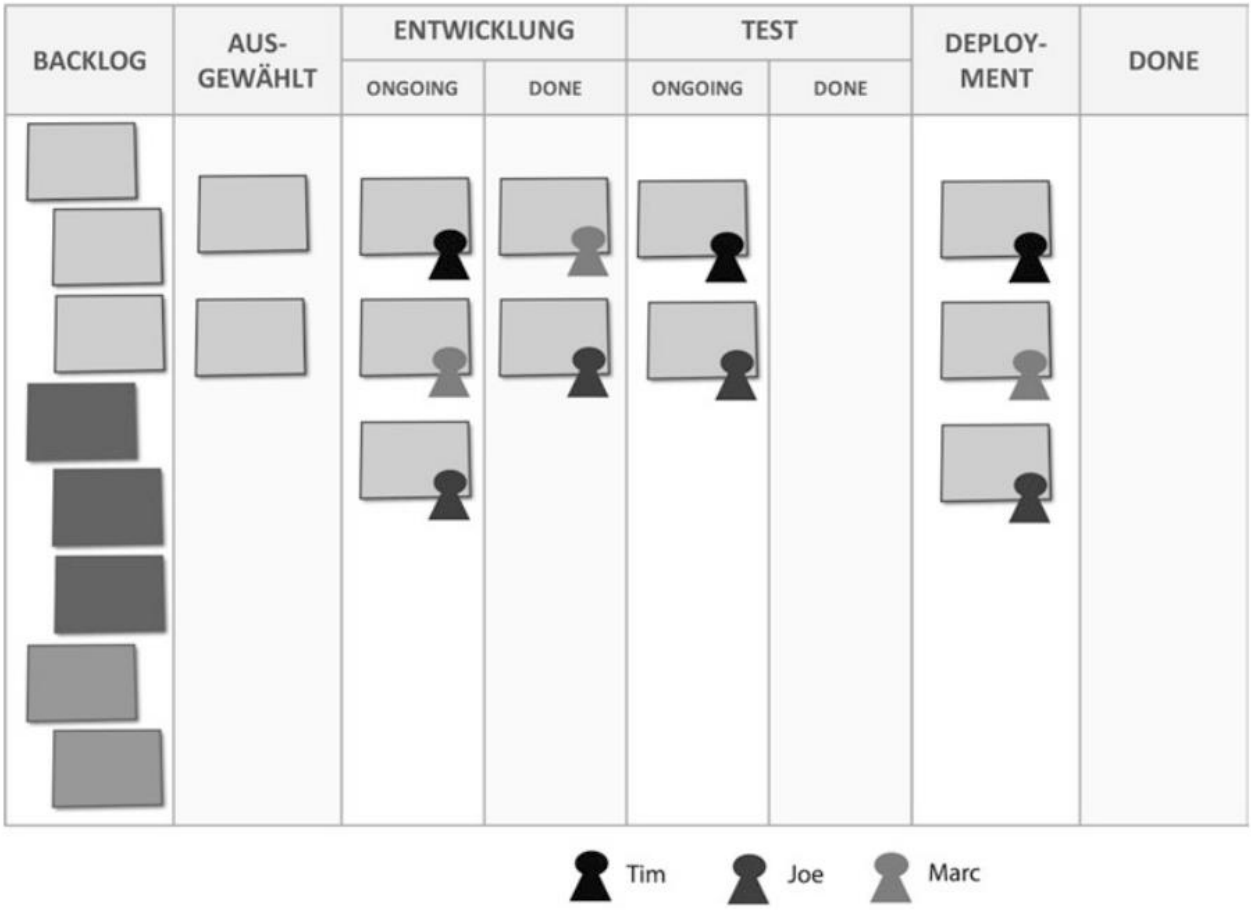

Abbildung 4: Beispiel eines Kanban-Boards [4, p. 23]

Bei Kanban gibt es keinen Master, der die Oberhand über die Arbeitspakete hat. Vielmehr pflegen die Teammitglieder das Kanban-Board selbst und ziehen die jeweiligen Arbeitspakete selbständig in die nächste Phase, sofern dort für diese noch Platz sind. [4, p. 23]

Für die wichtige kontinuierliche Verbesserung in Kanban werden unter anderem tägliche Standup-Meetings, Arbeitsreviews und sinnvolle Messgrößen wie Durchlaufzeiten, Fehlerraten oder Termintreue herangezogen. [4, p. 24]

Wichtig für dieses Vorgehensmodell ist ein gut funktionierendes Konfigurationsmanagement sowie automatisierte Build-, Test- und Deployment-Prozesse. [4, p. 27]

3.3 Scrum vs. Kanban

Kanban ist einfacher zu nutzen als Scrum. Bei Kanban müssen die Team-Mitglieder beispielsweise keine Routinen wie die Daily Scrums, Retrospektiven oder Sprint-Planungen durchlaufen. Somit können Teams schneller nach Kanban arbeiten als nach Scrum, weil hier weniger Methodenbestandteile existieren. Sind allerdings genau diese methodischen Vorgaben erwünscht, so ist Scrum eher das Mittel der Wahl. [4, p. 28]

Hanschke [4] definiert in Tabelle 1 die markanten Unterschiede zwischen Scrum und Kanban.

	Scrum	Kanban
Time Boxing	Regelmäßige Sprints mit fester Länge vorgeschrieben	Kein Methodenbestandteil; Fokus auf kontinuierlichem Fluss; Iterationen sind optional
Begrenzung der Inhalte	Ergebnis Sprint-Planung – Team vereinbart Menge an Arbeit, die in Sprint zu erledigen ist	Work-in-Progress (WIP); Commitments optional
Granularität der Anforderungen	Anforderungen müssen in die Iteration eingepasst und hierfür ggf. zerlegt werden	Keine Vorgabe bzgl. Granularität der Anforderungen
Release-Methoden	Am Ende jedes Sprints, wenn vom Product Owner genehmigt	Nach Ermessen des Teams; in der Regel Continuous Delivery
Rollen	Product Owner, Scrum Master und Scrum Team	Keine Rollen
Metriken für Planung und Prozessverbesserung	Velocity	Durchlaufzeit
Art der Planung bzw. Planungsänderung	Sprint-Planung und keine Änderung während des Sprints	Backlog kann jederzeit geändert werden und Teammitglied kann jederzeit neue Kanban-Karte bearbeiten

Tabelle 1: Unterschied zwischen Scrum und Kanban [4, p. 29]

Grundsätzlich sind Scrum und Kanban aber auch kombinierbar (Scrumban). Dabei werden die Visualisierungsaspekte und der Fokus auf den Work-in-Progress aus Kanban mit den Rollen und dem Vorgehensmodell aus Scrum vermengt. [4, p. 29]

Generell kann festgehalten werden, dass für kleinere Projekte mit geringeren Veränderungsraten Kanban eine gute Lösung ist. [4, p. 30]

3.4 Agile Techniken

In diesem Kapitel werden einige agile Techniken aufgezählt, jedoch nicht inhaltlich beschrieben. Hier soll nur ein kleiner Überblick gegeben werden, welche Techniken es gibt. Dies dient dem Zweck, die in dieser Arbeit weiter behandelten Themen ihrer Herkunft bzw. ihrer thematischen Zugehörigkeit zuzuordnen.

Agile Techniken repräsentieren einen Baukasten um agile Vorgehensweisen (aber auch klassische) wie Scrum, Kanban, etc. zu erweitern. Sie lassen sich wie folgt gruppieren [4, p. 30]:

- Agile Techniken:
    - Stand-up-Meeting
    - Planning Game
    - Collective Ownership
    - Continuous Integration
    - Testgetriebene Entwicklung
    - Refactoring
    - Pair Programming
- Management Techniken:
    - Daily Scrum
    - Burn-down Chart
    - Definition-of-Done
    - Sprint Review
    - Retrospektive
    - Servant Leadership
- Planungstechniken:
    - Planning Poker
    - Planungs-Kick-off-Meeting
    - Sprint-Planung
    - Minimal Viable Product (MVP)
    - User-Storys und Akzeptanztests
    - Backlog Grooming
- Softwareauslieferungsprozess:
    - Continuous Delivery
    - DevOps

Diese Arbeit vertieft sich in den Themen Continuous Delivery, Continuous Integration und DevOps weiter.

4 Geänderte Anforderungen an den IT-Betrieb durch Cloud-Technologien

IT-Abteilungen sehen sich immer mehr damit konfrontiert, ihre IT Services stärker an den Business-Anforderungen auszurichten. Dies führt dazu, dass die IT-Organisationen nicht mehr als reine Kostenstellen betrachtet werden, sondern als Business Enabler. Dies bedingt aber auch einen hohen Standardisierungsgrad bei den Basisdiensten (Datenbanken, virtuelle Server, etc.) und einen sehr hohen Automatisierungsgrad, um diese Basisdienste schnell verfügbar zu machen. Somit liegt die Kernaufgabe der IT-Organisationen nicht mehr nur auf der Bereitstellung der Basisdienste, sondern vielmehr auf der Analyse von Business-Anforderungen um daraus neue Standarddienste zu entwickeln bzw. bestehende zu ergänzen. [8, pp. 9-10]

Zusätzliche Trends wie die stark personalisierte und schnelle Kommunikation, die auf Smartphones und mobilen Geräten basiert, erzeugen weitere Anforderungen für die IT. Die meist auf Webapplikationen basierenden Softwareprodukte haben spezielle Anforderungen, da sie direkt von Endbenutzerinnen und Endbenutzern verwendet werden. Sie müssen einfach zu bedienen und hoch performant sein, und Änderungen aufgrund von Kundenwünschen oder auf Grund des Marktdrucks müssen rasch umgesetzt werden können. Somit sind schnellere Release-Zyklen und die Einbindung von Cloud-Technologien wichtiger geworden. [8, p. 10], [9, p. 4]

Dies fasst auch Greg Hughes in seinem Artikel [10] sehr gut zusammen:

> *„[…] Balancing time-to-market pressures with regulatory needs and business continuity demands is a challenge for highly regulated large enterprises. Automating processes and mastering proven practices of release management makes developing and releasing software predictable, reliable, and repeatable. […]“ [10]*

Zum Thema Cloud-Technologien kann ergänzt werden, dass sie sich in den vergangenen Jahren einer wachsenden Beliebtheit erfreuen konnten. Ein Grund für das gestiegene Interesse war mitunter die Vielzahl von öffentlichen Services, die für jedermann zugreifbar waren und auch weiterhin sind. Die Nutzung von Fremdhardware bzw. Fremdressourcen ist eine attraktive Einsparungsmöglichkeit. So muss weniger Hardware beschafft werden und in weiterer Konsequenz auch weniger qualifiziertes Personal zur Betreuung dieser Hardware beschäftigt werden. Kleinen Unternehmen verschafft dieser Zukauf von IT-Dienstleistungen Flexibilität in Bezug auf Wachstum. Werden an einem Tag mehr Ressourcen benötigt, so können sie genau für diesen Zeitraum zugekauft und anschließend wieder zurückgegeben werden. Die Kosten können dadurch an die tatsächlich notwendigen Aufwände angepasst werden. Der Mehrwert von Cloud-Diensten ist allerdings nicht nur in der reinen Abstrahierung von der Hardware zu suchen, denn durch die Einführung von Cloud-Diensten sind auch Themen wie On-Demand Provisioning verfügbar geworden. Dies bedeutet nichts anderes als die Automatisierung des Provisioning-Prozesses, sodass wenige bis keine manuellen Eingriffe mehr notwendig sind. [11, pp. V-VI]

4.1 Cloud Computing

In dieser Arbeit spielt Cloud Computing als eine Cloud-Technologie eine wesentliche Rolle. [12]
Cloud Computing ist ein Modell um Ressourcen wie z.B. Server, Storage, Netzwerke, Anwendungen und Services ohne manuelle Eingriffe on-demand der Benutzerin bzw. dem Benutzer zur Verfügung zu stellen. Dieses Modell setzt sich aus fünf Charakteristiken, drei Servicemodellen und vier Deploymentmodellen zusammen. [13, p. 2]

Charakteristiken von Cloud Computing [13, p. 2]:

- On-Demand Self-Service:

  Kundinnen und Kunden eines Serviceproviders können zu jederzeit selbständig Computing-Ressourcen provisionieren ohne manuelle Eingriffe seitens des Serviceproviders.

- Broad Network Access:

  Die Ressourcen müssen über das Netzwerk mittels standardisierter Mechanismen und über verschiedenste Plattformen (Smartphone, Tablet, Laptop, Workstation, …) erreichbar sein.

- Ressource Pooling:

  Die Ressourcen des Serviceproviders sind in einem Pool zusammengefasst und stehen verschiedenen Kundinnen und Kunden über ein Multi-Tenant-Modell[3] zur Verfügung. Dabei ist die Lokation der Ressourcen völlig irrelevant und die Kundin bzw. der Kunde hat darüber weder Kenntnis noch Kontrolle. Die Lokation ist für die Kundin bzw. den Kunden nur auf einer abstrakteren Ebene wählbar, wie z.B. ein Land oder ein bestimmtes Rechenzentrum.

[3] Multi-Tenancy-Architekturen sind Architekturen bei denen mehrere Kundinnen bzw. Kunden auf derselben Plattform bzw. Infrastruktur arbeiten. Die Anwendungen werden hier virtuell geteilt. [75]
Tenant = Mandant

- Rapid Elasticity:

  Ressourcen können online und dynamisch provisioniert und wieder freigegeben werden. In bestimmten Fällen auch automatisch gemessen an dem aktuellen Bedarf. Aus Benutzersicht sind die Ressourcen unendlich und jederzeit in beliebiger Menge zu provisionieren.

- Measured Service:

  Cloud Systeme werden in der Regel auf Pay-per-Use[4]-Basis abgerechnet. Dies geschieht auf Basis definierter Metriken wie z.B. Speicherbedarf, Processing Time, Bandbreite, aktive Benutzerinnen bzw. Benutzer, etc. Diese Ressourcen werden überwacht, kontrolliert und in Berichten aufbereitet. Dies hat den Vorteil der völligen Transparenz sowohl für die Kundin bzw. den Kunden als auch für den Serviceprovider.

Servicemodelle von Cloud Computing [13, pp. 2-3]:

- Software as a Service (SaaS):

  Die Anwendungen, welche die Kundin und der Kunde verwendet, werden auf der Cloud-Infrastruktur des Serviceproviders ausgeführt. Die Kundin und der Kunde verwaltet hierbei allerdings nicht die darunter liegende Cloud-Infrastruktur. Ihm wird lediglich ein eingeschränktes Set an Einstellungsmöglichkeiten innerhalb der angebotenen Anwendung zur Verfügung gestellt.

- Platform as a Service (PaaS):

  Die Anwendungen, welche die Kundin und der Kunde verwendet bzw. die er selbst durch die vom Serviceprovider bereitgestellten Bibliotheken, Services, APIs, etc. erstellt hat, werden auf der Cloud-Infrastruktur des Serviceproviders ausgeführt. Auch hier wird die darunter liegende Infrastruktur nicht von der Kundin bzw. vom Kunden verwaltet. Der Kundin bzw. dem Kunden gehört allerdings die Kontrolle über die von ihm ausgerollten Anwendungen und ggf. Konfigurationsmöglichkeiten für die Hosting-Umgebung.

- Infrastructure as a Service (IaaS):

  Die Kundin und der Kunde kann selbstständig Ressourcen wie CPU, Storage, Netzwerk, etc. provisionieren. Auf diesen erstellten virtuellen Maschinen kann er selbstständig beliebige Software installieren (einschließlich des Betriebssystems). Die Kundin bzw. der Kunde verwaltet auch hier nicht die darunter

[4] https://www.itwissen.info/Pay-per-Use-pay-per-use.html

liegende Cloud-Infrastruktur, ihm wird ein entsprechendes Interface zur Verfügung gestellt, wo die gewünschten Provisionierungen durchführen werden können.

Deploymentmodelle von Cloud Computing [13, p. 3]:

- Private Cloud:

  Die Cloud-Infrastruktur steht einer Organisation exklusiv zur Verfügung und bedient dabei meist verschiedenste Geschäftseinheiten innerhalb dieser Organisation. Sie steht dabei im Eigentum dieser Organisation oder wird als Dienstleistung zugekauft und wird entweder on- oder off-premises betrieben.

- Community Cloud:

  Die Cloud-Infrastruktur steht einer bestimmten Kundengruppe aus verschiedenen Kundenorganisationen zur Verfügung. Diese Gruppen haben dann ähnliche Anforderungen; was Compliance, Security, Policys, etc. betrifft. Sie wird dabei von einer oder mehreren Organisationen in dieser Kundengruppe betrieben oder wird ebenfalls als Dienstleistung zugekauft.

- Public Cloud:

  Die Cloud-Infrastruktur steht der Öffentlichkeit zur Verfügung und wird von einem Serviceprovider innerhalb seiner Rechenzentren betrieben.

- Hybrid Cloud:

  Der Typ der Cloud-Infrastruktur ist eine Kombination von zwei oder mehreren Cloud-Infrastruktur-Deploymentmodellen (private, community oder public). Die einzelnen Cloud-Infrastrukturen bleiben dabei unabhängige Einheiten, sind allerdings in der Lage über standardisierte oder proprietäre Protokolle Daten untereinander auszutauschen. (Bsp.: Cloud Bursting, etc.)

4.2 Microservices und Container

Ziel von Microservices ist es, große und komplexe Applikationen in kleinere und überschaubarere Teile zu zerlegen. Diese kleineren Einheiten können einfacher ausgetauscht und gewartet werden. Darüber hinaus können auch die Entwicklerteams parallel an unterschiedlichen Microservices arbeiten, ohne sich gegenseitig zu blockieren, und die Entwicklerinnen und Entwickler haben mehr Kontrolle über das Design dieser abgeschlossenen Einheiten. Das ist gerade bei großen Projekten mit vielen Entwicklerinnen und Entwicklern sehr hilfreich. Neue Funktionen oder Techniken können einfacher getestet werden, da hierfür nicht die gesamte Applikation verändert werden muss, sondern lediglich der betroffene Microservice ausgetauscht werden muss. [14]

Um die flexible Wartbarkeit von Microservices zu ermöglichen, ist allerdings eine gute Schnittstellendefinition erforderlich. Solange Änderungen an einem Microservice keine Auswirkung auf die Schnittstellendefinitionen haben, können diese sehr schnell durchgeführt werden. Wenn sich allerdings dabei diese Definitionen ändern, so hat das Auswirkungen auf mehrere oder auch auf alle Services. [15]

Microservices werden nach Umfragen von [16] an erster Stelle dafür verwendet, um einfach skalierbare Applikationen zu schreiben. An zweiter Stelle werden sie eingesetzt, um ein schnelleres Deployment zu ermöglichen, danach folgen Gründe wie die Fokussierung der Entwicklerteams auf bestimmte Teile der Anwendung und das Ermöglichen einer einfacheren Fehlersuche durch gezielteres Eingrenzen von Bugs auf die verursachenden Microservices.

Jeder dieser Microservices läuft in seinem eigenen Prozess und kommuniziert mit den anderen Services über Protokolle wie z.B. http (Hypertext Transfer Protocol) und WebSockets. Diese Art von Architektur erleichtert bzw. ermöglicht erst agile Techniken wie Continuous Integration, Continuous Deployment, die in den folgenden Kapiteln näher erläutert werden. [15]

Abbildung 5 vergleicht grafisch eine traditionelle Anwendung und eine auf Microservices basierenden Anwendung. Traditionelle Anwendungen sind eher als ein Monolith designt. Diese können nur skalieren, indem die gesamte Einheit, auf der sie ausgeführt werden, vervielfacht wird. Diese Einheit kann ein Server, eine virtuelle Maschine oder ein Container sein. Beim Microservice-Ansatz besteht die Anwendung aus unabhängigen Microservices, die auf verschiedenen Servern ausgeführt werden, da sie sich ohnedies über das Netzwerk austauschen.

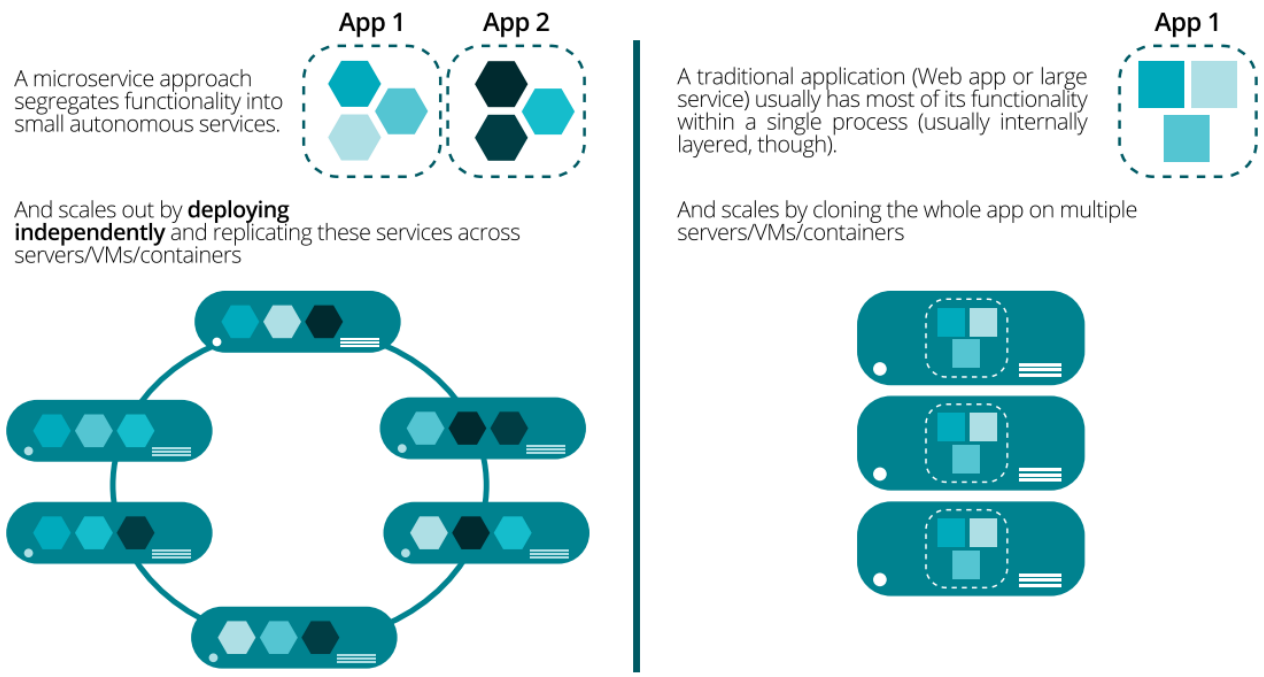

Abbildung 5: Microservices-Architektur vs. traditionelle Architektur [17, p. 3], [15]

Eine Erkenntnis jener Teams, die bereits mit DevOps oder Continuous-Delivery-Methoden arbeiten, besteht darin, dass sich der Einsatz von Microservices und Containern trotz des hohen Trainings- und Setupaufwandes insofern lohnt, als diese in der Lage sind, Code schneller zu deployen. [16]

Ein weiterer spannender Aspekt ist die Einordnung von Containern in die Servicemodelle von Cloud Computing. Dabei werden die folgenden Servicemodelle IaaS und PaaS betrachtet, die in den vorherigen Kapiteln bereits näher erläutert wurden. Grundsätzlich kann man sagen, dass einige PaaS-Umgebungen bereits auf Container setzen, um Multi-Tenant-Applikationen auf einer Shared Infrastructure betreiben zu können. [18]
Setzt man auf Container und IaaS in Kombination mit Orchestrationslösungen[5], erhält man eine Abstraktionsschicht zwischen der Applikation und der darunterliegenden Infrastruktur, was aber mit einer höheren Gesamtkomplexität bezahlt werden muss. [18]
Somit sind auch neue Produkte auf dem Markt erschienen, wie zum Beispiel Container as a Service (CaaS). Dieses Servicemodell ist dabei zwischen IaaS und PaaS einzuordnen und bietet somit eine fertige Umgebung inklusive Funktionen wie Orchestrierung, Hochverfügbarkeit, Image Repository und Continuous Integration/Continuous Deployment Pipelines an. [19]
CaaS ist somit für die Kundin bzw. den Kunden weniger Komplex als IaaS mit Container und Orchestrierung, bietet aber mehr Flexibilität als PaaS.

---

[5] Kubernetes: https://kubernetes.io/
Vgl. „Heat“ für OpenStack in Kapitel 4.3.
Cloudify (OpenStack, VMware, AWS): https://cloudify.co/

Für den Einsatz von Microservice-Architekturen sind nicht nur die technischen Aspekte zu betrachten, sondern auch die organisatorischen. Die Team- bzw. Unternehmensorganisation muss für diese Art von Architektur entsprechend aufgestellt sein. Konkret ist die Organisation von Feature-Teams sinnvoll. In diesen Teams sind unterschiedliche Skills zusammengewürfelt, die sich um die Implementierung eines bestimmten Features kümmern. Dabei sollte der durch die Teamzerteilung verursachte Koordinationsaufwand nicht unterschätzt werden. [20]

Als konkretes Beispiel für eine Container Engine soll Docker nachfolgend beschrieben werden. Docker bietet die Möglichkeit, Anwendungen in eine losgelöste Umgebung (Container) zu verpacken [21], und besteht aus den folgenden Komponenten, die in Abbildung 6 dargestellt sind:

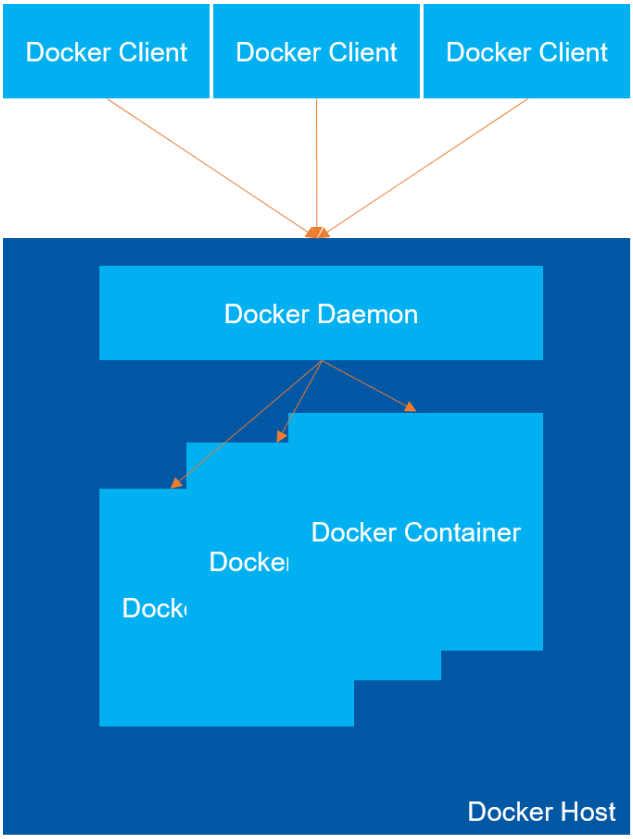

Abbildung 6: Docker-Architektur [22, p. 10]

- Docker Engine (Docker Client und Server) [22, p. 9]:

  Docker ist eine Client-Server-Anwendung, wobei der Docker Client mit dem Docker Daemon kommuniziert. Der Docker Daemon kann über den mitgelieferten Docker Command Line Client oder eine Restful API angesprochen werden.

- Docker Images [22, p. 11], [21]:

  Images sind Read-Only-Templates, welche alle notwendigen Instruktionen und Daten enthalten, um den Docker Container ausführen zu können. Images können auf andere Images aufbauen.

- Registrys [22, p. 11], [21]:

  Zum Speichern der Docker Images gibt es Registrys. Docker Hub ist eine öffentliche Plattform für das Speichern solcher Images.

- Docker Container [22, p. 12], [21]:

  Ein Container ist eine laufende Instanz eines Image und kann über die Docker API oder CLI verwaltet werden.

4.3 OpenStack

OpenStack ist eine Cloud-Plattform, die als gemeinsames Projekt zwischen der NASA und Rackspace entstanden ist. Der Grundgedanke war eine Open-Source-Alternative mit Schnittstellen zu Amazon Elastic Compute Cloud (Ec2) zu schaffen. Technisch bietet OpenStack unter anderem die Möglichkeit der Virtualisierung von Compute-, Storage- und Network-Ressourcen. Mittlerweile ist OpenStack zu einem Keyplayer in der Cloud-Plattform Industrie geworden. [11, p. V] Abbildung 7 zeigt die historische Release-Übersicht seit dem ersten Release im Jahr 2010.

Series	Status	Initial Release Date	Next Phase	EOL Date
Stein	Future	2019-04-11 *estimated* (schedule)	Development *estimated 2018-08-30*	
Rocky	Development	2018-08-30 *estimated* (schedule)	Maintained *estimated 2018-08-30*	
Queens	Maintained	2018-02-28	Extended Maintenance *estimated 2019-08-25*	
Pike	Maintained	2017-08-30	Extended Maintenance *estimated 2019-03-03*	
Ocata	Maintained	2017-02-22	Extended Maintenance *estimated 2018-08-27*	
Newton	End Of Life	2016-10-06		2017-10-25
Mitaka	End Of Life	2016-04-07		2017-04-10
Liberty	End Of Life	2015-10-15		2016-11-17
Kilo	End Of Life	2015-04-30		2016-05-02
Juno	End Of Life	2014-10-16		2015-12-07
Icehouse	End Of Life	2014-04-17		2015-07-02
Havana	End Of Life	2013-10-17		2014-09-30
Grizzly	End Of Life	2013-04-04		2014-03-29
Folsom	End Of Life	2012-09-27		2013-11-19
Essex	End Of Life	2012-04-05		2013-05-06
Diablo	End Of Life	2011-09-22		2013-05-06
Cactus	End Of Life	2011-04-15		
Bexar	End Of Life	2011-02-03		
Austin	End Of Life	2010-10-21		

Abbildung 7: OpenStack Releases [23]

Die meisten OpenStack Services sind in Python geschrieben. Alle OpenStack Services bieten REST APIs[6], über die sie durch Benutzerinnen und Benutzer oder durch andere Services angesprochen werden können. Die Komponenten eines Service kommunizieren grundsätzlich über eine Message Queue. Das bringt durchaus Vorteile,

[6] Vgl. „REST APIs“ auf Seite 36.

denn so können Anfragen bis zu ihrer Verarbeitung in einer Warteschlange geparkt und entsprechend lastverteilt werden. [24, p. 13]

Exkurs zu REST APIs: REST steht für Representational State Transfer und API für Application Programming Interface. Dabei geht es um die Kommunikation innerhalb von verteilten Systemen. REST basiert dabei auf einem Client-Server-Modell und es ist zustandslos. Das bedeutet, dass Client und Server ihre Anfragen bzw. Antworten immer vollständig abschließen müssen, da auf keinen vorherigen Kontext zurückgegriffen werden kann. Der große Vorteil liegt unter anderem in einer einheitlichen Schnittstelle, die REST-konforme Services nutzen. Diese Schnittstelle ist vom implementierten Dienst losgelöst und vereinfacht somit die Architektur. Dafür müssen die Informationen aber in ein standardisiertes Format gebracht werden und können somit nicht an die Bedürfnisse der Anwendung angepasst werden. Die Datenübertragung erfolgt dabei meist über http/https (Hypertext Transfer Protocol/Hypertext Transfer Protocol Secure). [25]

Abbildung 8 zeigt eine Übersicht über die Funktionen von OpenStack. Nachfolgend werden die Kernfunktionen näher beschrieben, diese sind in Abbildung 8 mit einem blauen Pfeil markiert.

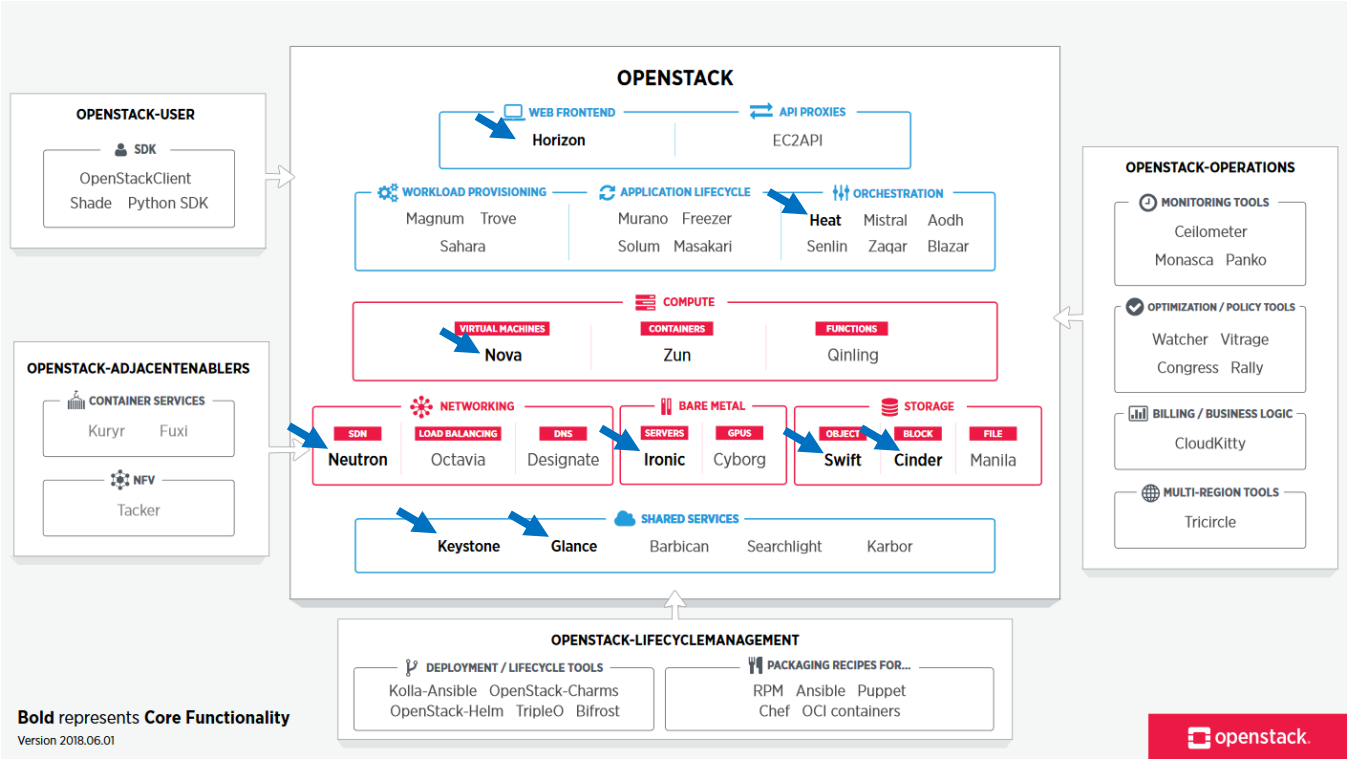

Abbildung 8: OpenStack Map [26]

Keystone – Identity Management [27], [24, p. 14]:

- Keystone ist der einfachste Service in der OpenStack-Architektur und ist für das Authentifizieren und Autorisieren der einzelnen Tenants und Services zuständig.

- Dabei können zahlreiche Authentifizierungsmechanismen verwendet werden. Beispielsweise Username und Passwort, Token, Lightweight Directory Access Protocol (LDAP) und Pluggable Authentication Module (PAM).

Swift – Object Storage [28], [24, p. 15]:

- Swift ist ein hochverfügbarer und verteilter Object/Blob Store. Object Stores bieten die Möglichkeit, Daten in Form von Objekten abzuspeichern. Swift teilt die Daten in kleine Chunks und speichert sie in unterschiedlichen Containern. Diese Container sind in redundanter Ausführung vorhanden und sind über alle Storage Nodes verteilt, um Hochverfügbarkeit, Auto-Recovery und eine horizontale Skalierung zu ermöglichen.

- Die Vorteile dabei sind, dass es durch die Verteilung keinen Single Point Of Failure (SPOF) gibt.

Cinder – Block Storage [29], [24, pp. 15-16]:

- Cinder ist für den Block Storage in der OpenStack-Architektur zuständig. Diese Storage wird auf Blockebene angeboten und kann als virtuelle Festplatte für virtuelle Maschinen dienen.

- Volumes, die von Cinder angeboten werden können, können geklont werden. Darüber hinaus bieten sie Funktionen wie Snapshots etc. an.

- Cinder kann mehrere Storageprovider ansprechen. Dies ist durch konfigurierbare herstellerspezifische Treiber möglich. So können Storagesubsysteme von IBM, NetApp, Nexenta, Vmware, etc. ebenfalls als Blockstorage-Quelle herangezogen werden.

Ironic – Bare Metal [30]:

- Ironic ist für die Provisionierung von physischer Hardware zuständig. Dies kann mittels PXE oder anderen herstellerspezifischen Remote-Management-Protokollen geschehen.

Glance – Image Registry [31], [24, p. 17]:

- Glance bietet die Images einschließlich ihrer Metadaten an, auf Basis derer virtuelle Maschinen gestartet werden können. Dabei wird eine Reihe von Image-Formaten unterstützt. Beispielsweise KVM/Qemu, XEN, VMware, Docker, etc.

Nova – Compute Service [32], [24, p. 18]:

- Nova ist der zuständige Service für das Provisionieren von Compute-Ressourcen, also von virtuellen Maschinen.

Neutron – Networking Services [33], [24, p. 20]:

- Neutron bietet Network as a Service, Firewall as a Service, Load Balancing as a Service und Virtual Private Network as a Service innerhalb der OpenStack-Architektur.

Heat – Orchestration [34], [24, p. 23]:

- Heat ist der Service, der für die Orchestration innerhalb der OpenStack-Architektur zuständig ist.

- Ein Heat Template beschreibt, welche Ressourcen, wie z.B. Compute, Image, Storage, Netzwerk, etc., für eine Cloudanwendung notwendig sind.

- Für eine Full-Stack-Automatisierung sind Configuration Management Tools wie Chef zu integrieren.

Horizon – Dashboard [35], [24, p. 24]:

- Horizon ist das webbasierte OpenStack Dashboard, über das die OpenStack Services administriert werden können.

- Es wurde als Stateless und Dataless Application designt. Das bedeutet, dass Horizon nur OpenStack API Requests an die Services sendet und deren Rückmeldungen grafisch im Webportal darstellt. Es speichert selbst nur die Sitzungsinformationen und den eigenen Programmcode.

- Horizon kann an die jeweiligen Bedürfnisse eines Serviceproviders angepasst werden.

5 DevOps

Die Idee hinter DevOps ist die Zusammenführung von Development und Operations, also den Entwicklerinnen und Entwicklern, die die Software schreiben, und den für den Betrieb Verantwortlichen, die die Software schlussendlich betreiben. Aus diesen zwei Bereichen setzt sich auch das Kunstwort DevOps (Development und Operations) zusammen. Das Ziel von DevOps ist, kontinuierliche Softwarebereitstellung zu ermöglichen und dadurch schnell auf Marktveränderungen und Kundenwünsche reagieren zu können. [36, p. 189]

DevOps führt dabei agile Methoden wie z.B. Scrum, ein schlankes Management wie z.B. Kanban und IT Service Management basierend auf beispielsweise dem ITIL Framework zusammen. [36, p. 198]

Gründe, warum man sich durchaus mit DevOps befassen sollte, liegen im sich rasant wandelnden Umfeld einer IT-Organisation. Die durchdringende Digitalisierung in vielen Geschäftsbereichen erfordert eine engere Zusammenarbeit zwischen Business und IT. Dies gilt insbesondere im Web- und App-Bereich, der sehr kurze Produktivsetzungszeiten fordert [36, pp. 191-192].

Somit lässt sich das Einsatzfeld von DevOps auch einigermaßen abgrenzen. Sind viele Releasezyklen in kurzer Zeit zu bewältigen, ist DevOps das Mittel der Wahl. Für kritische Applikationen, die ohnedies lange Test- oder Zertifizierungsprozesse durchlaufen müssen, sind konservativere Ansätze die bessere Wahl. Beispiele dafür sind medizinische Anwendungen und Geräte, Anwendungen in der Luftfahrt etc. [37, p. 97]

Ein verbreiteter Irrglaube ist, dass durch diesen Ansatz der Bereich IT Operations völlig eliminiert wird (Stichwort „NoOps“). Das ist allerdings nicht die Intention dieses Ansatzes. Vielmehr verändert sich der Arbeitsfokus des IT-Operations-Teams weg von manuellen Tätigkeiten, die durch Tickets beauftragt werden, hin zur Verfügungstellung von APIs, welche die Entwicklerinnen und Entwickler nutzen können, um ihre Arbeit effizienter (und vor allem automatisierter) durchführen zu können. Das können Self-Service-Plattformen sein, mit denen die Entwicklerinnen und Entwickler ihre Testsysteme aufbauen und überwachen können. Somit arbeitet das IT-Operations-Team bei DevOps ähnlich wie die Entwicklerinnen und Entwickler selbst, indem sie ihr Produkt (die Plattform selbst) ständig weiterentwickeln. [38, p. 14]

Historisch entwickelte sich DevOps unter anderem aus dem agilen Manifest, welches bereits in Kapitel 3 behandelt wurde, aus Lean Thinking, das in diesem Kapitel noch näher erläutert wird und aus Methoden wie Infrastructure as Code und Continuous Delivery, welche ebenfalls noch beschrieben werden. [38, pp. 44-47]

5.1 DevOps-Kultur

Mit der starken Interaktion der beiden Bereiche Operations und Development gehen auch organisatorische Änderungen einher. Diese Änderungen sind notwendig, um eine DevOps-Kultur zu schaffen. [36, p. 193]
Das Ziel dieser Kultur ist es, die Zusammenarbeit zwischen Development und Operations zu verbessern. So können Entwicklerinnen und Entwickler ein besseres Infrastrukturverständnis aufbauen und die Mitarbeiterinnen und Mitarbeiter aus den Bereich Operations ein besseres Verständnis für den entwickelten Code erlangen. [36, p. 193]

Die Aspekte einer DevOps-Kultur sind nach [36, pp. 193-195] die folgenden:

- Kontinuierliches Lernen:

  Die Mitarbeiterinnen und Mitarbeiter einer DevOps-Organisation haben das Verlangen, jede Aktivität zu verstehen und inhaltlich zu begreifen. Dafür ist es absolut notwendig, das Wissen in den einzelnen Bereichen zu teilen, um hier für Transparenz zu sorgen. Dabei entsteht ein Verständnis der Gesamtzusammenhänge innerhalb der Wertschöpfungskette, an der man beteiligt ist. Dies kann durch das Erstellen von Prototypen realisiert werden, anhand derer das weitere Vorgehen im Entwicklungsprozess besprochen wird.

- Experimentieren:

  Neue Ideen werden anhand von Experimenten bewertet. Dafür muss ausreichend Zeit, Technik und Budget zur Verfügung gestellt werden und eine Kultur etabliert sein, die ein Scheitern akzeptiert. Beispielsweise kann diese Zeit zur Automatisierung von manuellen Arbeitsschritten herangezogen werden. Ein wichtiger Aspekt dabei ist, auch Zeit und Raum zum Austauschen zur Verfügung zu stellen und neue Ideen in einem dokumentierten Prozess zu behandeln.

- Eingebaute Qualität:

  Qualität ist ein wichtiger und zentraler Aspekt. Er beeinflusst alle Aktivitäten und führt zu Transparenz in den Arbeitsschritten. Dies kann dadurch erreicht werden, dass alle Beteiligten für das Gesamtergebnis verantwortlich sind und nicht nur für Teilschritte. Dies kann auch in einer Vereinbarung niedergeschrieben werden.

- Ingenieurskultur:

  Die beteiligten Mitarbeiterinnen und Mitarbeiter werden nach ihrem Wissen und ihren Kenntnissen bewertet und sind stets bestrebt, Probleme zu lösen, hilfreiche Produkteigenschaften zu schaffen, ihre Fertigkeiten zu verbessern und Betriebsprozesse zu optimieren. Dafür setzen sie ihr Wissen, ihre Fähigkeiten und ihre Kreativität ein.

- Kultur der Effektivität:

  Hierbei geht es im Wesentlichen darum, sinnvolle Produkteigenschaften zu liefern und das eigene Verhalten gemäß der DevOps-Kultur entsprechend zu gestalten. Dies kann dadurch erreicht werden, dass den Teams anstelle von starren Prozessvorgaben mehr Verantwortung für Entscheidungen gegeben wird.

- Produktdenken:

  Dadurch wird das klassische Applikationsdenken erweitert. Das Produkt ist im Sinne von DevOps die betreute Applikation und der schlussendlich zur Verfügung gestellte Service, der für die Nutzerinnen und Nutzer einen Mehrwert bringen muss. Daher ist es auch sinnvoll, Anwenderinnen und Anwender frühzeitig in den Entwicklungsprozess miteinzubeziehen, um frühzeitig Feedback zu den Entwicklungsschritten zu erhalten.

- Übernahme von Verantwortung:

  Entscheidend ist, dass sich alle Mitglieder des Teams für den Developmentprozess und den Operationsprozess verantwortlich fühlen.

Um eine DevOps-Kultur nach den geschilderten Aspekten zu entwickeln, bedarf es eines Kulturaufbaus, der durch ein Organisationsentwicklungsprojekt begleitet wird und in der täglichen Führung gelebt wird. [36, p. 194] Die geschilderten Aspekte der DevOps-Kultur haben auch Veränderungen in der Teamorganisation zur Folge. So muss vom traditionellen Silo-Denken, wie in Abbildung 10 dargestellt, Abstand genommen werden und vermehrt eine Ausrichtung am Produkt, wie in Abbildung 9 dargestellt, bzw. an der Arbeit selbst erfolgen. Somit bezieht sich die Arbeit nicht mehr auf Einzelne, sondern auf Teams. [36, pp. 195-196]

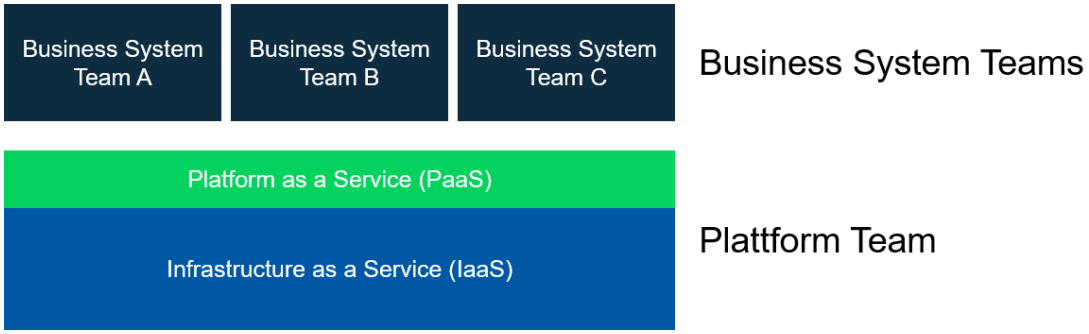

Abbildung 9: DevOps-Teamorganisation [36, p. 196]

Um dieses Silo-Denken zu minimieren, ist viel Zeit und Arbeit notwendig, in der den beteiligten Stellen eine andere Arbeitsweise schmackhaft gemacht werden soll. Ein Ansatz dafür ist, von Beginn an die beteiligten Stellen miteinzubeziehen. Darüber hinaus ist dafür zu sorgen, dass ein Rahmen geschaffen wird, in dem die beteiligten Stellen regelmäßig miteinander kommunizieren können. Sind die Barrieren dann eingerissen, so ist es wichtig, die Kommunikation zwischen den Bereichen aufrechtzuhalten und kontinuierlich weiterzuführen. [39, p. 28]

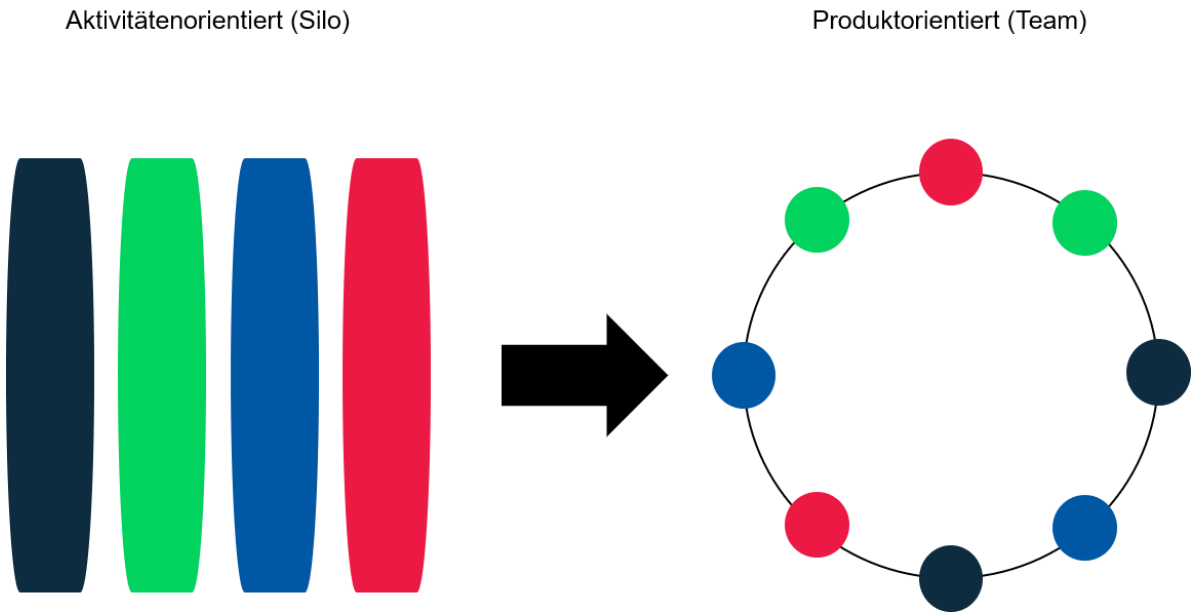

Abbildung 10: Entwicklung von traditioneller Aktivitäten- zur Produktorientierung [36, p. 196]

Dafür ist es hilfreich, ein System zu installieren, in dem alle beteiligten Stellen immer den aktuellen Status der zu liefernden Applikation einsehen können: den Zustand, die aktuellen Builds, welche Tests diese bereits absolviert haben und den Zustand der Umgebung, auf der sie deployt werden können. Diese Plattform sollte auch die Funktionen bieten, damit alle beteiligten Bereiche ihre Arbeit damit erledigen können. [39, p. 28]

Der daraus folgende Einfluss auf die Architektur lässt sich nach [36, pp. 196-197] wie folgt umreißen:
Große Anwendungskomplexe müssen in kleinere Services überführt werden. Dabei wird eine Architektur aus Microservices erschaffen.
Dies führt dazu, dass diese Anwendungskomplexe modularer werden. Der Ansatz von Microservices entstand vor ca. 12 Jahren als Nachfolger von serviceorientierten Architekturen. [37, p. 98]

Einer der großen Vorteile von DevOps ist, dass die Entwickler-Teams und die Qualitätssicherungs-Teams auf produktionsähnlichen Systemen testen können. Somit kann vor dem Deployment auf die Produktionssysteme umfangreich getestet werden, wie sich die aktualisierte Applikation auf diesen verhalten wird. [9, p. 7]

5.2 DevOps Life Cycle

Nun wurde die Idee hinter dem DevOps-Ansatz bereits beschrieben. Als nächstes soll nun näher auf die inhaltlichen Themen von DevOps eingegangen werden. Abbildung 11 zeigt als DevOps-Referenzarchitektur den DevOps Life Cycle und dessen Phasen, die nachfolgend näher beschrieben werden.

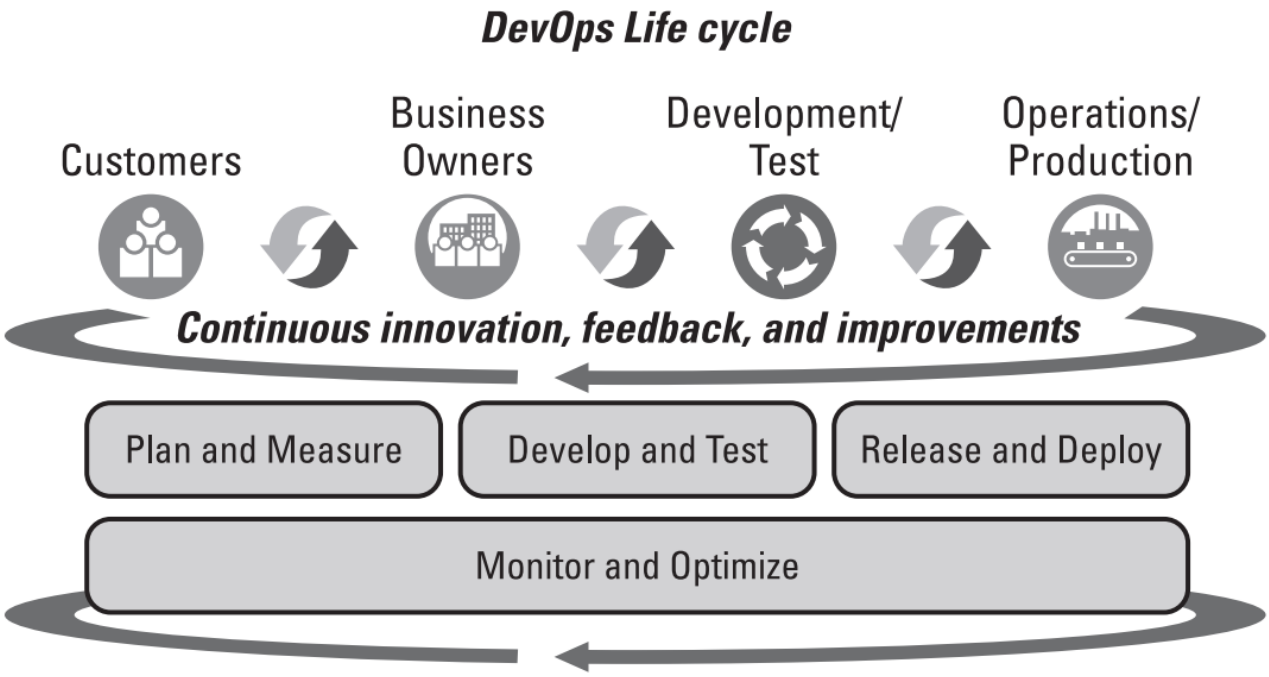

Abbildung 11: DevOps Life Cycle [9, p. 10]

Die Phase Plan and Measure konzentriert sich dabei ganz auf die kontinuierliche Planung des Business. Das Business muss dabei agil sein, damit es rasch auf neue Anforderungen aus Kundenfeedbacks reagieren kann. Dabei werden auch Methoden wie Lean Thinking angewandt. [9, p. 10]

Exkurs zu Lean: Die Philosophie hinter Lean stammt ursprünglich aus der Automobilindustrie. Konkret hat sie ihre Wurzeln im Toyota-Produktionssystem. Im ursprünglichen Kontext ging es um eine schnellere Reaktionsfähigkeit auf Veränderungen der Nachfrage am Markt, um sich dadurch Wettbewerbsvorteile zu verschaffen. Kern der Philosophie ist, jegliche Art von Verschwendung zu vermeiden. Dabei werden nicht nur direkte Verschwendungsarten wie Materialverschwendung untersucht, sondern auch indirekte, wie Überproduktion, Wartezeiten etc. [40, p. 35]
Lean im Kontext der IT bedeutet demnach nichts anderes, als die Lean-Prinzipien auf das IT-Management zu übertragen. Das bedeutet, die Prozesse und Organisation kundenorientiert mit einem kontinuierlichen Verbesserungsprozess aufzustellen und Verschwendungen, wie z.B. nicht vorbereitete Meetings, unnötige Formalismen, ineffiziente Genehmigungsverfahren und unklare Verantwortlichkeiten innerhalb der Prozesse zu eliminieren. [41]

In den Bereich Develop and Test fallen alle Aktivitäten, die einen möglichst kollaborativen Einwicklungs- und Testprozess betreffen und die in der Regel in den Bereichen Development und Qualitätssicherung angesiedelt sind. Beim kollaborativen Entwickeln von Software sind folgende Kompetenzen bzw. Fachbereiche in den Teams notwendig [9, p. 11]:

- Spezialistinnen und Spezialisten aus den Geschäftsfeldern (Fachbereichen), für die die Software entwickelt wird.
- Software-Architektinnen und Software-Architekten
- Entwicklerinnen und Entwickler
- Qualitätssicherungsspezialistinnen und Qualitätssicherungsspezialisten
- Spezialistinnen und Spezialisten aus dem Betrieb
- Security-Spezialistinnen und Security-Spezialisten
- Lieferanten
- Partnerinnen und Partner des Projektes

Die Zusammenarbeit soll dabei über eine gemeinsame Plattform erfolgen, da die einzelnen Mitglieder sehr wahrscheinlich an unterschiedlichen Orten arbeiten und möglicherweise unterschiedliche Zeitzonen haben. [9, p. 11]

Ein wesentlicher Bestandteil von „Release and Deploy“ ist eine gut automatisierte Pipeline (Näheres dazu folgt in den nachfolgenden Kapiteln), die es ermöglicht, die Software automatisiert zu testen und in Produktion zu stellen. [9, p. 12]
Dabei sind Konzepte wie z.B. Infrastructure as Code, welches in Kapitel 5.4 beschrieben ist, wesentlich. Dadurch werden Infrastrukturkomponenten identisch reproduzierbar. Die Konfiguration der Infrastruktur kann mit dem eigentlichen Applikations-Code versioniert werden. Darüber hinaus können Testsysteme anhand dieser Konfiguration schnell automatisiert aufgebaut werden und Fehler, die durch falsche Infrastrukturkonfiguration verursacht werden, entfallen. [37, p. 97]

„Monitor and Optimize“ ermöglicht einerseits die Beurteilung, wie produktive Anwendungen in der Praxis performen, und dient andererseits dazu, Feedback der Kundinnen und Kunden zu erhalten. Durch die kontinuierliche Überwachung werden Messdaten und Metriken an die Betriebsführung, Qualitätssicherung, Entwicklung und weiteren beteiligten Stellen im Softwareentwicklungsprozesse geliefert. [9, p. 13]

In den nachfolgenden Kapiteln werden nun einige zentrale Methoden beschrieben, die für DevOps relevant sind.

5.3 Softwareauslieferungsprozess

Um nun ein klareres Bild über die Methoden, die bei DevOps relevant sind, zu schaffen, werden nun die Begriffe wie Continuous Integration, Continuous Delivery, Continuous Deployment und die Deployment Pipeline erklärt. Abbildung 12 grenzt die Begriffe Continuous Integration, Continuous Delivery und Continuous Deployment voneinander ab und ordnet sie den jeweiligen Phasen im Softwareauslieferungsprozess zu.

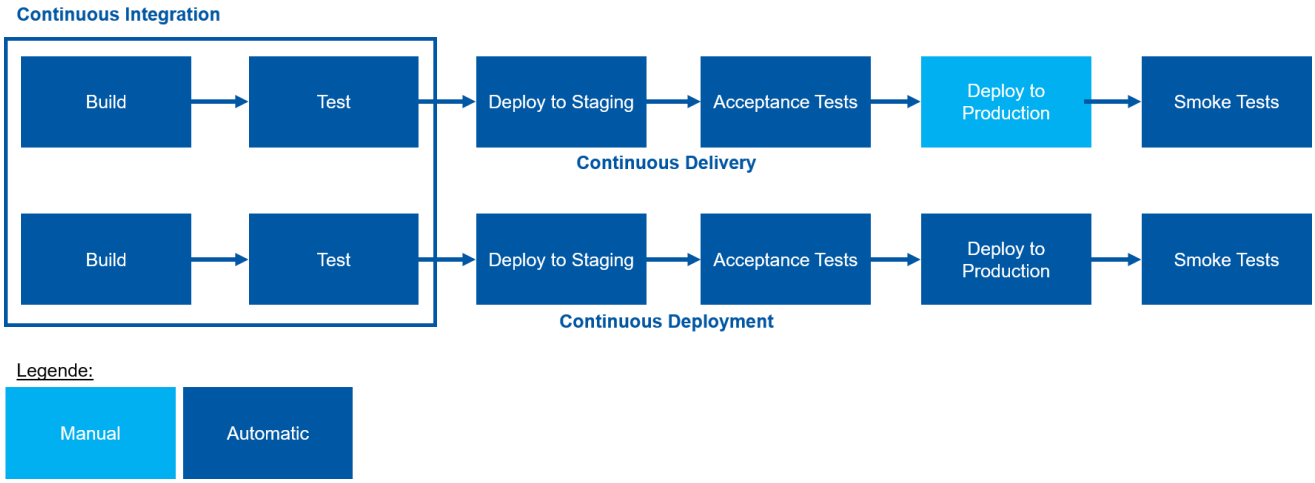

Abbildung 12: Vergleich Continuous Integration, Continuous Delivery, Continuous Deployment [42]

Nachfolgend werden verschiedene Testarten und Begriffe aus Abbildung 12 erläutert:

- Staging-Umgebung:

  Eine Staging-Umgebung ist eine Testumgebung, die der Produktionsumgebung exakt ähnelt. Sie ist somit eine vollständige, aber unabhängige Kopie dieser Umgebung inklusive aller Datenbanken etc. [43]

- Unit Test:

  Unter Unit Tests versteht man das isolierte Testen einer einzelnen Methode oder einer Klasse, um festzustellen, dass der darin entwickelte Code genau das tut, wofür er designt wurde. Dabei werden externe Abhängigkeiten oft durch statische Werte ersetzt. Beispielsweise wird beim Test einer Methode, die Daten aus einer externen Datenbank verarbeitet, die Datenbankabfrage selbst durch statische Werte ersetzt, um die Abfrage zu unterbinden und die Methode isoliert testen zu können. Dies ist notwendig, da diese Tests schnell und mit möglichst keinen Abhängigkeiten zu anderen Teilen der Anwendungen durchgeführt werden sollen. [38, p. 191]

- Acceptance Test:

  Bei einem Acceptance Test wird die Anwendung als Ganzes getestet. Hierbei wird getestet, ob die Anwendung das tut, was die Kundin bzw. der Kunde erwartet. [38, p. 192]

- Integration Test:

  Bei einem Integration Test wird überprüft, ob die Anwendung im Kontext des Gesamtsystems korrekt läuft, also ob sie mit anderen produktiven Anwendungen und Diensten korrekt interagiert. [38, p. 192]

- Smoke Test:

  Smoke Tests prüfen nach dem Deployment einer Anwendung, ob diese auch lauffähig ist. Das können Skripte sein, die die Anwendungen starten und überprüfen, ob die erwarteten Inhalte zurückgegeben werden. Darüber hinaus sollte jeder Service, den die Applikation benötigt, wie z.B. eine Datenbank, ein externer Service etc., überprüft werden, ob auch er läuft. Der Test sollte den Entwicklerinnen und Entwicklern bei Fehlschlägen auch erste diagnostische Daten mitliefern, um die Analyse zu vereinfachen. [39, p. 117]

5.3.1 Continuous Integration

Unter Continuous Integration versteht man im Wesentlichen den Build-Prozess und die darauffolgenden automatisierten Tests, die durch das Einchecken von neuen Code Changes in das Software Repository ausgelöst werden. Dies sorgt dafür, dass schnell klar ist, ob neue Codeänderungen einer Komponente im Gesamtkontext der Anwendung lauffähig sind. Durch das häufige Testen ist außerdem die Zuordnung des Fehlers zur verursachenden Änderung leichter herzustellen. [39, pp. 55-56]
Nach Abbildung 12 umfasst sie die Phasen Build und Test.
Continuous Integration ist jedenfalls eine Methode und kein Werkzeug. Sie lebt davon, dass sich die beteiligten Entwicklerteams an diese Methodik halten, indem sie kleine Änderungen am Code diszipliniert in das Code Repository einchecken und Fehler mit höchster Priorität korrigieren, da sonst keine weiteren Builds erstellt werden können und andere Entwicklerinnen und Entwickler somit kein Feedback auf ihre Änderungen erhalten. Demnach sind auch keine komplexen Werkzeuge notwendig, wenn auch viele Entwicklungsumgebungen spezielle Unterstützungen dafür bieten. [39, p. 57]
Der Vorteil dabei ist, dass zu einem sehr frühen Zeitpunkt im Softwareentwicklungszyklus mit dem Testen begonnen wird und Fehler einfacher und kostengünstiger korrigiert werden können, als dies später der Fall wäre. Sinnvoll ist es auch, diese Tests in einer sehr produktionsnahen Umgebung durchzuführen, um die Aussagekraft zu erhöhen. [9, p. 12]

Ein zentraler Bestandteil von Continuous Integration ist das Versionskontrollsystem (Repository). Dieses enthält gesammelt den gesamten Code, alle Tests, Datenbank-Skripte sowie die Build- und Deployment-Skripte. Somit enthält es alles was für das Erstellen, Installieren, Ausführen und Testen der Applikation notwendig ist. Die Build-Prozesse selbst müssen sich für Continuous Integration automatisieren lassen. [39, pp. 56-57]
Aus historischer Sicht entwickelte sich Continuous Integration aus den folgenden Methodiken [39, p. 65]:

- Nightly Builds:

  Die Idee hinter Nightly Builds ist, dass über nächtliche Batch-Prozesse die Code-Basis kompiliert und integriert wird. Das geht in den Ansätzen schon in Richtung Continuous Integration, nur haben Nightly Builds einen großen Nachteil. Treten bei der nächtlichen Kompilierung Fehler auf, so sind die Teams am nächsten Tag mit der Fehlersuche und -korrektur beschäftigt. Sind die Fehler dann korrigiert, muss wieder bis zur nächsten Nacht gewartet werden, um zu prüfen, ob die Korrekturen zweckbringend waren. Dies verzögert die Zeit, bis neue Änderungen getestet werden können. Darüber hinaus ist dies eine schwierige Methodik, wenn Teams in verschiedenen Zeitzonen arbeiten und diese aber kollaborieren müssen. Im Laufe der Zeit wurde diese Methodik noch um automatisierte Tests ergänzt.

- Rolling Builds:

  Der nächste Entwicklungsschritt waren die Rolling Builds. Hier werden anstelle von zeitgesteuerten Batch Jobs, kontinuierliche Jobs verwendet, die sofort nach Fertigstellung des laufenden Build-Vorganges den nächsten starten, mit der zu diesem Zeitpunkt eingecheckten aktuellen Code-Basis. Das Problem der Rolling Builds ist, dass die Entwicklerinnen und Entwickler zwar rascher ein Feedback bekommen und nicht bis zum nächsten Tag warten müssen, es aber keinen Zusammenhang zwischen eingechecktem Code und laufendem Build gibt. Daher muss erst wieder nach dem defekten Codesegment gesucht werden, das den Fehler verursacht hat. Daraus entwickelte sich schlussendlich auch der Ansatz der Continuous Integration.

Im Vergleich zu Softwareprojekten, bei denen die Integration Tests am Schluss der einzelnen Entwicklungsphasen der verschiedenen Komponenten durchgeführt werden, kann mit Continuous Integration insofern Zeit eingespart werden, als mögliche Probleme beim Zusammenführen der Einzelkomponenten zum Gesamtsystem früher erkannt werden können. Somit können auch Korrekturen sofort durchgeführt werden und nicht in einer späteren Phase eines Softwareentwicklungsprozesses, in der Änderungen weit aufwändiger und somit auch kostenintensiver werden. Dies setzt allerdings auch das Vorhandensein von automatisierten aussagekräftigen Tests voraus. [39, pp. 55-56]

5.3.2 Continuous Delivery

Continuous Delivery geht nach Abbildung 12 weiter als Continuous Integration.

Continuous Delivery ist eine der zentralen Methoden im DevOps-Ansatz und ist nach [36, p. 200] wie folgt definiert:

> *„[…] ● verlegt die Verantwortung und die Steuerung des Releaseplans weg von der IT hin zum Business,*
>
> *● sorgt für eine permanente Auslieferungsfähigkeit innerhalb der Entwicklung, d. h. dafür, dass jede Änderung prinzipiell ad-hoc in die Produktivumgebung gebracht werden können muss und*
>
> *● legt fest, dass die Arbeit der Entwickler erst beendet ist, wenn ihr Code in Produktion ist und nicht, wenn sie ihn an die Tester oder zur Qualitätssicherung übergeben haben. […]“ [36, p. 200]*

Um dieser Definition zu genügen, ist es erforderlich, einen hohen Automatisierungsgrad in allen Bereichen des DevOps Life Cycle zu erreichen. Somit ist Automatisierung in DevOps ein zentraler und wichtiger Bestandteil und ist die wesentliche Voraussetzung für Continuous Delivery. Dabei kommt Automatisierung in den folgenden Bereichen zum Einsatz: Build, Test, Deployment und Release. [36, p. 200]

Um den notwendigen Grad an Automatisierung zu erreichen, ist es ideal, wenn dies durch Skripte und Konfigurationsinformationen geschieht, die in einem Repository zur Versionskontrolle gespeichert sind. Der Aufbau dieser Komponenten durch die gespeicherten Skripte und Informationen muss dabei auf Self-Service-Basis ohne Zutun der Operation Teams möglich sein. So können gesamte Produktionsumgebungen auf Basis dieser Skripte und Informationen automatisiert neu erstellt werden. [38, p. 170]
Durch diese Art von Deployment wird ein möglicherweise schlecht dokumentiertes manuelles Aufbauen der einzelnen Infrastruktur- und Software-Komponenten mittels Tickets an den Operations-Bereich verhindert. Somit ist auch ein sauberer Systemzustand, den die beteiligten Bereiche gemeinsam erarbeitet haben, in Form von Code dokumentiert und existiert somit nicht ausschließlich in Dokumenten oder in den Köpfen der beteiligten Personen. Darüber hinaus ist diese Konfigurationsbasis in den automatischen Build-Prozess integriert. [38, p. 173]
Änderungen an den Infrastrukturkonfigurationen können ebenso einfach getestet werden und bei positiver Absolvierung dieser Tests in Produktion genommen werden.

Das so entstehende „Single Repository Of Truth“ enthält somit alle wichtigen Konfigurationsdateien, Software Installer und Dokumentationen und steht allen Bereichen wie Developement, Qualitätssicherung, Operations und Infosec zur Verfügung. Dies gewährleistet den schnellen und zuverlässigen Aufbau aller Komponenten einer

Umgebung inklusive der Anwendungen, der Produktionsumgebung und den entsprechenden Testumgebungen. [38, pp. 174-175]

Denkt man diesen Ansatz des zentralen „Single Repository Of Truth“ weiter, so kommt man zur Frage, wie im täglichen Betrieb sichergestellt werden kann, dass Konfigurationsänderungen nicht direkt auf produktiven Maschinen bzw. Systemen erfolgen, sondern nur über das Repository, das die Konfigurationen enthält.
Maßnahmen, um direkte Konfigurationsänderungen zu reduzieren, können unter anderem darin bestehen, Logins auf Produktionssystemen generell zu verbieten bzw. einzuschränken oder produktive Instanzen regelmäßig neu aufzubauen. [38, p. 179]
Der am höchsten entwickelte Reifegrad wird in der Literatur „Immutable Infrastructure“ genannt. Bei solchen Infrastrukturen sind alle manuellen Änderungen an produktiven Systemen verboten. Die einzige Möglichkeit Änderungen durchzuführen, führt über das zentrale „Single Repository Of Truth“ und das anschließende Neuaufbauen der betroffenen Komponenten. Dadurch ist es nicht möglich, dass Konfigurationsabweichungen zwischen der dokumentierten und der produktiven Konfiguration entstehen. [38, pp. 178-179]

Eine der wesentlichen Voraussetzungen für Continuous Delivery ist eine funktionierende Deployment Pipeline. Unter Deployment Pipeline versteht man einen Prozess, der die Schritte Build, Test und Deployment des Softwareauslieferungsprozesses über mehrere Stages hinweg automatisiert durchläuft. [39, pp. 106-107]

Im ersten Schritt werden beispielsweise die Source Codes kompiliert, diese Stage stellt dann die erzeugten Binary-Dateien der nächsten Stage zur Verfügung. Würden bereits hier Probleme auftreten, so würde dies sofort an die Entwicklerin bzw. den Entwickler berichtet. Spätere Stages können auch manuelle Aktionen erfordern, wenn sie nicht in automatisierten Tests abgebildet werden können. [44]
Die Übergabe in die Produktion stellt die letzte Stage der Pipeline dar. Sehr allgemein gesprochen ist die Aufgabe der Deployment Pipeline somit, Fehler, die in der Produktion möglicherweise auftreten können, im Vorfeld zu finden. Dies inkludiert auch Fehler in den Bereichen Security, Performance und Usability. [44]
Die Deployment Pipeline soll auch allen in den Softwareauslieferungsprozess beteiligten Gruppen die Möglichkeit der Kollaboration bieten, damit alle eine gemeinsame Sicht auf den aktuellen Status und die Änderungen haben. [44]

5.3.3 Continuous Deployment

Die letzte Methode in Abbildung 12 ist Continuous Deployment. Dies stellt die höchste „Ausbaustufe“ der Deployment Pipeline und des Automatisierungsgrades dar. Hierbei werden Code Changes automatisch in die Produktion deployt [45].

5.4 Automation und Infrastructure as Code

Unter „Infrastructure as Code“ versteht man im Wesentlichen, dass sämtliche Konfigurationsskripte, Installationsskripte, etc. versioniert in einem Software Repository abgelegt werden [38, pp. 11-12]. Dadurch wird der gesamte Aufbau einer Infrastrukturumgebung als Code gespeichert und kann automatisiert aufgebaut werden [38, p. 47]. Dies ist für den schnellen Aufbau von Testumgebungen ein großer Vorteil und kann darüber hinaus auch in den Entwicklungsprozess integriert werden [38, p. 47]. Darüber hinaus wird auch dadurch der Gedanke der agilen Methoden in die Welt der Systemadministratoren getragen. So kann dieser Denkansatz für das Managen der Infrastruktur weitergedacht werden [39, p. 279].

6 IT Service Management und DevOps

In diesem Kapitel geht es um die zentrale Frage, ob sich DevOps mit klassischem IT Service Management mit ITIL kombinieren lässt.

Zunächst muss man sich Folgendes ins Gedächtnis rufen: Grundsätzlich ist ITIL Best Practice. Daher schreibt ITIL nichts vor. ITIL ist auch nicht per se gegen agile Ansätze oder DevOps. Allerdings muss klargestellt werden, dass ITIL keine Best-Practices-Sammlung für das Entwickeln von Systemen ist. [17, p. 2]

Development	Operations
Proaktiv Entwicklerinnen und Entwickler erstellen Changes, auf die der Operationsbereich reagieren muss.	**Reaktiv** Der Betrieb reagiert auf Supportanfragen, Bugs und neue Releases. Releases sind Change Requests.
DevOps / Agil DevOps fokussiert sich stark auf das kontinuierliche und automatisierte Ändern von Systemen. Agile Methoden versuchen, eine kontinuierliche Softwareauslieferung zu ermöglichen.	**ITSM / ITIL** Der Betrieb ist stark auf Prozesse und Entscheidungsbäume fokussiert, um damit Changes zu managen und die Verfügbarkeit zu überwachen.
JIRA, Puppet[7], Chef[8] Tools für Entwicklerinnen und Entwickler sind darauf fokussiert, alle Funktionen zu bieten, die für das Ausliefern des nächsten Release notwendig sind.	**Remedy[9] / ServiceNow** Tools wie Remedy und Service Now können Releases als Change Requests abbilden, auf die der Betrieb reagieren muss.
Self-Service DevOps-Teams arbeiten sehr selbstständig und betrachten Produktions- und Testsysteme als Self-Service-Systeme, die auf internen oder externen Cloudsystemen laufen.	**Zentrales Management** Der Betrieb betrachtet Produktions- und Testsysteme als stark kontrollierte Einheiten, für die Changes verfolgt und Risiken gelenkt werden.

Tabelle 2: Gegenüberstellung Development und Operations [46, p. 4]

[7] https://puppet.com/de
[8] https://www.chef.io/chef/
[9] http://www.bmcsoftware.de/it-solutions/remedy-itsm.html

Tabelle 2 beschreibt ein paar zentrale Charakteristiken der beiden Bereiche Development und Operations. Operation Teams sind eher reaktiv und vermehrt auf Service Management und Incident Response fokussiert. Für diese Teams eignet sich ITIL sehr gut. [46, p. 3]
Development Teams arbeiten eher proaktiv. Diese Gruppen haben völlig andere Anforderungen und Werkzeuge im Einsatz. [46, p. 3]

DevOps und die ITIL-Prozesse sind, dennoch kein Widerspruch. Um die kurzen Deploymentzyklen, die mit DevOps einhergehen, zu ermöglichen, werden manuelle Arbeitsschritte automatisiert, was gleichzeitig dazu führt, dass die Configuration Management Databases und Software Librarys auf einem aktuellen Stand gehalten werden. [38, p. 13]
Aus Sicht der bestehenden IT-Service-Management-Systeme sind bei der Umstellung auf DevOps einige Aspekte bei der Prozessgestaltung zu beachten. Die Organisation der Mitarbeiterinnen und Mitarbeiter findet eher in Produktteams[10] als in Abteilungen statt. So können auch Übergabeprozesse und deren umfangreiche Beschreibung reduziert werden. Generell gilt, Prozesse weitestgehend zu automatisieren, dies ist insbesondere für schwerfällige Abläufe stark zu empfehlen. [36, p. 198]
Es muss auf jeden Fall vermieden werden, den sehr agilen DevOps-Prozess in einen chaotischen „Jeder-darf-alles-Zustand“ zu überführen. Es muss daher sichergestellt werden, dass es eine klar definierte Strategie gibt, die auf gut dokumentierten Prozessen basiert und regelmäßig überprüft und verbessert wird. [47]

DevOps verändert, wie Services designt, in den Betrieb überführt und schlussendlich betrieben werden, nichtsdestotrotz können ITIL-Inhalte angewandt werden [17, p. 3].

6.1 DevOps und ITIL Service Strategy

Im Rahmen der Strategieüberlegungen können hier auch Regelungen mitüberlegt werden, wie Scrums generell verwaltet werden und sich die einzelnen Sprints zusammensetzen [48].

6.2 DevOps und ITIL Service Design

Hier können die gut beschriebenen Inhalte von ITIL auch für den DevOps-Ansatz verwendet werden, wie z.B. das Kapazitätsmanagement, Security Management, etc. [17, p. 3].

6.3 DevOps und ITIL Service Transition

Hier sind auf jeden Fall die Ansätze Continuous Integration und Continuous Delivery zu berücksichtigen. Dabei bietet die Deployment Pipeline wieder eine gute Transparenz über den Status des jeweiligen Codes. Generell sollte darauf geachtet werden, kleinere, dafür häufigere Änderungen umzusetzen. Die Folgen davon sind in

[10] Vgl. Kapitel 5.1.

Abbildung 13 dargestellt. So werden mit klassischen Softwareentwicklungsansätzen, bei denen große Zeitabstände zwischen den einzelnen Releases entstehen, die Geschäftsanforderungen nur sehr langsam erfüllt. Mit agilen Methoden und in weiterer Folge mit DevOps und sehr vielen kleineren Releases in kürzerer Zeit, kann auch sehr schnell auf neue Geschäftsanforderungen reagiert werden. [17, pp. 3-4]

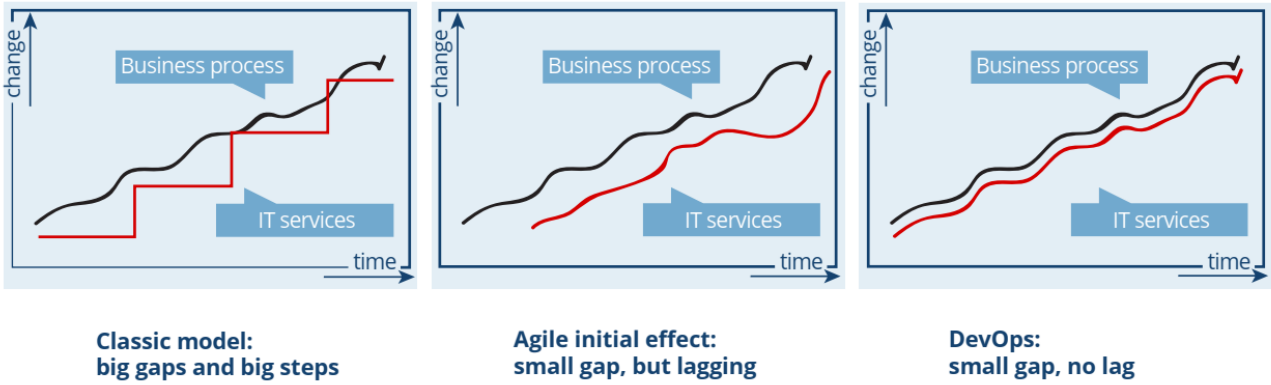

Abbildung 13: Vergleich kleine Releasezyklen zu großen Releasezyklen [17, p. 4]

Bei einer gut funktionierenden Deployment Pipeline kann auf Grund des geringen Risikos im Deploymentvorgang eine Vielzahl von Changes als Standard eingestuft werden und vom CAB vorautorisiert werden. Um die Einstufung als Standard Change vom CAB zu bekommen, kann als Referenz über das geringe Risiko eine signifikante Anzahl an Changes der letzten Monate und die dadurch verursachten Incidents als Vergleich herangezogen werden. [38, p. 438], [49]

In jedem Fall müssen aber auch für Standard Changes Einträge bzw. Tickets im Configuration Management System (CMS) existieren. Im Idealfall werden diese Einträge automatisiert erstellt und mit entsprechenden Referenzen für die weiteren Planungstools (z.B. Jira, etc.) versehen. [38, p. 438]

- DevOps und ITIL Service Asset and Configuration Management (SACM):

  Bei DevOps werden sehr viele Änderungen automatisiert in den verschiedensten Umgebungen durchgeführt. Der Vorteil davon ist, dass diese Konfigurationen versioniert in einem Respository gespeichert sind. Dies führt auch dazu, dass die Änderungen und die Historie der Konfigurationsstände nachvollziehbar gespeichert sind. Somit wird der ITIL-Prozess SACM durchaus einfacher. Dies wird durch die neuen Tools und die Deployment Pipeline ermöglicht. [17, p. 4]

- DevOps und ITIL Service Validation and Testing process:

  Auch das Testen hat bei DevOps einen automatisierteren Ansatz. Durch Continuous Integration werden alle Codeänderungen, die in das Repository eingecheckt werden, automatisiert getestet. So erhält die

Entwicklerin bzw. der Entwickler sofort Feedback über seinen Code. Durch diese neuen Methoden und Technologien kann auch dieser ITIL-Prozess profitieren. [17, p. 5]

6.4 DevOps und ITIL Service Operation

Durch die bereichsübergreifende Teamzusammenstellung aus Development und Operations, hat die Betriebsgruppe eine Ende-zu-Ende-Verantwortung für die Services, die sie betreuen. Somit kümmern sie sich um Change Requests, Incidents und Problems basierend auf einer Priorisierung, die vom Product Owner festgelegt wurde. Go-live-Aktivitäten stellen somit einen weiteren Service Request dar und sind technisch meist nur ein Klick auf einen Button. Darüber hinaus bringen die ITIL-Prozesse durchaus Vorteile. Beispielsweise kann durch sie die Koordination mehrerer DevOps-Teams erfolgen. Denn eine Änderung des ersten DevOps-Teams soll nicht die Änderung eines anderen Teams stören. Hier ist das Change Management durchaus hilfreich. [17, p. 5]

6.5 DevOps und ITIL Continual Service Improvement (CSI)

ITIL verfolgt hierbei das Konzept von Plan-Do-Check-Act für die kontinuierliche Verbesserung. Dies deckt sich sehr gut mit den Feedback-Zyklen, die bei DevOps eingesetzt werden. Bei DevOps werden darüber hinaus noch die Prinzipien der Lean IT für die kontinuierliche Verbesserung angewandt. [17, p. 5]

6.6 Zusammenfassung

ITIL ist im Vergleich zu DevOps eher sequentiell, Plan-zentriert und hat einen starken Fokus auf die Prozesse, Dokumentation und das Vorgehen allgemein. Denn ITIL definiert zuerst eine Service Strategy. Anschließend werden detaillierte Service Design Packages erstellt und entsprechende Entwicklungsprojekte inkl. der notwendigen Ressourcen geplant. Dann werden die Services designt, entwickelt, dokumentiert und nach einer genauen Verfahrensvorgabe der Produktion übergeben. Anschließend beginnt der kontinuierliche Verbesserungsprozess. Somit ist der Ansatz des Experimentierens, wie er beispielsweise bei DevOps in Kapitel 5.1 beschrieben ist, in ITIL nicht stark ausgeprägt. [50]

7 Use Case

In diesem Kapitel wird der Use Case „Entwickeln und Betreiben einer Java Web Application nach dem DevOps-Ansatz“ eines fiktiven, nach ITIL organisierten Serviceproviders beschrieben. Abbildung 14 skizziert die Infrastrukturarchitektur des Use Case. Als Beispiel wird eine in Java entwickelte Web Application, welche inhaltlich nicht näher beschrieben wird, verwendet. Diese Web Application wird in einem Docker Container verpackt und in einer virtuellen Maschine ausgeführt, diese virtuelle Maschine wiederum läuft auf einer OpenStack-Infrastruktur.

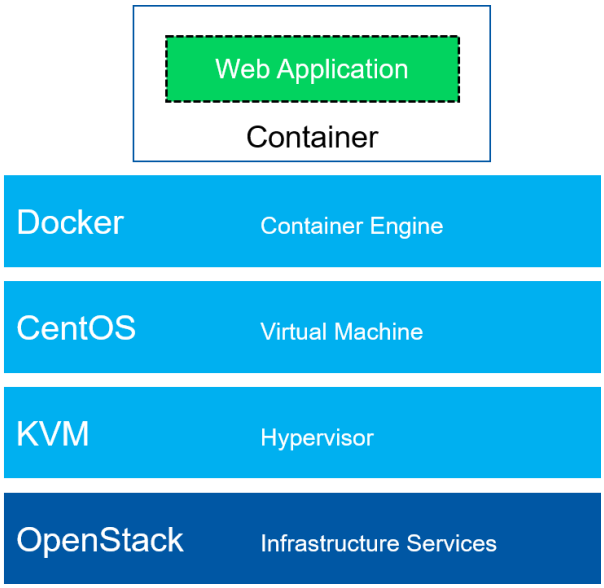

Abbildung 14: Use-Case-Übersicht[11]

Abbildung 15 zeigt einen Ausschnitt an Tools für eine mögliche Toolchain. Da die Auswahl an Tools für eine konkrete Toolchain sehr umfangreich ist [51], [52] und die Auswahlkriterien für die Tools nicht Hauptgegenstand der vorliegenden Arbeit sind, erfolgte die Auswahl für den vorliegenden Use Case anhand verschiedenster Artikel. [53], [54], [51], [55]

[11] Vgl. „Docker“ in Kapitel 4.2.
CentOS: https://www.centos.org/
KVM: https://www.linux-kvm.org/page/Main_Page
Vgl. „OpenStack“ in Kapitel 4.3.

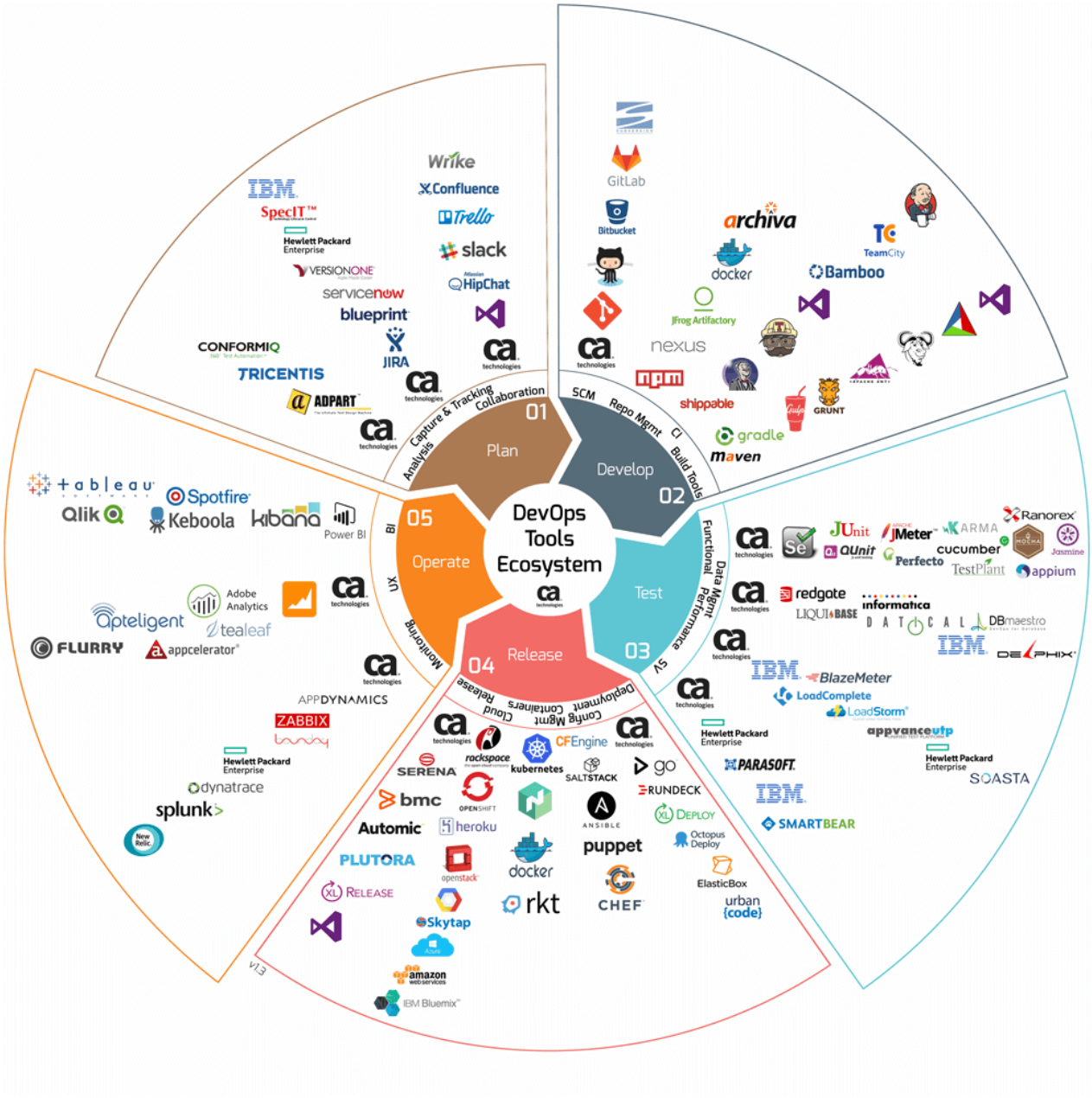

Abbildung 15: DevOps Tools Ecosystem [51]

In den nachfolgenden Kapiteln, werden für die in Abbildung 15 dargestellten DevOps-Phasen die Tools vorgestellt, mithilfe derer der Use Case umgesetzt werden kann.

Als Prozessframework für die Entwicklerinnen und Entwickler kommt Scrum zum Einsatz. Scrum ist in Kapitel 3 beschrieben.

Abbildung 16 gibt einen schematischen Überblick, wie ITIL, DevOps und Scrum zusammenhängen. DevOps verwendet für die Vorgehensweise im Entwicklungsprozess Scrum und ergänzt es um Continuous Delivery, Continuous Integration oder Continuous Deployment, je nach gewünschtem Automatisierungsgrad.

Die ITIL-Prozesse bilden rund herum ein umfangreiches Prozessnetzwerk, welches DevOps ergänzt. Konkrete Auswirkungen hat DevOps auf die ITIL-Prozesse Service Asset and Configuration Management, Change Management, Release and Deployment Management und Service Validation and Testing. Diese Prozesse müssen zum Großteil automatisiert werden, um die kurzen Time-to-Market-Anforderungen von DevOps erfüllen zu können.

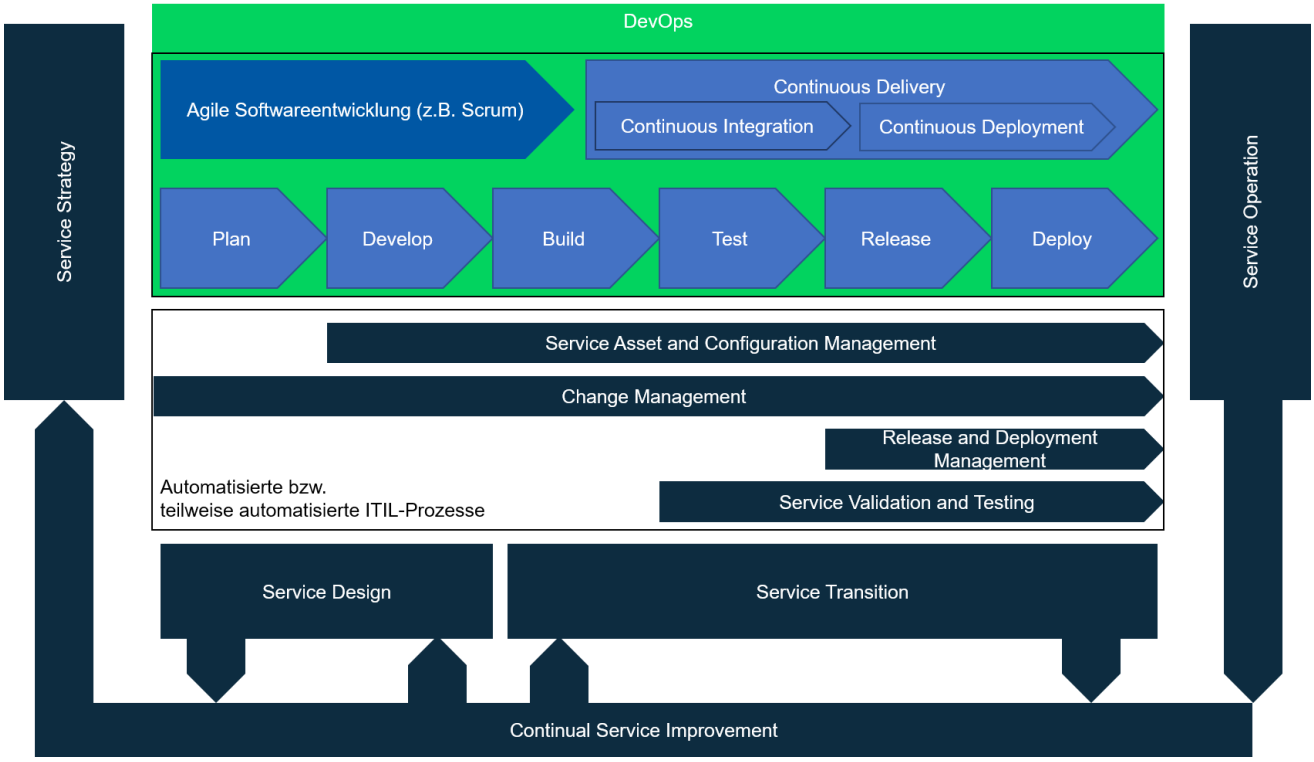

Abbildung 16: ITIL, DevOps und Scrum [56]

7.1 DevOps-Phase Plan

Die DevOps-Phase Plan, hat mit den in Tabelle 3 dargestellten ITIL-Phasen entsprechende Berührungspunkte.

	DevOps Plan	ITIL Service Strategy	ITIL Service Design	Continual Service Improvement
Fokus	Auf das Produkt bzw. auf die Produktentwicklung bezogen.	Auf die Prozesse bzw. auf das Service Management des Serviceproviders bezogen.	Auf die Prozesse bzw. auf das Service Management des Serviceproviders bezogen.	Auf die Prozesse bzw. auf das Service Management des Serviceproviders bezogen.
Referenz	Kapitel 5.2	Kapitel 2.1 und 6.1	Kapitel 2.2 und 6.2	Kapitel 2.5 und 6.5
Relevanz für Use Case	Die Sichtweise der DevOps-Phase Plan konzentriert sich viel konkreter auf die zu entwickelnde Java Web Application und weniger auf die Prozesse. Es geht hierbei um die kontinuierliche Weiterentwicklung der Java Web Application aufgrund von u.a. Kundenfeedback. Inhaltlich finden hier die planerischen Agenden von Scrum, wie z.B. die Sprint-Planung statt.	Das Strategy Management for IT Services sorgt dafür, dass die Geschäftsstrategie durch die Servicestrategie unterstützt wird. Der Bedarf für die fiktive Beispielanwendung Java Web Application wird im Service Portfolio Management aufgezeigt. Das Financial Management for IT Services ist für die Leistungsverrechnung zuständig.	Die ITIL-Service-Design-Phase hat auf Grund ihrer Prozesse sehr starke Überschneidungen mit der DevOps-Phase Plan. Hier muss man sich nun entscheiden, welche Inhalte aus welchen Prozessframeworks zur Anwendung kommen sollen. Am sinnvollsten erscheint es hier, sich für den jeweiligen Fall die besten Optionen aus allen Frameworks zu suchen.	Verbesserungs-vorschläge können für die Java Web Application über diesen Prozess angeliefert werden. Dies muss im Einklang mit Scrum stehen, da auch hier sehr schnelle Feedbackzyklen angedacht sind.

Tabelle 3: Vergleich DevOps Plan mit ITIL

(Tabelle 3: Fortsetzung auf der nächsten Seite)

	DevOps Plan	ITIL Service Strategy	ITIL Service Design	Continual Service Improvement
		Im Demand Management wird eine Kapazitätsplanung erstellt, um für den Betrieb der Java Web Application ausreichend Ressourcen im Sinne von Kapazität und Performance zur Verfügung zu haben. Das Business Relationship Management klärt die Businessanforderungen mit der Kundin bzw. dem Kunden. Diese Anforderungen müssen dann in den DevOps-Entwicklungsprozess einfließen.	Als agiles Vorgehensmodell für den DevOps-Ansatz kommt hier Scrum zur Anwendung. Das muss mit dem ITIL-Design-Coordination-Prozess im Einklang stehen. Dieser kümmert sich um die zentrale Steuerung und Koordination des Service-Design-Prozesses. Inhalte aus den folgenden ITIL-Prozessen können hier wieder zur Anwendung kommen: Service Catalogue Management, Service Level Management, Availability Management, Capacity Management, IT Service Continuity Management, Information Security Management und Supplier Management.	

Tabelle 3: Vergleich DevOps Plan mit ITIL (Fortsetzung)

Für die Projektplanung der Web Application nach dem DevOps-Ansatz können Jira und Confluence eingesetzt werden [51].

Jira ist ein Tracking-Tool zum Planen, Verfolgen (Tracken) und Veröffentlichen von Softwareprojekten für agil arbeitende Teams. Es bietet eigenen Dashboards für Kanban und Scrum an. Somit hat jedes Teammitglied eine Übersicht über den aktuellen Status der einzelnen Arbeitspakete. Dadurch können auch Engpässe identifiziert werden. Außerdem ist Jira hilfreich für das Bug-Management. [57]

Für Jira gibt es, wenn auch sehr eingeschränkt, Erweiterungen für die ITIL-Prozesse Incident Management, Problem Management, Change Management und Release Management [58].

Confluence ist ein Kollaborationstool für Teams. Es kann mit Jira kombiniert werden und bietet Funktionen wie Dokumentenmanagement, Versionierung etc. [59]

7.2 DevOps-Phase Develop

Die DevOps-Phase Develop, hat mit den in Tabelle 4 dargestellten ITIL-Phasen entsprechende Berührungspunkte.

	DevOps Develop, Test und Release	ITIL Service Design	ITIL Service Transition
Fokus	Auf das Produkt bzw. auf die Produktentwicklung bezogen.	Auf die Prozesse bzw. auf das Service Management des Serviceproviders bezogen.	Auf die Prozesse bzw. auf das Service Management des Serviceproviders bezogen.
Referenz	Kapitel 5.2	Kapitel 2.2 und 6.2	Kapitel 2.3, 6.3 und [48]
Relevanz für Use Case	Bei DevOps Develop steht der kollaborierende Entwicklungsprozess im Vordergrund. Dabei sollten Personen aus den Geschäftsfeldern, Software-Architektinnen und Software-Architekten, Entwicklerinnen und Entwickler, der Fachbereich Qualitätssicherung, Spezialistinnen und Spezialisten aus dem Betrieb, der Fachbereich Security, Lieferanten sowie Partnerinnen und Partner des Projekts involviert sein. Aus Scrum-Sicht wird hier der Sprint durchlaufen.	Die Entwicklung der Web Application fällt aus Prozesssicht in die ITIL Service Design Phase. Vergleiche dazu Tabelle 3.	Der Build, Test und Release der Anwendung fällt in die ITIL-Service-Transition-Phase. Die Deployment Pipeline automatisiert zum Teil die Prozesse Release and Deployment Management, Change Management und Service Validation and Testing. Außerdem hilft die Deployment Pipeline dem Service-Asset-and-Configuration-Management-Prozess, das CMS durch automatisierte Pflege aktuell zu halten.

Tabelle 4: Vergleich DevOps Develop, Test und Release mit ITIL

Für Continuous Delivery, Continuous Deployment und Continuous Integration kann das Open Source Tool Jenkins zum Einsatz kommen [55]. Jenkins ist ein Automatisierungsserver für die genannten Zwecke. Jenkins bietet dabei eine Vielzahl von Plugins, wodurch weitere Tools automatisiert aufgerufen werden können [60]. Die in Abbildung 17 dargestellte Deployment Pipeline basiert auf dem Jenkins-Pipeline-Konzept [61].

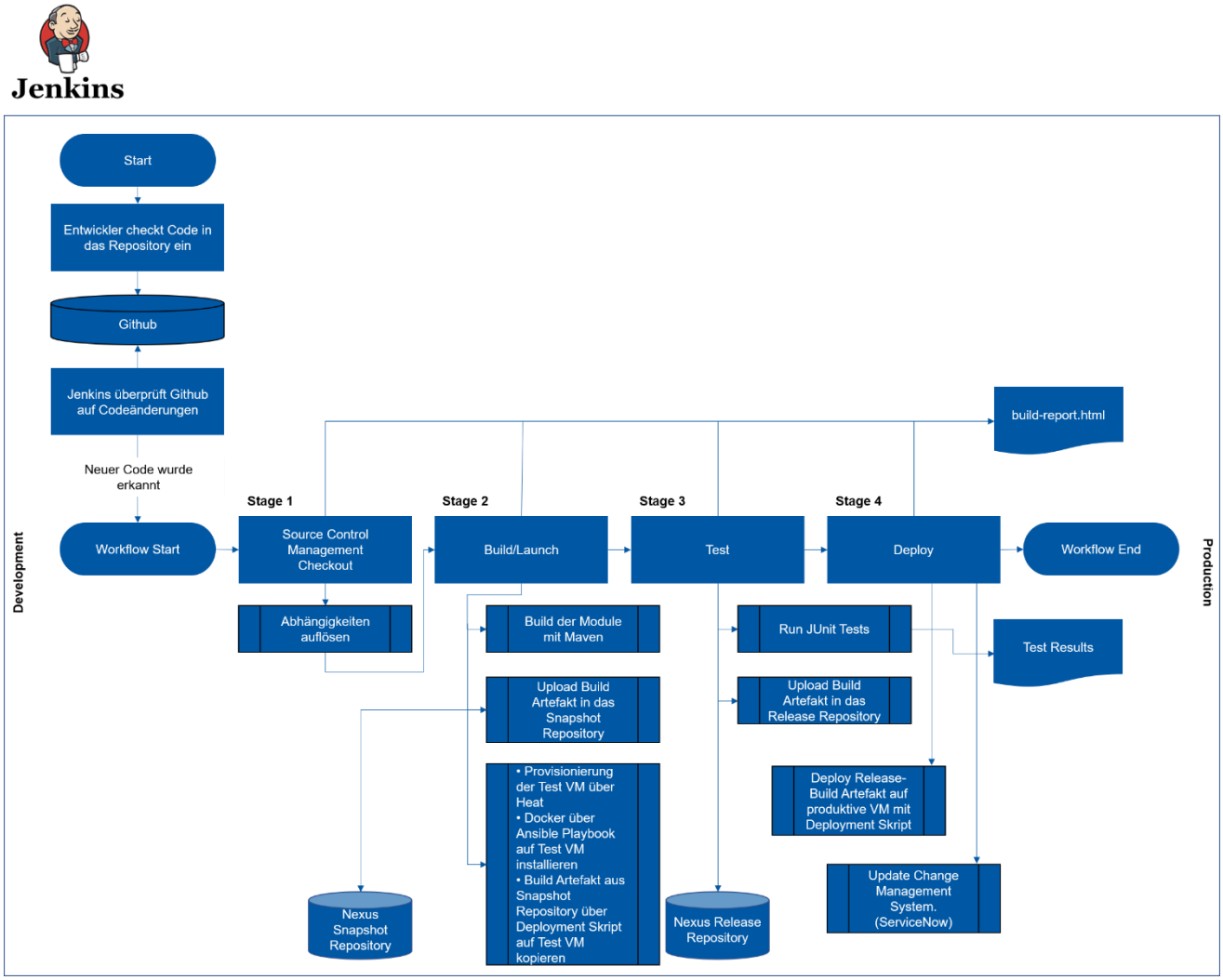

Abbildung 17: Jenkins Deployment Pipeline für Continuous Delivery

Der Workflow in Abbildung 17 startet mit dem Commit von neuem Code in das Code Repository Github durch die Entwicklerin bzw. den Entwickler. Durch das Jenkins Github Plugin [62] ist Jenkins in der Lage, Github regelmäßig auf Änderungen im verwalteten Sourcecode hin zu überprüfen.

In Stage 1 aus Abbildung 17 startet nun die Abarbeitung des Pipeline Workflow. Zunächst wird der Code inklusive aller Abhängigkeiten aus dem Code Repository ausgecheckt. In Stage 2 beginnt der Build-Prozess und der Package-Prozess mit Maven [63]. Maven ist das in diesem Beispiel verwendete Build Tool.
Das in der Stage 2 entstehende JAR File wird nun in das Artefakt Repository als Snapshot übertragen. Dafür steht das Nexus Snapshot Repository zur Verfügung [64].

Exkurs zu Repositorys: Ein Repository ist grundsätzlich eine Sammlung von Files und Metadaten, die in einer definierten Verzeichnisstruktur abgelegt werden. In diesem Use Case stellt das JAR File der Web Application

das Artefakt dar, welches im Repository gespeichert wird. Darüber hinaus wird hier zwischen Release und Snapshot Repository unterschieden. Release Repositorys sind für stabile und statische Release-Artefakte. Snapshot Repositorys speichern Artefakte, die gerade in Entwicklung stehen und ständig aktualisiert werden. [65]

7.3 DevOps-Phase Test

In Stage 3 aus Abbildung 17 wird nun die Testinstanz einer virtuellen Maschine neu provisioniert. Dies geschieht mit dem OpenStack Heat Plugin [66] von Jenkins. Das hier verwendete Heat Template enthält die notwendigen Informationen über die Cloud-Ressourcen, die zu provisionieren sind, und enthält weiters das entsprechende Ansible Playbook, welches Docker auf der neu entstandenen virtuellen Maschine installiert. Ansible [67] wird hier als Configuration Management Tool eingesetzt und liefert die in den Playbooks beschriebenen Installationsbefehle über SSH an die entsprechenden Server aus. Auf der nun laufenden und mit Docker versorgten virtuellen Test-Maschine wird nun ein Deployment-Skript von Jenkins ausgeführt, welches das zuvor erstellte JAR File der Web Application aus dem Snapshot Repository auf die virtuelle Test-Maschine kopiert.
Nun stößt Jenkins die Ausführung der Testfälle an. In diesem Beispiel werden nur Unit Tests mit JUnit durchgeführt. Hierfür bietet Jenkins ebenfalls ein entsprechendes Plugin [68].
Laufen alle Tests erfolgreich durch, so wird ein entsprechender Release Build erstellt und in das Nexus Release Repository übertragen.

7.4 DevOps-Phase Release

In Stage 4 aus Abbildung 17 wird nun die getestete Web Application auf die laufende, produktive virtuelle Maschine übertragen und ausgeführt. Über den gesamten Pipeline-Prozess entsteht auch ein entsprechender Report für die Entwicklerinnen und Entwickler. Sollte bei einem Schritt ein Fehler auftreten, so stoppt die Pipeline und ein Fehlerbericht wird erstellt.
Der letzte Schritt innerhalb der Stage 4 ist, dem ITSM Tool ServiceNow [69] zu melden, dass der Change abgeschlossen ist. Dafür gibt es ein Jenkins-Plugin [70]. Somit wird auch aus technischer Sicht der Bogen zwischen automatisierter Softwareauslieferung und IT-Servicemanagement geschlossen.

7.5 DevOps-Phase Operate

Die DevOps-Phase Operate, hat mit den in Tabelle 5 dargestellten ITIL-Phasen entsprechende Berührungspunkte.

	DevOps Operate	ITIL Service Operation
Fokus	Auf das Produkt bzw. auf die Produktentwicklung bezogen.	Auf die Prozesse bzw. auf das Service Management des Serviceproviders bezogen.
Referenz	Kapitel 5.2	Kapitel 2.4 und 6.4
Relevanz für Use Case	Hier steht die Überwachung der produktiven Web Application im Fokus. Verbesserungen werden entweder aufgrund von Kundenfeedback in die nächste Entwicklungsphase gebracht oder durch die erhobenen Überwachungsdaten bzw. Messdaten.	Bei DevOps hat das DevOps-Team eine Ende-zu-Ende-Verantwortung, daher kümmern sie sich auch um Themen, die in die ITIL-Service-Operation-Phase fallen. Das sind u.a. Change Requests, Incidents, Problems etc.

Tabelle 5: Vergleich DevOps Operate mit ITIL

Die Web Application ist nun produktiv im Betrieb und muss entsprechend überwacht werden. Darüber hinaus muss auch die Infrastruktur geeignet überwacht werden. Empfehlenswert ist es auch, die verschiedensten Überwachungssysteme in einem aggregierenden System zusammenzuführen.
Application Monitoring Tools überwachen die Anwendungen von innen, wie z.B. die Garbage Collection Performance, die Stack Traces und die Zugriffszeiten von Datenbankabfragen. NewRelic bietet mehrere Plugins für verschiedene Sprachen und Plattformen an, die überwacht werden sollen. Es muss lediglich ein NewRelic Agent in die Anwendung, die entwickelt wird, eingebaut werden. [71], [72]
Für die Überwachung der Infrastruktur können Tools wie Nagios verwendet werden [73].
Für die Zusammenführung der verschiedensten Meldungen der Monitoring- und Loggingtools eignet sich der ELK Stack [71]. Der ELK Stack besteht aus drei Open-Source-Projekten Elasticsearch, Logstash und Kibana. Elasticsearch ist eine Such- und Analyseengine. Logstash verarbeitet die Meldungen aus den verschiedenen Systemen und gibt sie in einem einheitlichen Format an Elasticsearch weiter. Kibana ist für die Visualisierung der aggregierten Daten in Elasticsearch zuständig. [74]

8 Conclusio

Diese Arbeit ist zum Ergebnis gekommen, dass ITIL und DevOps nicht im Widerspruch zueinander stehen. ITIL als umfangreiche Best-Practices-Sammlung für das IT Service Management liefert einen enormen Erfahrungsschatz, der IT-Serviceprovider dabei unterstützt, ihre Services über den gesamten Lifecycle hinweg zu managen.

DevOps als agile Technik, die aus verschiedensten Bewegungen bzw. Techniken wie Infrastructure as Code, agile Methoden, Lean Thinking, Continuous Deplyoment etc. entstand, hat als Zielgruppe eher die Entwicklerinnen und Entwickler, betrifft aber alle beteiligten Stellen im Softwareentwicklungsprozess, wie z.B. die Spezialistinnen und Spezialisten aus den Geschäftsfeldern, die Software-Architektinnen und Software-Architekten, die Qualitätssicherungsspezialistinnen und Qualitätssicherungsspezialisten etc. Da DevOps eine Ende-zu-Ende-Verantwortung dieser beteiligten Stellen fordert, entsteht überhaupt erst die Notwendigkeit, sich darüber Gedanken zu machen, wie DevOps und ITIL kombiniert werden können.

Der große Vorteil der Kombination von ITIL und DevOps liegt darin, dass sie eine höhere Automatisierung und somit eine Beschleunigung der ITIL-Prozesse ermöglicht. Dies betrifft insbesondere die Prozesse Change Management, Release and Deployment Management, Service Asset and Configuration Management sowie Service Validation and Testing. Darüber hinaus empfiehlt ITIL selbst, wie in Kapitel 2.3 beschrieben, den Einsatz von Automatisierung, um Fehler durch manuelle Eingriffe zu reduzieren.
Technisch wird die Automatisierung dieser Prozesse durch die Deployment Pipeline bewerkstelligt. Dabei müssen alle beteiligten Tools über geeignete Schnittstellen miteinander kommunizieren können.

Die Best-Practices-Ansätze aus ITIL sowie die methodischen und technischen Ansätze aus DevOps müssen aber im Einzelfall konkret an die Anforderungen eines Unternehmens angepasst werden.

OpenStack als Cloud-Plattform kann als technische Basis für den DevOps-Ansatz dienen und bietet eine gute Möglichkeit, um Infrastruktur-Ressourcen – wie beispielsweise virtuelle Maschinen – über Schnittstellen zu provisionieren. Dies ist für Konzepte wie Infrastructure as Code und generell für die schnelle und vor allem automatisierte Bereitstellung von standardisierten Infrastruktur-Ressourcen wichtig.

Abbildungsverzeichnis

Abbildung 1: ITIL-Service-Lebenszyklus [1, p. 2] 8
Abbildung 2: Phasen des Release and Deployment Management [1, p. 540] 16
Abbildung 3: Scrum-Prozess [4, p. 15] 21
Abbildung 4: Beispiel eines Kanban-Boards [4, p. 23] 24
Abbildung 5: Microservices-Architektur vs. traditionelle Architektur [17, p. 3], [15] 32
Abbildung 6: Docker-Architektur [22, p. 10] 33
Abbildung 7: OpenStack Releases [23] 35
Abbildung 8: OpenStack Map [26] 36
Abbildung 9: DevOps-Teamorganisation [36, p. 196] 42
Abbildung 10: Entwicklung von traditioneller Aktivitäten- zur Produktorientierung [36, p. 196] 42
Abbildung 11: DevOps Life Cycle [9, p. 10] 44
Abbildung 12: Vergleich Continuous Integration, Continuous Delivery, Continuous Deployment [42] 46
Abbildung 13: Vergleich kleine Releasezyklen zu großen Releasezyklen [17, p. 4] 54
Abbildung 14: Use-Case-Übersicht 56
Abbildung 15: DevOps Tools Ecosystem [51] 57
Abbildung 16: ITIL, DevOps und Scrum [56] 58
Abbildung 17: Jenkins Deployment Pipeline für Continuous Delivery 62

Tabellenverzeichnis

Tabelle 1: Unterschied zwischen Scrum und Kanban [4, p. 29] 25
Tabelle 2: Gegenüberstellung Development und Operations [46, p. 4] 52
Tabelle 3: Vergleich DevOps Plan mit ITIL 59
Tabelle 4: Vergleich DevOps Develop, Test und Release mit ITIL 61
Tabelle 5: Vergleich DevOps Operate mit ITIL 64

Literaturverzeichnis

[1] N. Ebel, Basiswissen ITIL® 2011 Edition, 1. Hrsg., Heidelberg: dpunkt.verlag GmbH, 2015.

[2] B. Aiello und L. Sachs, „Implement ITIL with DevOps," IBM, [Online]. Available: https://www.ibm.com/developerworks/library/d-implement-itil-devops/index.html. [Zugriff am 06 10 2018].

[3] S. Hofert, Agiler führen, Wiesbaden: Springer Gabler, 2018.

[4] I. Hanschke, Agile in der Unternehmenspraxis, Wiesbaden: Springer Vieweg, 2017.

[5] „Manifesto for Agile Software Development," [Online]. Available: http://www.agilemanifesto.org. [Zugriff am 25 07 2018].

[6] J. Sutherland und K. Schwaber, „Der Scrum Guide," [Online]. Available: http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-German.pdf. [Zugriff am 29 07 2018].

[7] „What's the Difference? Agile vs Scrum vs Waterfall vs Kanban," [Online]. Available: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban. [Zugriff am 04 08 2018].

[8] A. Kohne, H. Elschner, K.-U. Winter, K. Ludger, P. Kleinmanns, S. Dellbrügge und U. Pöhler, Die IT-Fabrik Der Weg zum automatisierten IT-Betrieb, Dortmund: Springer Vieweg, 2016.

[9] S. Sharma, DevOps For Dummies®, IBM Limited Edition, USA: John Wiley & Sons, Inc., 2014.

[10] G. Hughes, „6 Steps to Bridge DevOps with Release Management in the Enterprise," [Online]. Available: https://www.cmcrossroads.com/article/6-steps-bridge-devops-release-management-enterprise. [Zugriff am 13 10 2018].

[11] D. Radez, OpenStack Essentials, Birmingham: Packt Publishing Ltd., 2015.

[12] J. Repschläger, S. Wind und R. Zarnekow, „Klassifikationsrahmen für die Anbieterauswahl in der Cloud," *Jahrestagung der Gesellschaft für Informatik/Workshop: Neue Aspekte der zwischenbetrieblichen Integration durch Enterprise.,* Nr. 2, 2011.

[13] P. Mell und T. Grance, „The NIST Definition of Cloud Computing," 2011. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. [Zugriff am 19 08 2018].

[14] C. Ward, *THE DZONE GUIDE TO MICROSERVICES VOLUME 1,* p. 2, 2017.

[15] C. de la Torre, K. Deep Singh und V. Turecek, „Microsoft Azure - Azure Service Fabric and the Microservices Architecture," [Online]. Available: https://msdn.microsoft.com/en-us/magazine/mt595752.aspx. [Zugriff am 24 10 2018].

[16] M. Werner, „Executive Summary," *THE DZONE GUIDE TO MICROSERVICES VOLUME 1,* p. 3, 2017.

[17] S. Gondhalekar, N. Loader und R. Israels, „The Impact of DevOps on Your ITIL Implementation," [Online]. Available: http://cdn.instantmagazine.com/upload/1839/devopswhite_paper_-_the_impact_of_devops_on_your_itil_implementa.5e75b1ad3b6b.pdf. [Zugriff am 29 09 2018].

[18] R. Büst, „IaaS vs. PaaS vs. Container: Moderne Anwendungsentwicklung in der Cloud," [Online]. Available: https://www.crisp-research.com/iaas-vs-paas-vs-container-moderne-anwendungsentwicklung-der-cloud/. [Zugriff am 24 10 2018].

[19] T. Warner, „Container as a Service: Kurzüberblick über die drei wichtigsten CaaS-Anbieter," [Online]. Available: https://www.searchdatacenter.de/tipp/Container-as-a-Service-Kurzueberblick-ueber-die-drei-wichtigsten-CaaS-Anbieter. [Zugriff am 24 10 2018].

[20] T. Jardinet, „Microservices and Team Organization," *THE DZONE GUIDE TO MICROSERVICES VOLUME 1,* pp. 22-23, 2017.

[21] „Docker overview," [Online]. Available: https://docs.docker.com/engine/docker-overview/. [Zugriff am 06 11 2018].

[22] J. Turnbull, „The Docker Book," 11 03 2017. [Online]. Available: https://dockerbook.com/.

[23] „OpenStack Releases," OpenStack Foundation, [Online]. Available: https://releases.openstack.org/. [Zugriff am 19 08 2018].

[24] O. Khedher und C. D. Chowdhury, Mastering OpenStack, Second Edition Hrsg., Birmingham - Mumbai: Packt Publishing Ltd., 2017.

[25] D. Srocke und F. Karlstetter, „Was ist eine REST API?," [Online]. Available: https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/. [Zugriff am 18 01 2019].

[26] „Openstack-Map," [Online]. Available: https://www.openstack.org/assets/software/projectmap/openstack-map.pdf. [Zugriff am 21 10 2018].

[27] „Keystone, the OpenStack Identity Service," [Online]. Available: https://docs.openstack.org/keystone/latest/. [Zugriff am 23 10 2018].

[28] „Welcome to Swift's documentation!," [Online]. Available: https://docs.openstack.org/swift/latest/. [Zugriff am 23 10 2018].

[29] „Cinder, the OpenStack Block Storage Service," [Online]. Available: https://docs.openstack.org/cinder/latest/. [Zugriff am 23 10 2018].

[30] „Welcome to Ironic's documentation!," [Online]. Available: https://docs.openstack.org/ironic/latest/. [Zugriff am 23 10 2018].

[31] „Welcome to Glance's documentation!," [Online]. Available: https://docs.openstack.org/glance/latest/. [Zugriff am 23 10 2018].

[32] „OpenStack Compute (nova)," [Online]. Available: https://docs.openstack.org/nova/latest/. [Zugriff am 23 10 2018].

[33] „Welcome to Neutron's documentation!," [Online]. Available: https://docs.openstack.org/neutron/latest/. [Zugriff am 23 10 2018].

[34] „Welcome to the Heat documentation!," [Online]. Available: https://docs.openstack.org/heat/latest/. [Zugriff am 23 10 2018].

[35] „Horizon: The OpenStack Dashboard Project," [Online]. Available: https://docs.openstack.org/horizon/latest/. [Zugriff am 23 10 2018].

[36] D. Söllner, „DevOps in der Praxis – Handlungsfelder für eine erfolgreiche Zusammenarbeit von Entwicklung und Betrieb," *HMD Praxis der Wirtschaftsinformatik,* Bd. 54, pp. 189-204, April 2017.

[37] C. Ebert, G. Gallardo, J. Hernantes und N. Serrano, „DevOps," *IEEE Software,* pp. 94-100, 2016.

[38] P. Debois, J. Willis, G. Kim und J. Humble, The DevOps Handbook, Portland: IT Revolution Press, LLC, 2016.

[39] J. Humble und D. Farley, Continuous delivery: reliable software releases through build, test, and deployment automation, Boston, MA: Addison-Wesley, 2011.

[40] D. Schäfer, Lean-Informationstechnik im Finazdienstleistungssektor, Wiesbaden: Springer Gabler, 2015.

[41] I. Hanschke, „Lean IT-Management – Voraussetzung für Business-Agilität," [Online]. Available: https://www.informatik-aktuell.de/management-und-recht/projektmanagement/lean-it-management-voraussetzung-fuer-business-agilitaet.html. [Zugriff am 05 08 2018].

[42] S. Pittet, „Continuous integration vs. continuous delivery vs. continuous deployment," Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd. [Zugriff am 16 09 2018].

[43] „What is a staging environment?," [Online]. Available: https://www.programmerinterview.com/index.php/technical-vocabulary/what-is-a-staging-environment/. [Zugriff am 05 10 2018].

[44] M. Fowler, „DeploymentPipeline," [Online]. Available: https://martinfowler.com/bliki/DeploymentPipeline.html. [Zugriff am 04 08 2018].

[45] „Unterschiede zwischen Continuous Integration, Continuous Delivery und Continuous Deployment," [Online]. Available: https://www.scrum.de/unterschiede-zwischen-continuous-integration-continuous-delivery-und-continuous-deployment/. [Zugriff am 24 10 2018].

[46] „Release Management is not a "One Size Fits All" Solution," PLUTORA INC, [Online]. Available: https://www.plutora.com/resources/whitepaper-release-management-is-not-one-size-fits-all-solution. [Zugriff am 19 08 2018].

[47] A. Roberts, „DevOps and the ITIL Change Manager," 24 04 2017. [Online]. Available: https://www.icore-ltd.com/devops-itil-change-manager/. [Zugriff am 06 05 2018].

[48] „15 Hacks Required to Align ITIL and DevOps," [Online]. Available: https://www.joetheitguy.com/2017/09/20/15-hacks-required-align-itil-devops/. [Zugriff am 04 12 2018].

[49] J. Betteley, „DevOps in an ITIL environment," [Online]. Available: https://devopsnet.com/2016/10/28/devops-in-an-itil-environment/. [Zugriff am 21 10 2018].

[50] C. Betz, „ITIL and DevOps: Differences & Frameworks Working Together," BMC, 31 01 2017. [Online]. Available: https://www.bmc.com/blogs/itil-and-devops-lets-not-paper-over-the-differences/. [Zugriff am 19 10 2018].

[51] N. Cohen, „The Ultimate DevOps Tools Ecosystem Tutorial, Part II: Planning," [Online]. Available: https://dzone.com/articles/the-ultimate-devops-tools-ecosystem-tutorial-part-1. [Zugriff am 04 08 2018].

[52] „Wo befinden sich Ihre Tools in der Continuous Delivery-Landschaft?," [Online]. Available: https://assessment-tools.ca.com/tools/continuous-delivery-tools/de/. [Zugriff am 06 11 2018].

[53] „White Paper Continuous Delivery," levi9, [Online]. Available: https://www.levi9.com/wp-content/uploads/2018/04/Whitepaper-Continuous-delivery.pdf . [Zugriff am 08 11 2018].

[54] S. Gagnon, „https://dzone.com/articles/an-example-of-a-continuous-integration-delivery-pi," [Online]. Available: https://dzone.com/articles/mining-the-ground-truth-of-enterprise-toolchains. [Zugriff am 08 11 2018].

[55] N. Cohen, „The Ultimate DevOps Tools Ecosystem Tutorial, Part III: Developing," [Online]. Available: https://dzone.com/articles/the-ultimate-devops-tools-ecosystem-tutorial-part-2. [Zugriff am 04 08 2018].

[56] B. Martin, „DevOps, Agile Development, Continuous Delivery und ITIL: Was gibt es für Zusammenhänge und wie man sie nutzen kann," [Online]. Available: https://blog.itil.org/2014/10/kategorie-liste-home/itil/devops-agile-development-continuous-delivery-und-itil-was-gibt-es-fuer-zusammenhaenge-und-wie-man-sie-nutzen-kann/. [Zugriff am 08 12 2018].

[57] „Jira Software - Features," [Online]. Available: https://www.atlassian.com/software/jira/features. [Zugriff am 05 08 2018].

[58] M. Skemme, „Using JIRA to support ITIL processes," [Online]. Available: https://martinskemme.wordpress.com/2009/02/01/using-jira-to-support-itil-processes/. [Zugriff am 08 11 2018].

[59] „Confluence Features," [Online]. Available: https://www.atlassian.com/software/confluence. [Zugriff am 05 08 2018].

[60] „Jenkins User Documentation," [Online]. Available: https://jenkins.io/doc/. [Zugriff am 06 11 2018].

[61] „Pipeline," [Online]. Available: https://jenkins.io/doc/book/pipeline/. [Zugriff am 06 11 2018].

[62] „GitHub Plugin," [Online]. Available: https://wiki.jenkins.io/display/JENKINS/GitHub+Plugin. [Zugriff am 06 11 2018].

[63] „Apache Maven Project Introduction," [Online]. Available: https://maven.apache.org/what-is-maven.html. [Zugriff am 06 11 2018].

[64] „Nexus Repository OSS," [Online]. Available: https://de.sonatype.com/nexus-repository-oss. [Zugriff am 06 11 2018].

[65] T. OBrien, „What is an Artifact Repository?," [Online]. Available: https://blog.sonatype.com/2009/04/what-is-a-repository/. [Zugriff am 08 12 2018].

[66] „Openstack Heat Plugin," [Online]. Available: https://wiki.jenkins.io/display/JENKINS/Openstack+Heat+Plugin . [Zugriff am 06 11 2018].

[67] „USE CASE CONFIGURATION MANAGEMENT," [Online]. Available: https://www.ansible.com/use-cases/configuration-management. [Zugriff am 06 11 2018].

[68] „JUnit Plugin," [Online]. Available: https://wiki.jenkins.io/display/JENKINS/JUnit+Plugin. [Zugriff am 07 11 2018].

[69] „ITSM - Enterprise IT Service Management - ServiceNow," [Online]. Available: https://www.servicenow.com/products/it-service-management.html. [Zugriff am 08 12 2018].

[70] „ServiceNow API Jenkins Plugin," [Online]. Available: https://github.com/jenkinsci/service-now-plugin/blob/master/README.md . [Zugriff am 08 12 2018].

[71] N. Cohen, „The Ultimate DevOps Tools Ecosystem Tutorial, Part VI: Operate," [Online]. Available: https://dzone.com/articles/the-ultimate-devops-tools-ecosystem-tutorial-part-5. [Zugriff am 04 08 2018].

[72] „Introduction to New Relic for Java," [Online]. Available: https://docs.newrelic.com/docs/agents/java-agent/getting-started/introduction-new-relic-java. [Zugriff am 07 11 2018].

[73] „Nagios Open Source," [Online]. Available: https://www.nagios.org/. [Zugriff am 07 11 2018].

[74] „What is the ELK Stack?," [Online]. Available: https://www.elastic.co/de/elk-stack. [Zugriff am 08 11 2018].

[75] „Multi-Tenancy-Architektur," [Online]. Available: https://www.itwissen.info/Multi-Tenancy-Architektur-multi-tenancy-architecture.html. [Zugriff am 22 10 2018].