Fachhochschule Kiel

Fachbereich Wirtschaft

Wirtschaftsinformatik

Wintersemester 2016/2017

Bachelorthesis zum Thema:

**Behaviour-Driven Development unter den Voraussetzungen von Continuous Testing – Erfüllt Cucumber Ansätze beider Praktiken?**

Autor/in: Mareike Muus

Matrikelnummer: 924027

Adresse: Veilchenweg 37  
23626 Ratekau

E-Mail: mareike.muus@student.fh-kiel.de

Telefon: 017664919310

Betreuender Hochschullehrer: Prof. Dr. Doris Weßels

Abgabedatum: 23.02.2017

# Inhaltsverzeichnis

[Inhaltsverzeichnis I](#_Toc473558372)

[Gliederung II](#_Toc473558373)

[Abkürzungsverzeichnis IV](#_Toc473558374)

[Anhangsverzeichnis V](#_Toc473558375)

[Tabellenverzeichnis VI](#_Toc473558376)

[Abbildungsverzeichnis VII](#_Toc473558377)

[1 Einleitung 1](#_Toc473558378)

[2 Vorstellung des Unternehmens 3](#_Toc473558379)

[3 Einführung in die theoretischen Grundlagen 6](#_Toc473558380)

[4 Praxisbeispiel Behaviour-Driven Development 29](#_Toc473558381)

[5 Experteninterviews 40](#_Toc473558382)

[6 Fazit und Ausblick 41](#_Toc473558383)

[7 Anhang 42](#_Toc473558384)

[Literaturverzeichnis 53](#_Toc473558385)

[Internetquellen 56](#_Toc473558386)

[Eidesstattliche Versicherung 60](#_Toc473558387)

# Gliederung

[Inhaltsverzeichnis I](#_Toc473558319)

[Gliederung II](#_Toc473558320)

[Abkürzungsverzeichnis IV](#_Toc473558321)

[Anhangsverzeichnis V](#_Toc473558322)

[Tabellenverzeichnis VI](#_Toc473558323)

[Abbildungsverzeichnis VII](#_Toc473558324)

[1 Einleitung 1](#_Toc473558325)

[1.1 Motivation und Problemstellung 1](#_Toc473558326)

[1.2 Ziel der Arbeit und Forschungsfragen 1](#_Toc473558327)

[1.3 Aufbau der Arbeit 2](#_Toc473558328)

[1.4 Abgrenzung des Themengebiets 2](#_Toc473558329)

[2 Vorstellung des Unternehmens 3](#_Toc473558330)

[2.1 IBM – International Business Machines 3](#_Toc473558331)

[2.2 IBM – Deutschland 3](#_Toc473558332)

[2.3 IBM GBS – Global Business Services 4](#_Toc473558333)

[2.4 IBM – Rational Center of Competence Germany 4](#_Toc473558334)

[3 Einführung in die theoretischen Grundlagen 6](#_Toc473558335)

[3.1 Die Softwareentwicklung 6](#_Toc473558336)

[3.1.1 Klassische Softwareentwicklung 6](#_Toc473558337)

[3.1.2 Agile Softwareentwicklung 8](#_Toc473558338)

[3.2 Behaviour-Driven Development 10](#_Toc473558339)

[3.2.1 Bedeutung und Herkunft 10](#_Toc473558340)

[3.2.2 Vorgehensweise bei BDD 12](#_Toc473558341)

[3.2.3 Vor- und Nachteile 14](#_Toc473558342)

[3.3 Der Konflikt innerhalb der IT-Organisation 15](#_Toc473558343)

[3.4 DevOps im Unternehmen 17](#_Toc473558344)

[3.4.1 Kernelemente von DevOps 18](#_Toc473558345)

[3.4.2 Agilität von DevOps 19](#_Toc473558346)

[3.5 Die DevOps Praktiken 20](#_Toc473558347)

[3.5.1 Continuous Integration 20](#_Toc473558348)

[3.5.2 Continuous Delivery 22](#_Toc473558349)

[3.5.3 Continuous Deployment 23](#_Toc473558350)

[3.5.4 Continuous Testing 25](#_Toc473558351)

[3.5.5 Continuous Monitoring 26](#_Toc473558352)

[3.6 Vor- und Nachteile von DevOps 27](#_Toc473558353)

[3.7 Zusammenfassung von BDD und DevOps 27](#_Toc473558354)

[4 Praxisbeispiel Behaviour-Driven Development 29](#_Toc473558355)

[4.1 Entwicklung von Testfällen mit Cucumber 29](#_Toc473558356)

[4.2 Vorstellung des Projekts 31](#_Toc473558357)

[4.3 Anforderungen an das Tool 31](#_Toc473558358)

[4.4 Definition von Features und Szenarien 33](#_Toc473558359)

[4.5 Erläuterung der Step Definition 34](#_Toc473558360)

[4.6 Automatisierungsprozesse 36](#_Toc473558361)

[4.7 Ausführung von Cucumber 37](#_Toc473558362)

[4.8 Bewertung der Durchführbarkeit 38](#_Toc473558363)

[5 Experteninterviews 40](#_Toc473558364)

[5.1 Gesprächsverlauf 40](#_Toc473558365)

[5.2 Vorherrschende Meinung 40](#_Toc473558366)

[6 Fazit und Ausblick 41](#_Toc473558367)

[7 Anhang 42](#_Toc473558368)

[Literaturverzeichnis 53](#_Toc473558369)

[Internetquellen 56](#_Toc473558370)

[Eidesstattliche Versicherung 60](#_Toc473558371)

# Abkürzungsverzeichnis

BDD Behaviour-Driven Development

CD Continuous Delivery

CI Continuous Integration

CLM Collaboration Lifecycle Management

CM Continuous Monitoring

CoC Center of Competence

CT Continuous Testing

Dev Anwendungsentwicklung (abgekürzt von development)

DevOps Development/ Operations

IBM International Business Machines

IT Informationstechnik

GBS Global Business Services

TDD Test-Driven Development

Ops IT-Betrieb

XP eXtreme Programming

# Anhangsverzeichnis

Anhang I: Prinzipien von XP 42

Anhang II: Primärpraktiken von XP 44

Anhang III: Folgepraktiken von XP 46

Anhang IV: Weitere User-Stories und Szenarien 48

# Tabellenverzeichnis

[Tabelle 1: Beispiel für eine User-Story 12](#_Toc473558392)

[Tabelle 2: Aufbau einer User-Story 13](#_Toc473558393)

[Tabelle 3: Beispiel für ein Szenario 13](#_Toc473558394)

[Tabelle 4: User-Story 2 48](#_Toc473558395)

[Tabelle 5: Szenarien für User-Story 2 48](#_Toc473558396)

# Abbildungsverzeichnis

Abbildung 1: Unternehmensstruktur der IBM **Fehler! Textmarke nicht definiert.**

Abbildung 2: Das Wasserfallmodell nach Royce 7

Abbildung 3: Die XP-Praktiken im Überblick 9

Abbildung 4: Der Red-Green-Refactor Cycle 11

Abbildung 5: Die Isolation innerhalb der IT-Organisation 16

Abbildung 6: Die Reichweiten der verschiedenen DevOps Praktiken 20

Abbildung 7: Der Continuous Integration-Prozess 21

Abbildung 8: Die CD-Pipeline 22

Abbildung 9: Die Continuous Deployment-Pipeline im Vergleich 24

Abbildung 10: Der Monitoring-Prozess 26

Abbildung 11: Die Funktionsweise von Cucumber 30

Abbildung 12: Das Tool im Überblick 32

Abbildung 13: Das erfolgreiche Testresultat 38

Abbildung 14: Die Übersicht der fehlenden Steps in Cucumber 38

Abbildung 15: Die Ansicht eines fehlenden Steps 39

# Einleitung

Dieses Kapitel befasst sich mit der Problemstellung, die im Verlauf dieser Arbeit analysiert werden soll.

## Motivation und Problemstellung

Die heutige Branche der Informationstechnik (IT) ist geprägt durch ihre Schnelllebigkeit. Kunden bestehen auf möglichste kurze Lieferzeiten der zu entwickelnden Software. Durch diese Bedingung ist der Einsatz von agilen Methoden unabdinglich. DevOps bietet zur agilen Vorgehensweise eine optimale Ergänzung, da hier ebenfalls die Zielsetzung der schnell verfügbaren und fehlerfreien Software verfolgt wird. Der kulturelle Aspekt aber auch die Automatisierung von Prozessen wird im Bereich DevOps außerdem fokussiert. Um eine fehlerfreie Software garantieren zu können, bildet das Testen in Form von Continuous Testing eine der Kernpraktiken von DevOps.

An diesem Ansatz entsteht die Überschneidung zu der Softwareentwicklungspraktik Behaviour-Driven Development. Es werden Automatisierungsprozesse eingesetzt und die Kommunikation zwischen den einzelnen Parteien bildet den Kern des Vorgehens.

Ist also ein Zusammenspiel von dem sehr modernen DevOps-Ansatz und der etwas älteren Praktik Behaviour-Driven Development möglich?

## Ziel der Arbeit und Forschungsfragen

Da das Thema DevOps in vielen Unternehmen heutzutage die strategischen Ziele mit abbildet, liegt der Fokus darauf, wie Behaviour-Driven Development im Bereich DevOps eingesetzt werden kann.

Die folgenden Forschungsfragen werden im Verlauf beantwortet:

1. Besteht Einklang zwischen den Behaviour-Driven Development-Ansätze und DevOps-Praktiken?
2. Ist Behaviour-Driven Development mit Cucumber im Bereich Continuous Testing möglich?
3. Wie beurteilen Experten die Kombination Behaviour-Driven Development und DevOps?

## Aufbau der Arbeit

Zur Einleitung beschreibt das erste Kapitel hauptsächlich die Problemstellung und deren Hintergründe, mit der sich diese Arbeit beschäftigt. Die Zielstellung, aus der sich die Forschungsfragen ableiten, wird definiert. Die Beantwortung dieser Fragen folgt im weiteren Verlauf.

Im zweiten Kapitel werden die theoretischen Grundlagen erläutert. Die klassische Softwareentwicklung wird im Vergleich zur agilen Softwareentwicklung in unterschiedliche Vorgehensmodelle vorgestellt. Des Weiteren wird der Bereich des Testens in der Softwareentwicklung genauer betrachtet. Das Prinzip Behaviour-Driven Development wird dem klassischen Test-Driven Development gegenübergestellt. Außerdem werden die Kernelemente sowie Praktiken von DevOps beschrieben. So wird ein Überblick zu den Themenbereichen Behaviour-Driven Development und DevOps gegeben. Im weiteren Verlauf dieses Kapitels wird der Zusammenhang zwischen DevOps und Behaviour-Driven Development erläutert.

Das dritte Kapitel wird sich auf die praktische Anwendung von Behaviour-Driven Development konzentrieren. Diese erfolgt an einem Softwareprojekt, welches zunächst erläutert wird. Außerdem findet die Vorstellung des im weiteren Verlauf verwendeten Tools Cucumber statt. Das vorgestellte Projekt wird nun mittels Cucumber analysiert und getestet.

Im vierten Kapitel werden die Forschungsfragen durch die vorherigen Erläuterungen und anhand der praktischen Anwendung fundiert beantwortet. Außerdem werden Expertenmeinungen zur Beantwortung der Forschungsfragen hinzugezogen.

Das abschließende fünfte Kapitel beschäftigt sich mit einem Fazit der Ausarbeitung, außerdem wird ein Ausblick darüber gegeben, welche Bereiche nicht bearbeitet wurden, aber noch genug Raumen geben, zur genaueren Untersuchung.

## Abgrenzung des Themengebiets

Es wird keine quantitative Untersuchung des Themengebiets, sondern lediglich die qualitative Überprüfung stattfinden. Des Weiteren wird festgehalten, dass auch keine weiteren Analysen zu zusätzlichen Funktionsweisen des Tools Cucumber erfolgen, sondern nur der Bereich des Testens konzentriert betrachtet wird.

# Vorstellung des Unternehmens

Das Unternehmen, in dem das Praktikum abgeleistet wurde, wird im Folgenden vorgestellt. Die unterschiedlichen Bereiche des Unternehmens werden erläutert.

## IBM – International Business Machines

International Business Machines (IBM) ist eines der weltweit führenden Unternehmen in der Branche der IT.

Im Jahr 1911 wurde das Unternehmen mit dem Hauptsitz in Armonk, New York, gegründet. Derzeit leitet Virginia Rometty die IBM.

Im Jahr 2015 erzielte das Unternehmen einen Jahresumsatz von 81,7 Milliarden US-Dollar und präsentiert sich mit Niederlassungen in mehr als 170 Ländern.[[1]](#footnote-1)

Als Hauptstrategie wird Innovation genannt. Durch einen hohen Einsatz in Forschung und Entwicklung konnte sich die IBM immer wieder neu definieren. So wurden 2015 mit Investitionen von mehr als sechs Milliarden US-Dollar mehr als 7.355 US-Patente erteilt. Mehr als 2.000 Patente davon beschäftigen sich mit den modernen Bereichen Cognitive und Cloud-Computing. Die IBM belegt damit zum 22. Mal in Folge den Spitzenplatz in der Rangliste des US-Patentamts.

Die Mitarbeiterinnen und Mitarbeiter der IBM arbeiten mit den Kunden in Form von Unternehmen, öffentlichen Auftraggebern und Non-Profit-Organisationen zusammen, um IT-Lösungen zu entwickeln, die die Kunden dynamischer und effizienter agieren lassen. Durch die Spezialisierung der einzelnen Mitarbeiter und der Vor-Ort-Betreuung erzielen die Kunden einen Vorteil.

## IBM – Deutschland

Die IBM Deutschland mit dem Hauptsitz in Ehningen wird seit 2011 von Martina Koederitz, einer Diplom Betriebswirtin, geleitet. Im Jahr 2013 wurde die DACH-Region (Deutschland, Österreich, Schweiz) als übergeordnete Organisation für IBM Deutschland eingeführt.

Bei der deutschen Unternehmung ist das 2015 gegründete IBM Watson Innovation Center in München zu unterstreichen, denn dies soll als weltweite Zentrale des Bereichs Watson / Internet of Things dienen.[[2]](#footnote-2)

IBM Deutschland ist in vier Kernkompetenzen gegliedert. Die Aufgabengebiete der jeweiligen Kernkompetenzen werden in Abbildung 1 dargestellt.

Abbildung : Die Unternehmensstruktur der IBM[[3]](#footnote-3)

## IBM GBS – Global Business Services

IBM Global Business Services (GBS) ist in den Kernbereich Solutions & Services einzuordnen. Dieser Bereich ist ebenfalls in 170 Ländern mit rund 125.000 Beratern vertreten. Mit einem Jahresumsatz von 17 Milliarden US-Dollar im Jahr 2015 erreicht die GBS rund ein Viertel des gesamten Jahresumsatzes der IBM. Die Leistungserbringung erstreckt sich dabei über die gesamte Wertschöpfungskette eines Unternehmens.[[4]](#footnote-4)

## IBM – Rational Center of Competence Germany

Das Rational Center of Competence (CoC) Germany wird von Dr. Frank Hollenberg geleitet und ist dem Bereich IBM GBS zugeordnet.

Im Jahr 2003 wurde die Firma „Rational“ durch die IBM aufgekauft, seither ist das Team um Dr. Frank Hollenberg auf die Beratung, Unterstützung und Bereitstellung der Rational Jazz-Produkte spezialisiert. Die Rational Jazz-Produkte bestehen aus den Tools „Rational Team Concert“, „DOORS next Generation“ und „Quality Manager“. Die Projektplanung ist durch das RTC abgedeckt. Im DNG wird das Requirements Management angeboten und das QM wird zum Anlegen des Testmanagements genutzt. Die Tools bilden so sämtliche Tätigkeiten ab und werden deshalb auch als Collaborative LifeCycle Management Tools bezeichnet.

Seit September 2016 wird an der Entwicklung eines neuen Angebots des Teams gearbeitet. Zukünftig sollen nun auch Open Source und Third Party Produkte die Produktpalette erweitern.

Das Ziel des Teams ist, die Arbeit in den Projektbüros durch den Einsatz der Jazz-Produkte und OS3P Produkte zu optimieren und zu beschleunigen. Projektbüros sind aus Sicht des Rational CoC Projekte bei Kunden der IBM, in denen IBM Mitarbeiter aus anderen Geschäftszweigen Tätigkeiten übernommen haben.

# Einführung in die theoretischen Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen erklärt. Der Einstieg in Thematik erfolgt über unterschiedliche Ansätze Softwareentwicklung. Außerdem wird die Praktik Behaviour-Driven Development vorgestellt und es findet eine Überleitung in die Thematik DevOps statt.

## Die Softwareentwicklung

Ziel der Softwareentwicklung ist es, ein Softwaresystem zu entwickeln, das den qualitativen Ansprüchen des Auftraggebers entspricht. Es werden auf Grundlage der vom Auftraggeber definierten Ziele Anforderungen entwickelt, welche dann in der Entwicklung der Software erfüllt werden müssen.[[5]](#footnote-5) Diese Tätigkeit wird als Requirements Engineering bezeichnet. Für den Prozess bis hin zur fertiggestellten Software werden verschiedene Ansätze verfolgt. Es wird jedoch darin unterschieden, in welcher Reihenfolge diese Schritte durchgeführt werden. Im Folgenden wird der klassische Softwareentwicklungsprozess anhand des Wasserfallmodells und der agile Softwareentwicklungsansatz am Beispiel von Scrum vorgestellt.

### Klassische Softwareentwicklung

Das bekannteste Modell der klassischen Softwareentwicklung ist das Wasserfallmodell. Im Jahr 1970 erläuterte Winston W. Royce dieses zum ersten Mal in seinem Artikel „Managing the Development of Large Software Systems“. Dieses Modell wurde 1981 von Berry Boehm aufgegriffen und erstmals als Wasserfallmodell bezeichnet.[[6]](#footnote-6) Dieser Name steht für die fließende Bewegung von einer Phase zur nächsten und ist damit auf den Grundgedanken des Modells zurückzuführen.

Das Wasserfallmodell verläuft in sieben Phasen. Begonnen wird mit einer Vorstudie, welche sich mit der Definition und Analyse des Problems, das durch die Software gelöst werden soll, beschäftigt. In der anschließenden Phase der „Systemspezifikation“ werden die Anforderungen an das System gesammelt und festgehalten. Außerdem werden die Anforderungen auf Machbarkeit, Priorität und Vollständigkeit geprüft. In der Phase des Grob-Designs werden die Strukturen der Software- sowie der Hardware-Architektur festgelegt. Es werden bereits erste Entwürfe für Benutzer-Handbücher und Testpläne entwickelt. Im Fein-Design wird die bisherige Design-Ausarbeitung in den Punkten Datenstruktur, Oberflächenanforderungen und Algorithmen überarbeitet. Idealerweise werden auch Klassen- oder Ablauf-Diagramme genutzt. Während der Implementierungsphase wird die eigentliche Programmierung vorgenommen, die mit dem Testen endet. Die entwickelte Software wird in dieser Phase auf ihre Korrektheit überprüft. Ist die Software funktionsfähig, wird diese durch den Kunden abgenommen und in Betrieb genommen. Oft werden noch weitere Wartungsarbeiten angeboten. Während des gesamten Entwicklungsprozesses gilt, dass jede Phase erst nach dem positiven Verlauf der Qualitätskontrolle vollständig abgeschlossen werden kann. Dieses Kontrollelement hat Boehm in das Modell von Royce integriert.[[7]](#footnote-7) Durch die Validierung wird der Nutzen geprüft und über die Verifizierung findet die Überprüfung der Vorgaben der Anforderungen statt.[[8]](#footnote-8) In der folgenden ist das Wasserfallmodell grafisch dargestellt.

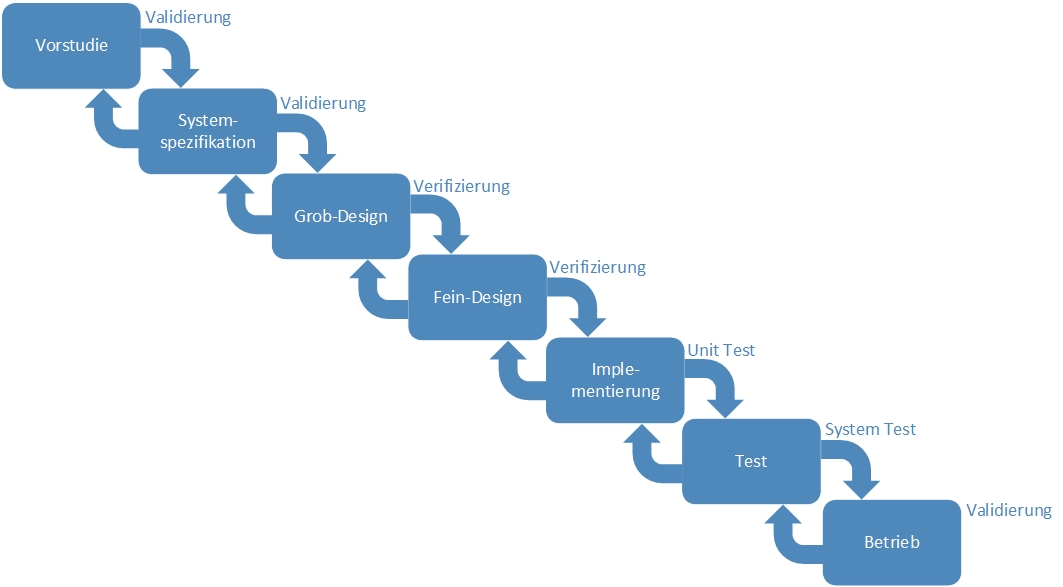


Abbildung : Das Wasserfallmodell nach Royce[[9]](#footnote-9)

Die Modelle von Boehm und Royce unterscheiden sich in der zuvor erläuterten Qualitätskontrolle, welche Royce nicht eingeplant hat. Ebenfalls waren im ursprünglichen Modell keine Rückschritte zwischen den Phasen vorgesehen. Durch diese wird in dem Modell der Rücksprung in die vorherige Phase erlaubt, wodurch die Problemstellung der nachträglichen Änderung ist gelöst.

Das Modell nach Royce setzte sich in der Praxis aufgrund der mangelnden Qualitätskontrolle nicht durch.[[10]](#footnote-10) Durch die erst weit hinten angesiedelte Überprüfung der Software können die Folgen von Mängeln so ausgeprägt sein, dass sowohl die angesetzten Kosten als auch der Zeitrahmen deutlich überschritten werden.

Das Modell nach Boehm besticht durch Einfachheit, jedoch sind an dieser Stelle auch die Grenzen des Vorgehensmodells gesetzt. Wie bereits beschrieben, entstehen durch nachträgliche Änderungen große Aufwände. So ist das Modell nur zu empfehlen, wenn alle Anforderungen zu Beginn des Projektverlaufs bekannt sind. Eine weitere Schwachstelle des Wasserfallmodells ist die Vernachlässigung sämtlicher Projektmanagementtätigkeiten.[[11]](#footnote-11)

### Agile Softwareentwicklung

Im Jahr 2001 wurde das agile Manifest als Basis der agilen Softwareentwicklung erarbeitet. Die Grundsätze des Manifests lauten:

*„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*“[[12]](#footnote-12)

Diese Grundsätze beschreiben die Flexibilität der agilen Softwareentwicklung.

Um die Ausgestaltung der agilen Softwareentwicklung hervorzuheben, wird im Folgenden eXtreme Programming (XP) erläutert.

Dieses agile Vorgehensmodell wurde im Jahr 2000 von Kent Beck erstmals vorgestellt. Beck bezieht sich mit diesem Modell auf die Werte Kommunikation, Feedback, Einfachheit sowie Mut und Respekt der einzelnen Teammitglieder.[[13]](#footnote-13) Projektspezifische Werte können durch das Team ermittelt werden, falls dies notwendig ist. Laut Beck sind diese wichtig, um einen Hintergrund und ein Ziel für die von ihm definierten Prinzipien[[14]](#footnote-14) herzustellen.[[15]](#footnote-15)

Das Team besteht bei diesem Vorgehensmodell aus Entwicklern, neben diesen wird noch die Rolle des Kunden fest definiert. Weitere Rollen, wie z.B. die eines Coachs, können in Abhängigkeit der Anforderungen des Projekts eingesetzt werden.[[16]](#footnote-16)

Bei der Anwendung von XP werden Primärpraktiken[[17]](#footnote-17) und bei erfahrenen Teams auch Folgepraktiken[[18]](#footnote-18) ausgewählt. Die sinnvolle Kombination ergibt sich dabei aus den definierten Werten und Prinzipien. Die stellt die Primärpraktiken im äußeren und die Folgepraktiken im inneren Kreis dar.

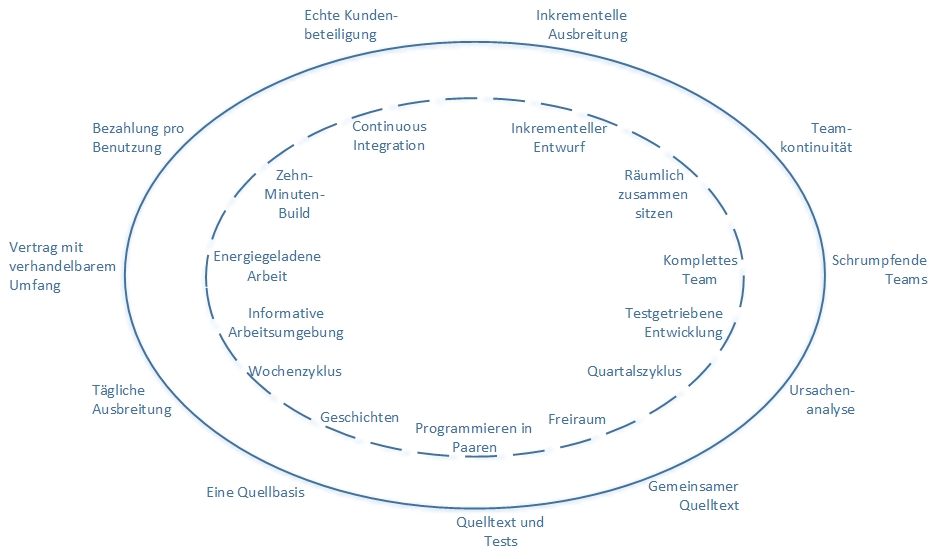


Abbildung : Die XP-Praktiken im Überblick[[19]](#footnote-19)

Die Anforderungen werden in Form von User-Stories durch den Kunden formuliert. Diese sind Beschreibungen von dem, was das System leisten soll.[[20]](#footnote-20) Der Projektverlauf mit dem Vorgehensmodell XP sieht nun vor, dass der Kunde die User-Stories für das erste Release auswählt und diese dann von den Entwicklern geschätzt werden. Nach der Schätzung wird eine Rücksprache mit dem Kunden gehalten und entschieden, ob mehr oder weniger der User-Stories bearbeitet werden sollen. Die so priorisierten User-Stories werden von den Entwicklern bearbeitet. Bei Rückfragen wird der Kunde von den Softwareentwicklern kontaktiert. Am Ende der Iteration erhält der Kunde eine Rückmeldung der Entwickler durch die Präsentation des Ergebnisses der Iteration. Dann erfolgt die Planung der nächsten Iteration. Ist ein Release fertiggestellt, so wird wieder bei der Planung des Release angesetzt.[[21]](#footnote-21)

## Behaviour-Driven Development

BDD ist ein Prinzip der Softwareentwicklung, welches seinen Ursprung im Test-Driven Development (TDD) hat. Dies wird unter anderem im Rahmen von XP als Primärpraktik genutzt.[[22]](#footnote-22)

Im Folgenden wird die Bedeutung vom Testen sowie TDD als Ursprungsform von BDD erläutert.

### Bedeutung und Herkunft

Software wird getestet, damit diese einwandfrei an den Kunden ausgeliefert werden kann. Durch Testen werden also Mängel in der Software festgestellt. Es werden die folgenden Testarten unterschieden. Normalerweise wird als erstes der Unit Test durchgeführt, dieser testet den Programmcode auf die Richtigkeit. Der Integration Test folgt, welcher die Zusammenwirkung der einzelnen Softwareelemente sicherstellt. Der System Test überprüft sämtliche Anforderungen, die vom Kunden erstellt wurden. Die funktionalen Anforderungen sind die, die sich auf die zu erfüllenden Leistungen der Software berufen. Sie betreffen somit die Funktionalität der Software. Nichtfunktionale Anforderungen dagegen beschäftigen sich mit der Qualität, in der die geforderte Funktionalität zu leisten ist. Weshalb sich die nicht-funktionalen Anforderungen auch mit dem Design der Software befassen, um beispielsweise die notwendige Performance garantieren zu können.[[23]](#footnote-23)

Es werden unterschiedliche Vereinbarungen getroffen, wann Tests durchgeführt werden. Wenn der Test als letztes durchgeführt wird, ergeben sich oft Probleme, die das Projekt sogar noch zum Scheitern verurteilen können. Denn wird ein Fehler während dieser Tests kurz vor der Auslieferung der Software entdeckt, führt dies zu enormen Kosten und auch einem hohen Zeitaufwand.[[24]](#footnote-24)

Einigt man sich in einem Projekt darauf, vor dem Schreiben des Programmcodes zu testen, müssen die Testfälle geschrieben werden, bevor die Software entwickelt wird. Dieses Vorgehen wird Test-Driven Development[[25]](#footnote-25) (TDD) genannt. Nach der Programmierung einer Funktion wird nun der zuvor erstellte Testfall durchgeführt. Erst, wenn dieser positiv verläuft, ist die Funktion fehlerfrei erstellt worden. Der Entwicklungsrhythmus verläuft wie folgt:

1. Erstellen einer Liste von Tests für die zu entwickelnde Funktion.
2. Einen Test aus der zuvor erstellten Liste auswählen.
3. An der Erstellung des Tests arbeiten.
4. Die Ausführung des Tests folgt. Allerdings wird dieser fehlschlagen, da noch kein Programmcode erstellt ist.
5. Veränderung des Programmcodes, bis der Test erfolgreich verläuft.
6. Restrukturierung des Programmcodes im Hinblick darauf, dass er kompakt formuliert ist und allen Standards der Programmierung entspricht.
7. Erneute Durchführung der Tests, um sicherzustellen, dass kein Problem durch die Restrukturierung entstanden ist.

Ist dieser Rhythmus abgeschlossen, wird wieder am Punkt 2 angesetzt, umso die gesamte Anforderung zu implementieren.[[26]](#footnote-26)

Die Kurzvariante von TDD nennt sich Test-Code-Refactor. Bei dieser wird ein Red-Green-Refactor Cycle verfolgt. Dieser Kreislauf ist durch das Schreiben eines Tests bestimmt. Da der Test vor dem Programmcode so geschrieben wird, schlägt dieser zu Anfang fehl. Der Programmcode, der den Test bestehen soll, wird erst, nachdem der Test fehlschlägt, erstellt. Ist der Programmcode ausreichend geschrieben, dass er den Test besteht, folgt die Überarbeitung. Dopplungen sollten nicht mehr im Programmcode vorhanden sein und alle Standards sollten umgesetzt sein.[[27]](#footnote-27) Dieser Kreislauf ist im Folgenden anhand der Abbildung 4 dargestellt.

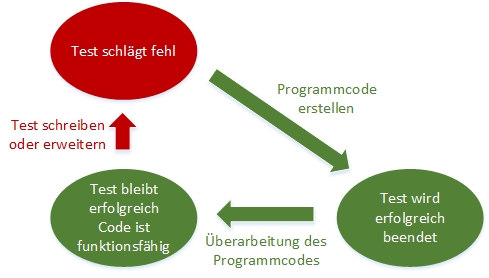


Abbildung : Der Red-Green-Refactor Cycle[[28]](#footnote-28)

### Vorgehensweise bei BDD

Die Zielsetzung von BDD ist die schnell verfügbare Software mit wenig Fehlern. Sowohl Zeit und Kosten sollen in Bezug auf die Programmierung als auch auf die Wartung reduziert werden.[[29]](#footnote-29) Dabei wird die Kommunikation zwischen den unterschiedlichen Beteiligten an einem Softwareentwicklungsprojekt betont.[[30]](#footnote-30) Durch diesen Aspekt ist BDD in die agile Softwareentwicklung als Prinzip einzuordnen.[[31]](#footnote-31)

Die grundlegende Funktionsweise von BDD baut auf die unterschiedlichen Verhaltensweisen einer Software, die von den Stakeholdern[[32]](#footnote-32) beschrieben werden, auf. Diese Beschreibungen sind in der natürlichen Sprache formuliert und stellen direkt auch den Kern von BDD dar. Es wird sichergestellt, dass alle Beteiligten das gleiche Verständnis von den erstellten Anforderungen haben. Um die Anforderung festzuhalten, werden User-Stories formuliert.[[33]](#footnote-33) Ein Beispiel für eine User-Story ist in der dargestellt.

|  |
| --- |
| User-Story: Aufruf der Seite „Kontakt“ eines Onlineshops |
| Als Kunde eines Onlineshops |
| Möchte ich die „Kontakt“-Seite aufrufen |
| So, dass ich das Unternehmen kontaktieren kann |

Tabelle : Beispiel für eine User-Story[[34]](#footnote-34)

Die Stories enthalten immer dieselbe Struktur und so können diese leicht mit Hilfe von einfachen Fragen beantwortet werden.

|  |
| --- |
| Titel der User-Story |
| Wer ist der Stakeholder dieser Story? |
| Was möchte der Stakeholder tun? |
| Welcher Mehrwert ergibt sich aus dem Handeln für den Stakeholder? |

Tabelle : Aufbau einer User-Story[[35]](#footnote-35)

Auf der Grundlage der erstellen User-Stories, erarbeiten die Entwickler nun die Funktionen des Systems.

Des Weiteren werden Szenarien formuliert. Diese beinhalten, wie sich das Systems verhalten soll, um die User-Stories erfüllen zu können. Dazu werden drei Informationen festgehalten: Die erste ist die „Given“ Information. Diese besteht aus dem Kontext, aus dem ein Benutzer im folgenden Verlauf handeln wird. Die zweite ist die „When“-Information. Diese Information enthält alles über das Verhalten des Benutzers, also was tut der Benutzer. Bei der dritten Information handelt es sich um das Ergebnis, dass die Software zurückliefern soll, nachdem der Benutzer den zweiten Schritt ausgeführt hat. Also welches Verhalten wird erwartet, wenn der Benutzer etwas in einer bestimmten Situation tut.[[36]](#footnote-36) Die folgende verdeutlicht dieses anhand eines Beispiels.

|  |  |
| --- | --- |
| Szenario: Seite einer Website öffnen | |
| Given | Der Kunde befindet sich auf der Seite „Home“ der Website |
| When | Button „Kontakt“ wird angeklickt |
| Then | Die Seite „Kontakt“ wird geöffnet |

Tabelle : Beispiel für ein Szenario[[37]](#footnote-37)

Daraus ergibt sich, dass bei BDD die Abnahmekriterien bereits vor der eigentlichen Programmierung erstellt werden.[[38]](#footnote-38) Die Business-Verantwortlichen müssen von den Entwicklern unterstützt werden und anders herum, um alle Kriterien umzusetzen. So wird der Softwareentwicklungsprozess gemeinsam erfolgreich durchgeführt.[[39]](#footnote-39)

Ein weiterer Punkt ist die Automatisierung des Akzeptanztests oder auch Spezifizierungscode genannt. Es handelt sich dabei um messbare Spezifizierung des Verhaltens des Systems. Für die erfolgreiche Ausführung ist es notwendig, dass die Testmanagementsoftware die Kriterien für ein positives Ergebnis automatisch aus den Szenarien erkennt und analysiert. Dafür ist eine Hinterlegung von Regeln notwendig, welche beinhalten, wie die Testmanagementsoftware vom Szenario zu dem Programmcode, der zur Spezifizierung der Software notwendig ist, gelangt. Die Hinterlegung von Regeln werden als Mapping bezeichnet. Wie genau dieser Prozess stattfindet, ist je nach verwendeter Testmanagementsoftware unterschiedlich.

Außer der Formulierung der Anforderungen sowie der erwünschten Ergebnisse wird deutlich, dass im Rahmen von BDD auf die Verwendung des Wortes „Test“ verzichtet wird. So wird sichergestellt, dass zu jedem Zeitpunkt jeder Beteiligter versteht, wie die Software sich verhalten soll. Dies hat auch psychologische Gründe, denn viele Stakeholder sind irritiert, sobald fachspezifische Begriffe genannt werden. Somit bezweckt die Vermeidung des Wortes die kollaborative Arbeit innerhalb des Teams.[[40]](#footnote-40)

### Vor- und Nachteile

Nachdem nun ein Einblick in die Praktik BDD gegeben wurde, sollen die Vor- und Nachteile zusammengefasst dargestellt werden.

Vorteile:

* Es werden keine besonderen Fähigkeiten oder Kenntnisse benötigt, um die Software auf das richtige Verhalten zu überprüfen.
* Die Anforderungen sind durch Beispiele hinterlegt so können sämtliche Missverständnisse umgangen.
* Sämtliche Stakeholder werden in den Entwicklungsprozess durch ständige Kommunikation einbezogen, so entsteht ein zusammenarbeitendes Team.
* Für die meisten Beteiligten ist es leichter, die Fragestellung, was soll das System tun, zu beantworten, als die Frage, wie soll eine bestimmte Anforderung implementiert werden.

Nachteile:

* Die Stakeholder, besonders der Kunde, muss kooperationsbereit sein und im Team mitarbeiten wollen, ansonsten kann die Kommunikation zur Hürde werden.
* Es muss ein Umdenken der Programmierer stattfinden, diese dürfen nicht weiter ihre Funktionen direkt in Algorithmen ausdrücken, sondern müssen diese zunächst in natürlicher Sprache definieren.

## Der Konflikt innerhalb der IT-Organisation

Die IT-Organisation gliedert sich in die Softwareentwicklung (Dev) und den IT-Betrieb (Ops) auf.

Der IT-Betrieb ist für die Verfügbarkeit der Software in der Umgebung des Endnutzers verantwortlich. Somit versucht der IT-Betrieb eine Anwendung, die zuverlässig in der Verfügbarkeit ist, möglichst vor Änderungen zu schützen, um das Risiko des Ausfalls zu minimieren.

Die Wertschöpfung der Entwicklung dagegen ist die Umsetzung der Anforderungen und die damit verbundene Erweiterung der Software. Der Kunde erwartet von der Entwicklung, dass alle von ihm gestellten Anforderungen möglichst schnell umgesetzt werden. So werden seitens der Entwicklung häufig neue Funktionen entwickelt und Änderungen in der Software vorgenommen.[[41]](#footnote-41)

Somit ist deutlich, dass ein Interessenkonflikt zwischen den Parteien Anwendungsentwicklung und IT-Betrieb entsteht. Diesen Konflikt und die damit einhergehende Isolation verdeutlicht die folgende Abbildung 5.

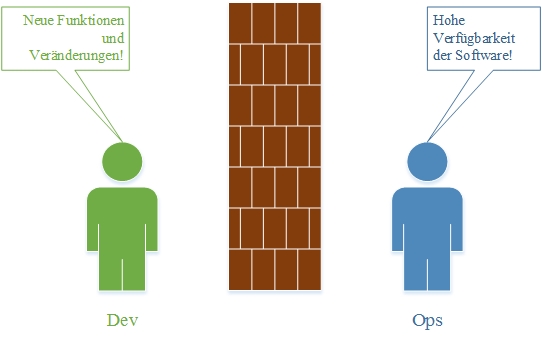


Abbildung : Die Isolation innerhalb der IT-Organisation[[42]](#footnote-42)

Der Konflikt entsteht nachdem die Änderungen in einer Testumgebung auf die Probe gestellt wurden. Die Tests sollen der Softwareentwicklung die Sicherheit geben, dass die Software einwandfrei ist und dem IT-Betrieb weitergegeben werden kann, damit dieses Team die Software in die Produktivumgebung einsetzt.

Nachdem der IT-Betrieb dies umgesetzt hat, ist die Verfügbarkeit der Software häufig nicht mehr garantiert, da Fehler, welche in der Testumgebung noch nicht deutlich waren nun erkannt werden.

An diesem Punkt beginnen dann die Auseinandersetzungen zwischen dem IT-Betrieb und der Entwicklung. Die Argumente der Softwareentwicklung gegen den IT-Betrieb beziehen sich auf die Lauffähigkeit der Software in der Testumgebung, dagegen spricht jedoch für den IT-Betrieb, dass die Verfügbarkeit in der Produktivumgebung nicht mehr gewährleistet ist. Beide Seiten verfolgen nur ihre Interessen und es wird durch diesen Konflikt häufig nicht an einem gemeinsamen Ziel gearbeitet. Diese entstehende Uneinigkeit und die daraus resultierenden Konflikte werden als „Blame Game“ bezeichnet.[[43]](#footnote-43)

## DevOps im Unternehmen

Der Begriff DevOps besteht aus den englischen Worten Development und Operations, also die Anwendungsentwicklung und der IT-Betrieb. Der Begriff ist 2009 während einer Konferenz in Belgien entstanden und steht für die Vereinigung der unterschiedlichen Teams innerhalb der IT-Organisation. Eindeutig definiert ist DevOps derzeit nicht, so definiert Verona in seinem Buch „Practical DevOps“ wie folgt: „DevOps ist ein Bereich, der aus unterschiedlichen Disziplinen besteht. Es ist ein Bereich, der sehr praktisch ausgelegt ist, es müssen aber der technische Hintergrund als auch die nichttechnischen kulturellen Aspekte verstanden werden.“[[44]](#footnote-44) Ein Mitarbeiter der IBM definiert DevOps in einem Blogg dagegen etwas anders. „DevOps ist die unternehmerische Fähigkeit die kontinuierliche Bereitstellung von Software zu nutzen, um den Zeitpunkt zum Kunden-Feedback zu reduzieren und dem Kunden durch schnell verfügbare Software neue Marktchancen zu ermöglichen.“[[45]](#footnote-45)

Um eine Definition für DevOps aufzustellen, ist es notwendig erstmal die grundliegenden Elemente von DevOps zu verstehen.

Der Kern von DevOps beschäftigt sich mit der Schaffung eines integrierten Prozesses der Anwendungsentwicklung und des IT-Betriebs, es soll ein funktionsübergreifendes Team gebildet werden. Das gemeinsame Ziel ist es, Anwendungen frühzeitiger zur Verfügung zu stellen und einwandfrei zu entwickeln.[[46]](#footnote-46) Dies verlangt vor allem die heutige Marktsituation. Die Kunden erwarten aber nicht nur frühzeitige Lieferungen, sondern ebenfalls kurzfristige Reaktionen auf Änderungen.

Es wird kein feststehender Prozess vorgeschrieben, sondern der Fokus liegt bei der Zusammenarbeit der Beteiligten, sowohl der eingesetzten Prozesse und Werkzeuge als auch der Mitarbeiter.[[47]](#footnote-47) Damit ist DevOps nicht das Ziel eines Unternehmens, sondern DevOps hilft lediglich, die Ziele des Unternehmens zu erreichen. Die Reichweite von DevOps geht über Anwendungsentwicklung und IT-Betrieb hinaus, es sollten sämtliche Stakeholder bei diesem Prozess integriert werden. So lässt sich ein übergreifender Plan entwickeln, der die Prozesse und Tools sowie die Unternehmenskultur beschreibt.

DevOps beinhaltet aber keinesfalls die Bildung einer neuen Abteilung. Es ist mehr eine Umstrukturierung der bestehenden Abteilungen. Die Herausforderung besteht also darin, die Zusammenarbeit von Anwendungsentwicklern und Mitarbeitern des IT-Betriebs zu stärken.[[48]](#footnote-48) Jegliche Barrieren innerhalb der IT-Organisation sollen reduziert und so die reibungslose Zusammenarbeit garantiert werden.

Diese Inhalte von DevOps führen zu der folgenden Definition: „DevOps bildet die Vereinigung der unterschiedlichen Abteilungen der IT-Organisation ab. Von diesem funktionsübergreifenden Team werden Methoden eingesetzt, die zu häufigeren Releasen und somit häufigerem Feedback der Stakeholder führen.“

### Kernelemente von DevOps

Die Kernelemente von DevOps bilden sich einerseits aus der Kultur und andererseits aus der Standardisierung von Prozessen und Tools.

Um eine DevOps-Kultur einzuführen, ist die Auflösung der Isolation der einzelnen Teams innerhalb der IT unbedingt notwendig.[[49]](#footnote-49) Ein andauernder Lernprozess sollte innerhalb des Unternehmens geschaffen werden. Dafür sind Rückmeldungen unbedingt notwendig, wie Swartout in dem Buch „Continuous Delivery and DevOps: A Quickstart Guide“ erläutert. Besonders wichtig für Rückmeldungen sind die Faktoren Offenheit und Ehrlichkeit.[[50]](#footnote-50) In einem Unternehmen der DevOps-Kultur sollten diese Werte unbedingt ein Hauptaugenmerk bilden. Um einen andauernden Lernprozess zu etablieren, sollen Fehler erlaubt oder sogar erwünscht sein. Dabei sollte es allerdings keine Schuldzuweisungen geben, denn so würde eine negative, demotivierende Stimmung entstehen, welche die Produktivität sehr einschränkt.[[51]](#footnote-51) Zur Umsetzung dieser Kultur stellt neben Offenheit und Ehrlichkeit auch Respekt gegenüber anderen Mitarbeitern einen wesentlichen Anteil dar. Denn wenn kein Respekt untereinander herrscht, ist es kaum möglich, eine solche Kultur der Zusammenarbeit zu schaffen.[[52]](#footnote-52)

Durch den Einsatz von standardisierten Tools wird die Arbeit erleichtert.[[53]](#footnote-53) So ist ein zentrales Tool zur Softwareverwaltung mit einer integrierten Versionsverwaltung für ein Projektteam zu empfehlen. Dadurch ist sichergestellt, dass alle Beteiligten über den aktuellen Stand der Software informiert sind und auch an diesem arbeiten können. Informationen liegen somit offen und die Transparenz kann garantiert werden. Durch den Einsatz von Tools, die Automatisierungen erstellen anstelle von manueller Arbeit, wird die Kontrolle trotz der Geschwindigkeit innerhalb der Erstellung gewährleistet. [[54]](#footnote-54)

Die Standardisierung von Prozessen im Zusammenhang mit DevOps erstellt einen verbindlich vorgegebenen Weg, wie die Entwicklung bearbeitet wird. Hierbei ist allerdings zu beachten, dass dies bei jedem Projekt unterschiedlich ist. An dieser Stelle werden die eingesetzt Methoden also festgelegt, so dass zum Beispiel darüber entschieden wird, welche agile Vorgehensweise genutzt werden soll.[[55]](#footnote-55)

### Agilität von DevOps

Mit dem Einsatz von agilen Softwareentwicklungsmethoden wird die Schnelligkeit der Auslieferungen garantiert. Wenn IT-Betrieb und Anwendungsentwicklung jedoch nicht zusammenarbeiten, stellt sich die Frage, welchen Zweck die regelmäßigen Auslieferungen durch die Entwicklung haben, wenn der IT-Betrieb diese nicht zur Verfügung stellt, sondern neue Funktionen beispielsweise nur einmal im Quartal an den Kunden ausliefert? Diese Fragestellung wird durch die Integration der beiden Organisationseinheiten gelöst, denn so wird die Produktivität aber auch die Qualität der Software sichergestellt.[[56]](#footnote-56) Somit ist eindeutig, dass agile Softwareentwicklung eigentlich nur durch den Einsatz von DevOps vollständig umgesetzt werden kann.[[57]](#footnote-57) Es lässt sich sagen, dass die agilen Softwareentwicklungsmethoden die technischen Vorgehensweisen agil umsetzen. Durch den Einsatz von DevOps wird nun auch der menschliche Aspekt in den agilen Zyklus eingebunden, denn es sind keine Barrieren zwischen Dev und Ops mehr vorhanden.[[58]](#footnote-58)

## Die DevOps Praktiken

Es werden fünf Praktiken bei DevOps zur Ausgestaltung des gesamten Softwareentwicklungsprozesses angewendet. Kern dieser Praktiken sind die Automatisierungen und Standardisierungen.[[59]](#footnote-59)

In der wird die Reichweite der einzelnen Praktiken grafisch dargestellt und im Anschluss erläutert.



Abbildung : Die Reichweiten der verschiedenen DevOps Praktiken[[60]](#footnote-60)

### Continuous Integration

Der Prozess von Continuous Integration[[61]](#footnote-61) (CI) beschreibt das Vorgehen der Integration der neuen Programmcode Elemente in die aktuelle Version des Programms. Dieser Prozess sollte in kontinuierlichen und kurzen Zeitabständen stattfinden, um Probleme oder Fehlerquellen schneller identifizieren und beseitigen zu können.

Sobald durch einen Entwickler eine Änderung im Programmcode vorgenommen wurde und der Entwickler die Änderung an den Source Control Server übergibt, startet ein CI-Server automatisch oder nach einem angegebenen Zeitfenster ein Build. Ein Build ist dabei ein Abbild der Software. Nach der Fertigstellung des Builds wird die Logik des Programmcodes getestet. Dieser Test wird als Unit Test bezeichnet. Das Ergebnis wird an den CI-Server zurückgegeben, welcher dann das Feedback, oft in Form einer grafischen Aufbereitung, an die Entwickler weitergibt. Ist das Ergebnis positiv, kann der Programmcode freigegeben werden. Wenn der CI-Server allerdings ein negatives Ergebnis meldet, müssen die Entwickler den Fehler beheben und einen neuen Build anstoßen.[[62]](#footnote-62) Dieser Ablauf ist in der dargestellt.

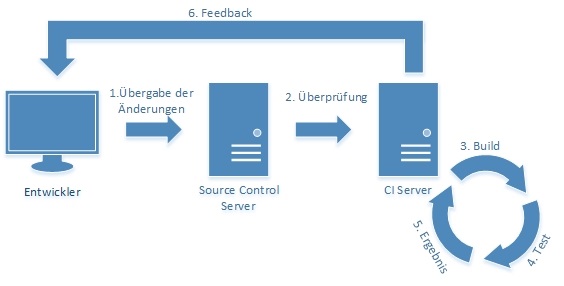


Abbildung : Der Continuous Integration-Prozess[[63]](#footnote-63)

Durch CI wird somit die Problemstellung der reibungslosen Zusammenarbeit innerhalb der Entwicklung gelöst. Entwickler können gleichzeitig an Branchs[[64]](#footnote-64) des Programmcodes arbeiten. Durch die kontinuierliche Integration und automatische Testfunktion ist die Einbindung des Branchs zum Master-Branch störungsfrei.

Ein Vorteil des CI Vorgehens ist außerdem, dass eine Transparenz im Programmcode dadurch entsteht, dass dieser für alle Beteiligten verfügbar ist. Es ist leicht überprüfbar, wann welcher Anteil des Programmcodes freigegeben wurde und ob er ein Problem beinhaltet hat. Außerdem ist natürlich auch das frühzeitige Feststellen von Problemen positiv zu vermerken.

### Continuous Delivery

Continuous Delivery[[65]](#footnote-65) (CD) verfolgt das Ziel, den Prozess bis zu dem neuen Release[[66]](#footnote-66) der Software zu optimieren. Ursprünglich wurde die Einführung neuer Software oft am Wochenende durchgeführt.[[67]](#footnote-67) Es wurde zwar versucht, eine lauffähige Software bereitzustellen, dennoch war dies oft nicht gegeben. Durch die manuelle Bearbeitung wurde dieser sehr komplexe Prozess noch langwieriger und ebenfalls fehleranfälliger, folglich wurden Release nur selten in der Produktion eingesetzt.

Durch die Einführung von CI werden Fehler schnell identifiziert, um das Risiko aber weiter zu minimieren, setzt CD direkt nach CI an. Die grundlegende Idee ist es, dass bei einem kleinen Release auch das Risiko des Ausfalls eingegrenzt wird. Außerdem besteht dadurch dann auch die Notwendigkeit häufigerer und regelmäßigerer Release. Dies hat für die Softwareentwicklung den positiven Effekt, dass weniger Probleme auftreten.

Die Regelmäßigkeit ist also ein wichtiger Wert von CD, um die Zuverlässigkeit in der Lieferkette zu sichern. Des Weiteren ist es notwendig, dass jede Stufe der Software und Infrastruktur nachvollziehbar ist. Dies wird durch den Einsatz eines Artefakte Repository möglich. Sämtliche Dateien, die erzeugt wurden, werden in diesem Repository gespeichert. Regressionen sollte in der Softwareentwicklung vermieden werden, diese tritt ein, wenn ein neues Softwarerelease vorhanden ist und dieses in das vorhandene System eingebunden werden muss. Dadurch werden zuvor bereits fertiggestellte Funktionen der Software erneut verändert, deshalb müssen diese Teile nochmals getestet werden. So entsteht ein kontinuierlich wachsender Testprozess, welcher nur durch Automatisierungen zu bewältigen ist.

Die folgende Abbildung 8 zeigt die CD-Pipeline, diese gliedert sich in fünf Phasen auf.



Abbildung : Die CD-Pipeline[[68]](#footnote-68)

Der erste Schritt dieser Pipeline wird „Commit“ bezeichnet und beschreibt den CI-Prozess der Erstellung der Builds und Unit Tests.[[69]](#footnote-69) In der folgenden Phase „Akzeptanztest“ wird die Interaktion des Release mit der Benutzeroberfläche getestet. Bei der Automatisierung dieser Tests besteht auch die Möglichkeit die Testfälle auf Grundlage der Anforderungen in natürlicher Sprache zu formulieren.[[70]](#footnote-70) Die dritte Phase beschäftigt sich mit der Leistungsfähigkeit der Software. Es wird die Skalierbarkeit getestet, weshalb diese Phase als „Kapazitätstest“ bezeichnet wird. Um die Skalierbarkeit und Leistungsfähigkeit zu testen, sollte die Testumgebung möglichst den Bedingungen der Produktivumgebung entsprechen. In der darauffolgenden Phase „explorative Tests“ werden die Funktionen und das Verhalten der Software auf die unvorhersehbare Benutzung geprüft. An diesem Punkt ist ein automatisierter Test nicht sinnvoll. Anstelle dessen sollten Experten eingesetzt werden, die jegliche Verhaltensweisen der zukünftigen Benutzer kennen. In der letzten Phase der Pipeline, namens „Produktion“, wird die Anwendung dann manuell installiert und die Dokumentation beendet.[[71]](#footnote-71) Nachdem diese Phase abgeschlossen ist, findet ein Feedback mit dem Kunden statt, wodurch die Zufriedenheit gemessen wird.[[72]](#footnote-72)

Sollte diese Pipeline an einer Phase scheitern, so ist die Software zu überprüfen. Der Fehler ist dann zu bereinigen, bevor die CD-Pipeline erneut mit dem „Commit“ gestartet wird. Dieser Prozess wird so lange durchexerziert, bis das Release in der Produktion installiert ist und somit alle Phasen erfolgreich abgeschlossen sind.

### Continuous Deployment

Der Prozess von Continuous Deployment[[73]](#footnote-73) knüpft sich an die CD-Pipeline an. So führt bei Continuous Deployment jede Änderung im Programmcode automatisch zu einer Installastion in der Produktivumgebung.[[74]](#footnote-74) Somit befasst sich Continuous Deployment mit der Automatisierung des letzten Schritts der CD-Pipeline.[[75]](#footnote-75) Dieser Unterschied wird in der folgenden Abbildung 9 verdeutlicht.[[76]](#footnote-76)

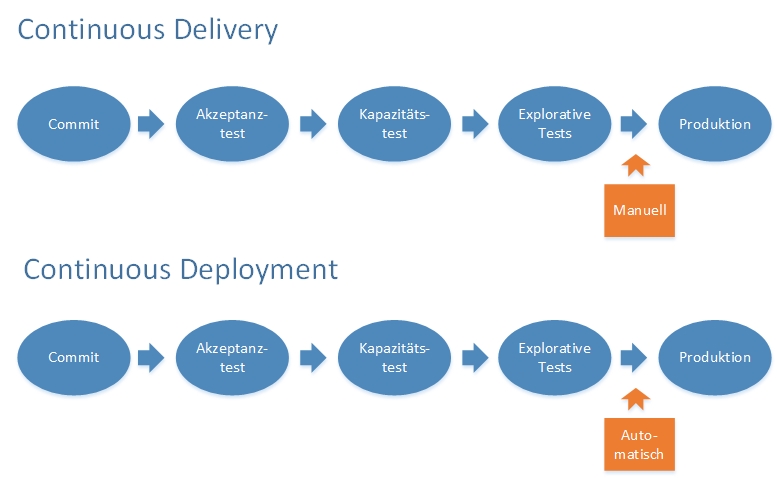


Abbildung : Die Continuous Deployment-Pipeline im Vergleich[[77]](#footnote-77)

Sobald eine Fehlerhafte Software allerdings in die Produtivumgebung installiert wird, sind die Folgen oft dramatisch. Deshalb gibt es unterschiedliche Verfahren um die Situation beherrschen zu können. Durch die Durchführung eines so genannten Rollbacks wird ein vorheriger Versionsstand der Software wiederhergestellt. Ein Rollforward dagegen sieht die schnelle Beseitigung des Fehlers, durch die Freigabe einer neuen fehlerfreien Version vor.[[78]](#footnote-78)

Vorteil der automatischen Installation in der Produktivumgebung ist das häufigere Feedback durch den Kunden. Fehlfunktionen werden so früher rückgemeldet und es entsteht eine Minimierung des Risikos. Außerdem ist es möglich, dass nur Teile einer Funktion in der Produktivumgebung freigegeben werden. So kann für diese bereits eine Kundenrückmeldung eingeholt werden und die weitere Programmierung baut auf keinen fehlerhaften Programmcodeelementen auf.[[79]](#footnote-79)

### Continuous Testing

Bei der vierten DevOps Praktik handelt es sich um Continuous Testing[[80]](#footnote-80) (CT).

In der CD-Pipeline[[81]](#footnote-81) sowie in der Continuous Deployment-Pipeline[[82]](#footnote-82) wird bereits deutlich, in welchem Detailgrad dies praktiziert wird. Im CI-Prozess wird die Logik des Programmcodes bereits durch einen Unit Test geprüft. Nach Erfolg dieses Tests wird die Integration der einzelnen Bestandteile der Software überprüft. Ist die fehlerfreie Zusammenarbeit der einzelnen Komponenten gewährt, wird die Software in eine Testumgebung, die weitgehend der Produktionsumgebung entspricht, überführt und hier die weitere Qualitätssicherung vorgenommen.

Es wird in funktionale und nicht-funktionale Tests unterschieden. Bei den funktionalen Tests besteht das Ziel darin, die Anforderungen an die Funktionen der Software zu überprüfen, dagegen wird bei den nicht-funktionalen Tests das Verhalten der Software in der Umgebung beurteilt.[[83]](#footnote-83)

Im Bereich des funktionalen Testens werden beispielsweise die einzelnen Komponenten der Software unabhängig voneinander überprüft. Es wird aber auch das Zusammenspiel der Komponenten untereinander sichergestellt. Die Tests „Akzeptanztest“ und „exploratives Testen“ aus der CD-Pipeline sind außerdem in diesem Bereich angesiedelt.[[84]](#footnote-84)

Beim nicht-funktionalen Testen wird unter anderem die Performanz des Systems wie auch die Komptabilität der für die Software eingesetzten Systeme geprüft. Besonders wichtig bei den nicht-funktionalen Tests sind die Sicherheitstests. Bei diesen wird überprüft, ob es Lücken in der Software gibt, durch die beispielsweise ein Hacker in das System eindringen könnte.[[85]](#footnote-85)

### Continuous Monitoring

Continuous Monitoring[[86]](#footnote-86) (CM) liefert durch Metriken und Daten Aufschluss über den aktuellen Status der Software. Hierbei ist das Ziel, vor allem Fehler zu erkennen, Performance Probleme zu diagnostizieren und auch widerrechtliches Eindringen in die Software festzustellen.[[87]](#footnote-87) Gemessen werden diese Werte an festgelegten Standardkennzahlen.

Um CM durchzuführen, werden verschiedene Komponenten benötigt. Eine ist die Extraktion der Daten. Eine andere Komponente wird benötigt, um die geladenen Daten zu analysieren. Ein drittes System wird zur Weiterverarbeitung der Ergebnisse der Analyse eingesetzt, also beispielsweise zur Erstellung von Grafiken oder zur Workflow Modellierung.[[88]](#footnote-88) Dieser Prozess wird in der Abbildung 10 verdeutlicht.



Abbildung : Der Monitoring-Prozess[[89]](#footnote-89)

Durch die kontinuierlichen Änderungen sollte dieser Prozess automatisiert an die aktuelle Software angepasst werden. Dies bedeutet, wenn beispielsweise Änderungen in den Einstellungen der Server vorgenommen werden, dass diese Änderungen im Monitoring beachtet werden und nicht zu einem Alarm führen.[[90]](#footnote-90) Verläuft die Überwachung der IT-Systeme konstant, so werden Veränderungen jeglicher Art verzeichnet und es kann auf diese individuell reagiert werden.

## Vor- und Nachteile von DevOps

Nachdem nun ein Einblick in den thematischen Bereich von DevOps gegeben wurde, sollen die Vor- und Nachteile zusammengefasst dargestellt werden.

Vorteile:

* Regelmäßige Softwarerelease führen zu einer flexibleren Bereitstellung der Software beim Kunden und zu einer Risikoreduzierung durch die Minimierung der Komplexität der Release.
* Es wird ein regelmäßiges Feedback durch den Kunden ermöglicht.
* Automatisierung erleichtert den Prozess für die Entwickler aber auch für den Betrieb. Außerdem wird das Risiko weiter reduziert, durch die Vermeidung von manueller Arbeit.
* Die Kundenzufriedenheit wird außerdem durch die Stabilität der Verfügbarkeit der Software weiter gesteigert.

Nachteile:

* Es entstehen hohe Kosten durch die notwendigen Schulungs- und Teambildungsmaßnahmen.
* Die Einführung der Praktiken ist sehr zeitaufwändig.
* Mitarbeiter können sich durch die hohe Transparenz innerhalb des Teams schnell überwacht fühlen.

Sofern ein Unternehmen den Nutzen des Agierens eines funktionsübergreifenden Teams erkennt, ist es sinnvoll, DevOps einzuführen. Es ist natürlich abzuwägen, welche Praktiken dabei für das betreffende Unternehmen einzuführen sind.

## Gegenüberstellung von BDD und DevOps

Der Fokus von BDD liegt in der Erarbeitung von Anwendungsfällen gemeinsam mit allen Stakeholdern. Mittels dieser Anwendungsfälle wird dann der Programmcode geschrieben. Durch dieses Vorgehen soll sichergestellt werden, dass alle vom Stakeholder beschriebenen Funktionen in der Software implementiert werden. Die Kommunikation innerhalb des Teams ist durch die Verwendung der natürlichen Sprache zur Formulierung der Anwendungsfälle gesichert. Erst bei der Programmierung der einzelnen Elemente der User-Stories werden allein die Softwareentwickler tätig und sorgen für eine Automatisierung der Szenarien. Das kollaborative Arbeiten findet also im Bereich BDD besondere Beachtung.

Im Bereich DevOps wird ebenfalls die gemeinsame Arbeit fokussiert. Hier wird dies umgesetzt, in dem ein funktionsübergreifendes Team an versiert wird. Durch Wissensaustausch wird dieses Team vor allem generiert. Das Testen im Rahmen von DevOps wird in Form von Continuous Testing als eine der Praktiken durchgeführt, so dass eine stabile und funktionsfähige Software entwickelt wird. Außerdem bildet das Automatisieren von Prozess ein Kernelement. Es soll möglichst wenig manuelle Arbeit verrichtet werden. Durch dieses Vorgehen werden Fehler vermieden und das Risiko minimiert.

# Praktische Umsetzung von Behaviour-Driven Development

Im Folgenden Kapitel wird BDD anhand einer Projekts des Unternehmens IBM vorgestellt.

## Entwicklung von Testfällen mit Cucumber

Cucumber ist ein OpenSource Tool, welches seinen Einsatz im Bereich BDD findet. Es lassen sich textuelle Spezifikationen von Anforderungen an eine Software formulieren, welche durch das Tool automatisiert überprüft werden. So wird eine Brücke zwischen technischen und Business Verantwortlichen in einem Projekt geschlagen. Das Tool wird dabei über die Kommandozeile bedient.

Cucumber wurde ursprünglich in der Programmiersprache Ruby geschrieben und ebenfalls für Ruby Anwendungen eingesetzt. Inzwischen unterstützt Cucumber allerdings auch andere Programmiersprachen wie Java, C++ oder JavaScript.[[91]](#footnote-91)

Die in diesem Tool verwendete Beschreibungssprache nennt sich Gherkin. Das folgende Beispiel zeigt die Syntax dieser Beschreibungssprache.

Gherkin verfügt über unterschiedliche Schlüsselwörter, welche eingesetzt werden, um die unterschiedlichen Szenarien zu beschreiben. Diese Schlüsselwörter können ebenfalls in unterschiedlichen Sprachen genutzt werden, Deutsch und Englisch werden beispielsweise durch Gherkin unterstützt.

Um Spezifikationen über Cucumber funktionsfähig und automatisch zu generieren, werden unterschiedliche Dateien benötigt. Die gesammelten User-Stories, die bei Cucumber Features genannt werden, werden gemeinsam mit den definierten Szenarien inklusive der Steps „Given“, „When“ und „Then“ in einer Datei gespeichert. Diese Datei enthält somit nur natürliche Sprache. Im Folgenden ist dafür ein Beispiel dargestellt.

*Feature: Open the contact site of an onlineshop*

*As a customer of an onlineshop*

*I want to open the Site “Contact”*

*So that I can contact the shop*

*Scenario: Open the Site*

*Given the customer is on the „Home“-Website of the onlineshop*

*When he clicks on the button „Contact“*

*Then he would see the contact details*

Danach werden die Programmierer des Teams tätig. Es wird eine Step Definition erstellt, die durch die gleiche Benennung der Methoden und der Szenarien Verweise zwischen der natürlichen Sprache und dem Programmcode generiert. Außerdem können alle Ausdrücke, die in dem Szenario in Anführungszeichen geschrieben wurden in der Step Definition verwendet werden. Die zuvor ausgeschriebenen Elemente werden einer Variable zugeordnet durch den Ausdruck {VARIABLE:stringInDoubleQuotes}. Die Variable kann dann in die folgende Funktion übergeben werden. Durch dieses Vorgehen wird das Ausschreiben von Textelementen erspart und die Programmierung erleichtert. Außerdem erfolgt eine sinnvolle Navigation innerhalb des Programmcodes durch festgelegten Variablen.

Der Support Code ist für die Ausführung des Programms notwendig. Er enthält die notwendigen Befehle, wie z.B. der Aufruf eines Treibers, welcher für die Ausführung der Testfälle benötigt wird.[[92]](#footnote-92)

Nach der Erstellung des Support Codes ist dann das System ausführbar.

Der Ablauf der Erstellung der unterschiedlichen Elemente ist in der folgenden Abbildung 11 dargestellt.

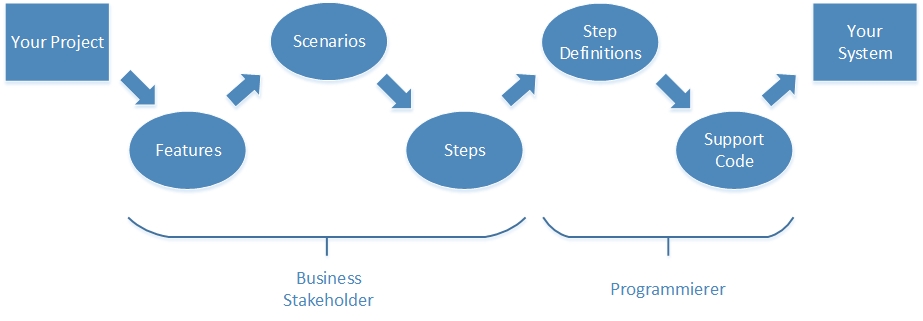


Abbildung : Die Funktionsweise von Cucumber[[93]](#footnote-93)

Wenn das Tool alle Steps einwandfrei durchlaufen hat, wird die Ausgabe in der Konsole grün markiert und ist somit positiv verlaufen. Ist allerdings ein Fehler auftreten, wird ein Szenario in Rot als fehlgeschlagen, in Orange als Andauernd in Orange oder in Gelb als nicht auffindbar markiert.

## Vorstellung des Projekts

Das Projekt wurde im Rahmen der „DevOps Backbone Initiative“ umgesetzt. Es handelt sich dabei um ein Tool zur Datenhaltung und Datenermittlung der DevOps-Skills der IBM-Mitarbeiter. So sollen diese künftig ihre Kompetenzen im Bereich DevOps anhand dieses Tools speichern. Hierbei sind nicht nur die DevOps Praktiken relevant, sondern auch jegliche Erfahrungen mit Tools im DevOps Bereich sollen ausgewählt werden können. Außerdem sind bestimmte DevOps Rollen innerhalb der IBM vorgesehen. Der Mitarbeiter muss aus diesem Grund auch angeben welche Rolle er im DevOps Kontext ausübt. Es werden also Rollen keine, Berater, Architekt, Release Manager und Implementation Spezialist unterschieden.

Bevor dieses Projekt begonnen hat, wurde die Auswertung der Skills mühsam per Microsoft Excel Dateien gepflegt. Jeder Mitarbeiter erhielt einen Fragebogen in Form einer Excel-Datei, welche nach dem Ausfüllen und der Rücksendung per E-Mail in eine Datenbank aufgenommen wurde. Änderungen waren somit nur durch die Mitarbeiter, welche das Einpflegen der Daten übernommen hatten, möglich. Mit dem neuen Tool sollen die Mitarbeiter ihre Skills selbstständig verwalten können.

Die Zielstellung dieses Tools ist somit die vereinfachte Auswahl von Mitarbeitern für Projekte, in denen bestimmte Rollen mit bestimmten Kompetenzen noch nicht besetzt sind.

## Anforderungen an das Tool

Die Anforderungen an das Tool wurden durch Gespräche mit den Stakeholdern gesammelt und festgehalten. Es wurde anhand von Beispielen festgemacht, welche Anforderungen priorisiert werden. So soll der Login über den IBM-internen Single Sign-On[[94]](#footnote-94) Service durchgeführt werden. Das Besondere an diesem ist die Notwendigkeit des Intranets[[95]](#footnote-95) der IBM. Ohne die Anbindung ist ein Login also nicht möglich. Nach dem Login eines Benutzers sollen automatisch seine Daten abgebildet werden, welche von einer Datenbank aufgrund der Login Daten abgerufen werden. Die Datenspeicherung oder Datenänderung ist dann ganz einfach durch den „Save“-Button und „Edit“-Button möglich. Dabei wird unterschieden ob es sich um einen Mitarbeiter handelt, der noch keine Daten gespeichert hat, dieser erhält direkt die Sicht auf den „Save“-Button. Oder der Mitarbeiter hat bereits Daten eingegeben, dann werden ihm dieser Anzeigt und die Bearbeitung ist über den „Edit“-Button möglich. Der Zeitpunkt der letzten Datenmodifikation, wird durch das Tool ebenfalls durch das entsprechende Datum abgebildet. Löschen soll der Mitarbeiter seine Daten ebenfalls können, dies funktioniert über einen „Delete“-Button. Nachdem anklicken des Buttons sind sämtliche Daten des Benutzers von der Datenbank entfernt.

Bei Unklarheiten soll es dem Benutzer möglich sein, eine Erklärung durch einfaches Herüberfahren mit dem Cursor angezeigt zu bekommen. Sobald der Benutzer seinen Eintrag fertig bearbeitet hat, soll er sich mittels Button ausloggen können.

Die folgende Abbildung 12 zeigt eine Übersicht des Tools der DevOps Backbone Initiative. Der „Save“-Button ist dabei in Blauer Färbung rechts oben und unten erkennbar. Oberhalb des ersten „Save“-Buttons befindet sich außerdem der „Logout“-Button. Der „Delete“-Button befindet sich mit grauer Färbung im unteren Linken Bereich des Tools. Der „Edit“-Button ist nur Sichtbar, wenn der Benutzer bereits Daten gespeichert hat, die trifft allerdings für den Benutzer „MMustermann“ nicht zu.

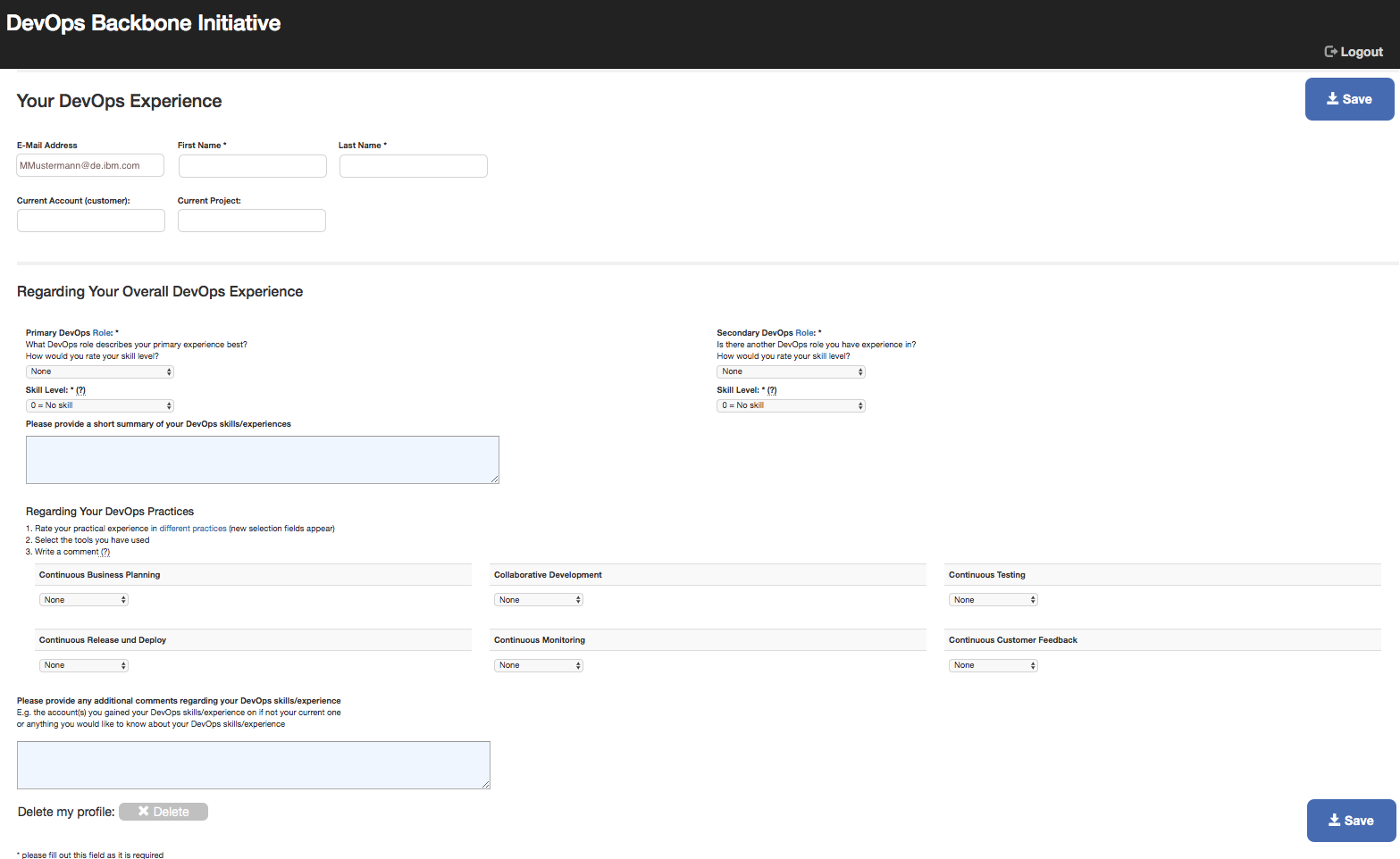


Abbildung : Das Tool im Überblick[[96]](#footnote-96)

## Definition von Features und Szenarien

Als Beispiel wird eine Anforderung dienen, weitere Testfälle werden im Anhang dargestellt. [[97]](#footnote-97)

Bei dieser Anforderung handelt sich um die Login Funktion des Tools. Diese ist notwendig, damit ein Benutzer seine Skills bearbeiten kann und kein Zugriff und keine Bearbeitung durch andere Personen möglich ist. Hieraus ergibt sich das folgende Feature:

*//features/login.feature*

*Feature: Login on „DevOps Backbone Initiative“-Website*

*As an IBM Employee*

*I want to log in on “DevOps Backbone Initiative”-Website*

*So that I can edit my DevOps skills*

Die dazugehörigen Szenarien werden außerdem in dieser Datei abgelegt. Es ergibt sich für dies die Situation, dass der Anwender, wenn er sich anmelden möchte, auf der Website der “DevOps Backbone Initiative” sein muss. Hier ist der Login mit der IBM-internen User ID und dem dazugehörigen Passwort möglich. Anschließend muss der „Sign In“-Button angeklickt werden, um dann auf die Profilseite zu gelangen. Allerdings ist auch möglich, dass der User ein falsches Passwort eingibt und dann eine Fehlermeldung erhält. Um beide Szenarien in dem Feature abzubilden müssen die Informationen, die bei beiden Szenarien gleich sind nämlich, dass der Anwender auf die Website geht und dort die User ID eingibt, als „Background“ definiert werden. Alle weiteren Informationen sind dann wieder dem jeweiligen Szenario zugeordnet. Im Folgenden ist der Background und die Szenarien dargestellt.

*Background:*

*Given As a IBM Employee I am on the Website "DevOps Backbone Initiative"*

*When I fill in my User ID*

*Scenario: Successful Login on “DevOps Backbone Initiative”-Website*

*When I fill in my password*

*When I click on “Sign In”*

*Then I should see my Profile on "DevOps Backbone Initiative"*

*Scenario: Fail Login on “DevOps Backbone Initiative”-Website*

*When I fill in a wrong password*

*When I click on “Sign In”*

*Then I should see an Error Message*

## Erläuterung der Step Definition

In der Step Definition wird festgelegt, welche Steps durchgeführt werden müssen, damit das Programm lauffähig ist und die Tests erfolgreich abgeschlossen werden können. Diese beinhalten damit die Programmierung der im Szenario definierten einzelnen Steps. Der Browser muss also automatisch die Website aufrufen und die Felder „User ID“ und „Password“ ausfüllen. Außerdem muss der Button „Sign In“ angeklickt werden.

Die Datei „login\_steps.js“ befindet sich dabei im Ordner „step\_definitions“, welcher ein Unterordner des Ordners „Feature“ bildet. Als erstes wird im Programmcode das starten des Treiber für Selenium festgelegt. Selenium ist ein Testtool, dass die Automatisierung von Websites unterstützt. Außerdem wird die Ausführung der Methode „defineSupportCode“ an Cucumber delegiert.

*//features/step\_definitions/login\_steps.js*

*var seleniumWebdriver = require('selenium-Webdriver');*

*var {defineSupportCode} = require('cucumber');*

Im nächsten Abschnitt des Codes wird die Methode „defineSupportCode“ aufgerufen mit den Funktionen „Given“, „When“, und „Then“. Im ersten Abschnitt der „Given“-Funktion wird dann die Website der DevOps Backbone Initiative durch einen “get”-Aufruf geöffnet.

*defineSupportCode(function({Given, When, Then}) {*

*Given('As a IBM Employee I am on the Website "DevOps Backbone Initiative"', function(){*

*return this.driver.get("https://devopscoc.w3ibm.mybluemix.net/");*

*});*

Im folgenden Abschnitt wird die User ID automatisch in das Feld namens “username” eingetragen. Das selbe wird danach mit dem Passwort durchgeführt. Außerdem wird der Login durch das Anklicken des Buttons „Sign In“ durchgeführt.

*When('I fill in my IBM ID', function () {*

*return this.driver.findElement({name: "username"}).then(function(element) {*

*return element.sendKeys("mmustermann@de.ibm.com");*

*});*

*});*

*When('I fill in my password', function () {*

*return this.driver.findElement({name: "password"}).then(function(element) {*

*return element.sendKeys("password");*

*});*

*});*

*When('I click on “Sign In”', function () {*

*return this.driver.findElement({id: "btn\_signin"}).then(function(element) {*

*return element.click();*

*});*

*});*

Als Resultat soll dann das Profil angezeigt werden, welches über die „Then“-Funktion geprüft wird. Es wird überprüft, ob der Begriff der zuvor in Anführungszeichen im Feature genannt wurde im Titel der Website angezeigt wird.

*Then('I should see my Profile on {arg1:stringInDoubleQuotes}', function (arg1) {*

*var condition = seleniumWebdriver.until.titleContains(arg1);*

*return this.driver.wait(condition.fn, 20000);*

*});*

Da auch das Szenario existiert, dass ein Benutzer ein falsches Passwort eingibt, setzt die folgende Methode nach dem Schritt der Eingabe der User ID an. Es wird ein falsches Passwort automatisch eingegeben. Danach findet auch der Klick auf den Button zum Einloggen statt.

*When('I fill in a wrong password', function () {*

*return this.driver.findElement({name: "password"}).then(function(element) {*

*return element.sendKeys("wrongpassword");*

*});*

*});*

Es soll dann auf der Login-Seite eine Fehlermeldung ausgegeben werden, was über eine Funktion geprüft wird, welche den Titel der Website überprüft, wenn der Benutzer nämlich vom Login nicht weitergeleitet wird, ist der Login fehlgeschlagen.

*Then('I should see an Error Message', function () {*

*var condition = seleniumWebdriver.until.titleContains("IBM W3 ID");*

*return this.driver.wait(condition.fn, 20000);*

*});*

*});*

Damit ist die Step Definiton für den Login Prozess abgescholossen. Sollte weitere Features überprüft werden, so ist es immer notwendig, dass ein Login als „Given“-Step eingefügt wird. Somit enthalten alle weiteren Step Definitions den Prozess bis zu der Ansicht des eigenen Profils.[[98]](#footnote-98)

## Automatisierungsprozesse

Der Support Code befindet sich dabei in der Datei „world.js“ im Unterordner „Support“ des Ordners „features“. Bevor die Tests gestartet werden können, muss festgelegt werden, in welchem Browser diese durchgeführt werden sollen. Der dazu gehörige Treiber muss angesprochen werden, damit die Durchführung stattfinden kann. Damit Cucumber die Tests automatisch durchführen kann, muss der Selenium Treiber gestartet werden. Danach wird im Programmcode festgelegt, dass durch Selenium der jeweilige Browser geöffnet wird. In diesem Beispiel wird der Browser Google Chrome genutzt. Außerdem muss die Standardlaufzeit von Cucumber von fünf Sekunden auf 20 Sekunden erhöht werden, da die Website vollständig geladen sein muss, die Überprüfung der Szenarien sinnvoll durchgeführt werden kann.

*//features/support/world.js*

*require('chromedriver')*

*var seleniumWebdriver = require('selenium-webdriver');*

*var {defineSupportCode} = require('cucumber');*

*function CustomWorld() {*

*this.driver = new seleniumWebdriver.Builder()*

*.forBrowser('chrome')*

*.build();*

*}*

*defineSupportCode(function({setWorldConstructor}){*

*setWorldConstructor(CustomWorld)*

*})*

*defineSupportCode(function({setDefaultTimeout}) {*

*setDefaultTimeout(20\*1000);*

*});*

Alle Schritte zur Automatisierung sind dabei unabhängig von den Step Definitions zu programmieren. Abgelegt werden diese in einem Ordner der mit „Support“ bezeichnet wird.

## Ausführung von Cucumber

Das Starten des Tools Cucumber ist über die Kommandozeile möglich. Nun werden automatisch die Dateien, in denen die Features und Szenarien gespeichert sind, mit der Endung „.feature“ aufgerufen. Das Tool erstellt das Mapping zu den Step Definitions, welche als JavaScript Dateien gespeichert sind. Durch das ausführen des Support Codes wird der Test gestartet.

In der folgenden Abbildung ist der erfolgreich verlaufene Test dargestellt.

Es sind nochmals alle Szenarien und Steps aufgelistet. Der Erfolg lässt sich an dem Haken zu Beginn der Zeilen und der grünen Markierung erkennen.

Des Weiteren wird angezeigt, wie viele Szenarien mit wie vielen Steps insgesamt durchführt wurden und die gesamte Durchlaufzeit.

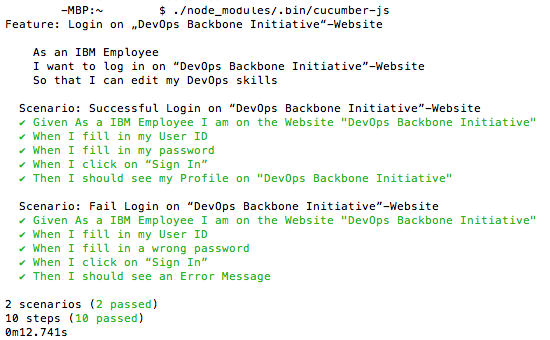


Abbildung : Das erfolgreiche Testresultat[[99]](#footnote-99)

## Bewertung der Durchführbarkeit

Die Erstellung und Durchführung der Tests in Cucumber.js stellt sich als sehr einfach dar. Nach Beginn der Ausformulierung der Features und Szenarien zeigt das Tool nach dem Starten an welches Feature und welche dazugehörigen Szenarien formuliert wurden, dies wird in Abbildung 14 dargestellt. Die Fragezeichen in den Zeilen der einzelnen Informationen geben Aufschluss darüber, dass das Tool hierzu noch keine Step Definitions findet.

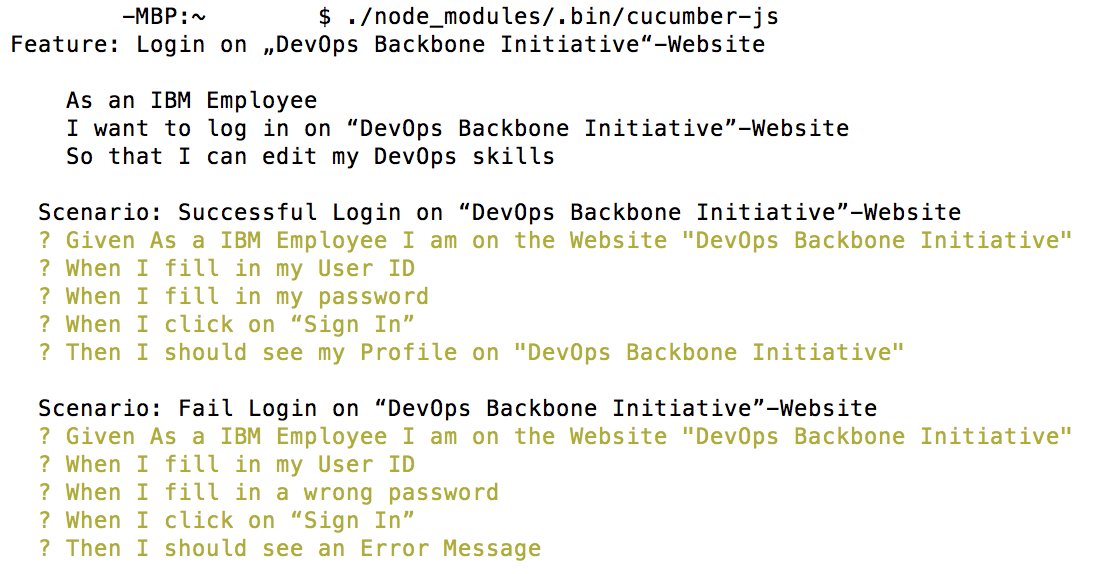


Abbildung : Die Übersicht der fehlenden Steps in Cucumber[[100]](#footnote-100)

Um die Step Definitions zu programmieren gibt das Tool Hilfestellung, indem Cucumber anzeigtdarüber, Funktionen benötigt werden. Beispielhaft ist die in der nächsten Abbildung 15 für den Step der „Given“-Information formuliert. Anhand dieser Vorgaben ist die Umsetzung mittels des Vorgehen des Red-Green-Refactor Cycles[[101]](#footnote-101) sehr einfach möglich.

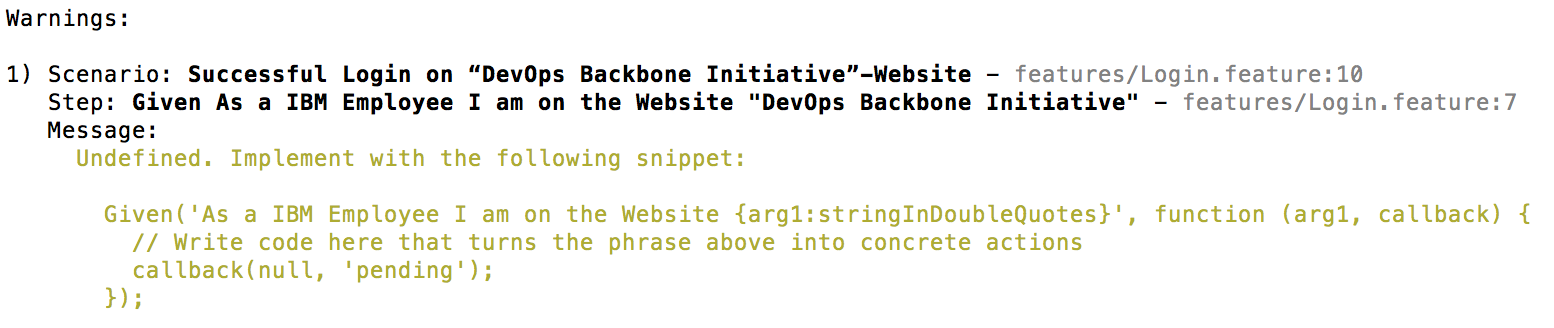


Abbildung : Die Ansicht eines fehlenden Steps[[102]](#footnote-102)

Die Erwartungen der Stakeholder werden durch die Feature eindeutig dargelegt. Jegliche Missverständnisse sind so nahe zu unmöglich. Damit wäre ein großes Problem der Softwareentwicklung gelöst. Denn oft verläuft schon die eindeutige Definition der Anforderung im Requirements Engineering problematisch und der Kunde erhält nicht die Software mit den benötigten Funktionen, welche eigentlich in Auftrag gegeben wurde. Cucumber bietet durch die natürliche Sprache die Möglichkeit der Kommunikation auf derselben Ebene.

# Experteninterviews

## Gesprächsverlauf

## Vorherrschende Meinung

Als Experten wurden Spezialisten der Thematik Testen von IBM befragt.

* Sinnvoll die Durchführung solcher Tests?
* An welcher Stelle sollte dies genauer durchdacht werden?

# Fazit und Ausblick

* Quantitative Forschung wäre noch zu machen um heraus zu finden ob dieser Einsatz auch bei größeren Projekten lohnenswert ist
* Akzepttanztests werden im Rahmen von der DevOps Praktik Continuous Testing automatisch durchgeführt. Die Abnahmekriterien der Kunden werden hinterlegt und überprüft. Wie im Kapitel 3 gezeigt wird lassen sich diese Abnahmekriterien der Kunden mittels Gherkin einfach definieren. Auf Grund dieser User-Stories können dann die Szenarien definiert werden. Durch den Einsatz vom Supportcode findet eine vollständige Automatisierung des Testdurchlaufs. Somit werden die wesentlichen Kennzeichen von Continuous Testing im Bereich des Akzepttests durch Cucumber abgebildet. 🡪 kein Ersatz für Exploratives Testen
* *(Beide Ansätze verfolgen somit fehlerfreie Software, die durch Austausch von allen Beteiligten entsteht. Durch diese gemeinsame Zielstellung ist ein Einklang der Ansätze zu bestätigen.--> Für das Fazit)*
* automatisierte Build Prozesse möglich

# Anhang

Anhang I: Prinzipien von XP

|  |  |
| --- | --- |
| Prinzip | Kurzbeschreibung |
| Menschlichkeit | Es wird auf das Wohlbefinden des gesamten Teams, aber auch des einzelnen geachtet, da Software von Menschen für Menschen entwickelt wird. |
| Wirtschaftlichkeit | Aus einem Projekt muss immer ein Mehrwert generiert werden, dieser kann monetärer Art sein aber auch aus Erfahrung bestehen. |
| Gegenseitiger Vorteil | Es soll für Alle beteiligten eine Win-Win Situation entstehen. Dies inkludiert, dass alle Beteiligten nicht auf ihren persönlichen Vorteil arbeiten, sondern auch zukunftsorientiert und teamorientiert handeln. |
| Selbstähnlichkeit | Dieses Prinzip zielt auf die Widerverwendung ab. Es wird sich an diesem Punkt allerdings nicht auf den Programmcode beschränkt, sondern bezieht sich auf das gesamte Projekt. |
| Verbesserung | Alle Projektmitglieder sollten immer ihre bestmögliche Leistung erbringen. Dennoch soll weiterhin Raum für Verbesserung geschaffen werden. |
| Mannigfaltigkeit | Das Team sollte aus unterschiedlichen Charakteren bestehen. So können neue Perspektiven entstehen und durch Konflikte innovative Lösungen entwickelt werden. |
| Reflektion | Die Beteiligten sollen sich regelmäßig Gedanken über aktuell und zukünftig geleistete Arbeit machen. Umgesetzt wird dieses Prinzip durch spezielle Teammeetings. |
| Fluss | Fluss in der Entwicklung entsteht durch stetige Auslieferung der Software und parallele Durchführung von Entwicklungstätigkeiten. |
| Gelegenheit | Das Team soll lernen durch Probleme Gelegenheiten zu nutzen Neues zu lernen. |
| Redundanz | Die Dopplung von Kompetenzen ist mit diesem Prinzip abgedeckt, so kann sich bei der Paarprogrammierung auf zwei Programmierer mit derselben Fähigkeit verlassen werden. |
| Fehlschläge | Der Lerneffekt aus einem Fehlschlag soll erkannt und genutzt werden. Somit sollten nicht vermeidbare Fehlschläge akzeptiert werden. |
| Qualität | Durch Qualität kann weniger Aufwand in der Zukunft entstehen, da weniger Nachbesserungen notwendig sind. |
| Babyschritte | Das Risiko aber auch die Komplexität lassen sich durch kleine Schritte reduzieren. Fehlschläge fallen nicht so stark ins Gewicht und das Risiko des Scheiterns wird gemindert. |
| Akzeptieren von Verantwortlichkeiten | Die eigene Verantwortung muss von innen heraus befürwortet werden, damit sie akzeptiert wird. |

Anhang II: Primärpraktiken von XP

|  |  |
| --- | --- |
| Primärpraktiken | Erläuterung |
| Räumlich zusammen sitzen | Die Kommunikation wird durch die Nähe der Beteiligten unterstützt und so werden Auseinandersetzungen vermieden. |
| Komplettes Team | Es sollten im Projektteam alle erforderlichen Qualifikationen vertreten sein, so dass keine externe Hilfe benötigt wird. |
| Informative Arbeitsumgebung | Der Arbeitsraum des Teams sollte mit Ausdrucken und Flipcharts von Entwürfen oder offenen Aufgaben gestaltet sein. |
| Energiegeladene Arbeit | Jedes Teammitglied arbeitet engagiert und zeigt vollen Einsatz im Projekt. |
| Programmieren in Paaren | Durch die Arbeit von zwei Entwicklern an einem Computer entsteht eine ständige Kontrolle und Fehler oder Unklarheiten können meist ausgeräumt werden. |
| Stories | Die Anforderungen werden in Form von Stories erhoben. Es ist nicht festgelegt, ob der Kunde oder die Entwickler diese entwerfen. |
| Wochenzyklus | Die Iterationen werden mit einer Laufzeit von einer Woche angesetzt. |
| Quartalszyklus | Innerhalb von drei Monaten werden Release fertiggestellt. |
| Freiraum | Freiräume sind für Entwickler die Phasen, in denen sie sich weiterbilden sollen. Dies ist unbedingt notwendig, da die Technologie zu schnelllebig ist. |
| Zehn-Minuten-Build | Bei diesem Prinzip wird vorgegeben, dass es maximal zehn Minuten dauern darf, bis ein Build des Projekts erzeugt ist. |
| Continuous Integration | Die Änderungen der Entwickler werden mehrmals täglich integriert. |
| Test-Driven Development | Der Testcode wird geschrieben, bevor der Programmcode der Software geschrieben wird. Nach jedem Programmierschritt werden dann die Tests gestartet, umso regelmäßige Rückmeldungen zu erhalten. |
| Inkrementeller Entwurf | Der Entwurf der Software wird schrittweise zu den Anforderungen erstellt. Es werden so immer nur die nächsten konkreten Anforderungen benannt. |

Anhang III: Folgepraktiken von XP

|  |  |
| --- | --- |
| Folgepraktiken | Beschreibung |
| Echte Kundenbeteiligung | Ein Mitarbeiter des Kunden sollte mit in das Team integriert werden, sobald das Team Storys bearbeitet oder inkrementelle Entwürfe erstellt. |
| Inkrementelle Ausbreitung | Altsysteme sollten möglichst schnell durch neue Release abgelöst werden, dies ist allerdings nur durch eine sinnvolle Planung der Release möglich. So können Altsystem und die neu Implementierten Funktionen parallel eingesetzt werden. |
| Teamkontinuität | Die Projektteammitglieder sollten ihre gesamte Arbeitszeit in einem Projekt eingeplant sein und nicht parallel in mehreren Projekten arbeiten, da so Kommunikationsprobleme auftreten können. |
| Schrumpfende Teams | Die Produktivität der einzelnen Entwickler soll soweit erhöht werden, dass einzelne Teammitglieder in andere Projekte transferieren können und so ihr Wissen austauschen können. |
| Ursachenanalyse | Ein Problem wird systematisch analysiert um die Ursache zu identifizieren und es so eliminieren zu können. Diese Analyse kann über unterschiedliche Fragestellungen durchgeführt werden. |
| Gemeinsamer Quelltext | Jeder Entwickler kann den gesamten Programmcode verändern. Wenn das Team nicht gemeinsam arbeitet entstehen dann oft Probleme im Programmcode, welche über Architekturanalysen und Metriken ermittelt werden können. |
| Quelltext und Tests | Es werden lediglich Quelltext und Tests als Artefakte des Projekts erstellt. Eine weitere Dokumentation ist nicht erlaubt, so soll der Quelltext für sich sprechen. |
| Eine Quelltextbasis | Es wird nur auf Basis eines Quelltextes gearbeitet, Branchs sind nur kurzzeitig erlaubt. Wenn zu vermuten ist, dass die Qualität des Programmcodes nicht angemessen ist, muss auf diese Praktik verzichtet werden, denn ansonsten können aufgrund von Weiterentwicklungen fehlerhafte Elemente entstehen. |
| Tägliches Deployment | Es wird täglich ein Deployment durchgeführt. Diese Praktik ist nur zu empfehlen, wenn die Anzahl der Fehler, die der Anwender entdeckt gering ist. |
| Vertrag mit verhandelbarem Umfang | Es wird kein Festpreisvertrag ausgehandelt. So werden weder Qualität, Leistungsumfang, Termine oder der Preis zu Beginn des Projekts ausgehandelt. Durch Festpreisverträge werden Lernprozesse im Projektverlauf verhindert, da ein oft strenger Terminplan eingehalten werden muss. |
| Bezahlung je Benutzung | Die Bezahlung des Systems verläuft nicht nach dem Release, sondern nach Benutzung der Funktionen. So wird erreicht, dass die Softwareentwickler das System so verbessern, dass die profitablen Funktionen noch häufiger verwendet werden. |

Anhang IV: Weitere User-Stories und Szenarien

*//features/edit.feature*

*Feature: Edit Skill Level*

*As an IBM Employee*

*I want to edit my DevOps skills*

*So that I can change my actual skill level*

*Scenario: Editing of Skill Level*

*Given I am logged into the DevOps Backbone Initiative-Website*

*When I click on "Edit"-Button*

*Then I should edit my Skills*

*//features/step\_definitions/edit.js*

*var seleniumWebdriver = require('selenium-Webdriver');*

*var {defineSupportCode} = require('cucumber');*

*defineSupportCode(function({Given, When, Then}) {*

*Given('I am logged into the DevOps Backbone Initiative-Website', function (){*

*this.driver.get("https://devopscoc.w3ibm.mybluemix.net/");*

*this.driver.findElement({name: "username"}).sendKeys("mmustermann@de.ibm.com");*

*this.driver.findElement({name: "password"}).sendKeys("password");*

*this.driver.findElement({id: "btn\_signin"}).click();*

*var condition = seleniumWebdriver.until.titleContains("DevOps Backbone Initiative");*

*return this.driver.wait(condition.fn, 10000);*

*});*

*When('I click on "Edit"-Button', function(){*

*return this.driver.findElement({id:"edittop"}).then(function(element){*

*return element.click();*

*});*

*});*

*Then('I should edit my Skills', function (){*

*return this.driver.findElement({id:"savetop"});*

*});*

*});*

//features/delete.feature

*Feature: Delete Skills*

*As an IBM Employee*

*I want to delete my DevOps Skills*

*So that my skills are deleted from database*

*Scenario: Deleting of skills*

*Given I am logged into the DevOps Backbone Initiative-Website*

*When I click the "Delete"-Button*

*Then I should see the Confirmation*

*//feature/step\_definitions/delete\_steps.js*

*var seleniumWebdriver = require('selenium-Webdriver');*

*var {defineSupportCode} = require('cucumber');*

*defineSupportCode(function({Given, When, And, Then}) {*

*Given('I am logged into the DevOps Backbone Initiative-Website', function (){*

*this.driver.get("https://devopscoc.w3ibm.mybluemix.net/");*

*this.driver.findElement({name: "username"}).sendKeys("mmustermann@de.ibm.com");*

*this.driver.findElement({name: "password"}).sendKeys("password");*

*this.driver.findElement({id: "btn\_signin"}).click();*

*var condition = seleniumWebdriver.until.titleContains("DevOps Backbone Initiative");*

*return this.driver.wait(condition.fn, 10000);*

*});*

*When('I click the "Delete"-Button', function (){*

*return this.driver.findElement({id:"delete"}).then(function(element){*

*return element.click();*

*});*

*});*

*Then('I should see the Confirmation', function(){*

*var condition = this.driver.until.titleContains("Delete Confirmation Page");*

*return this.driver.wait(condition.fn, 10000);*

*});*

*});*

*//features/logout.feature*

*Feature: Log Out*

*As an IBM Employee*

*I want to log out*

*So that I should safely leave the Website of “DevOps Backbone Initiative”*

*Scenario: Log Out*

*Given I am logged into the DevOps Backbone Initiative-Website*

*When I click on “Logout”-Button*

*Then I should see the Confirmation*

*//feature/step\_definitions/logout.js*

*var seleniumWebdriver = require('selenium-Webdriver');*

*var {defineSupportCode} = require('cucumber');*

*defineSupportCode(function({Given, When, Then}) {*

*Given('I am logged into the DevOps Backbone Initiative-Website', function (){*

*this.driver.get("https://devopscoc.w3ibm.mybluemix.net/");*

*this.driver.findElement({name: "username"}).sendKeys("mmuus@de.ibm.com");*

*this.driver.findElement({name: "password"}).sendKeys("Winter-02-17");*

*this.driver.findElement({id: "btn\_signin"}).click();*

*var condition = seleniumWebdriver.until.titleContains("DevOps Backbone Initiative");*

*return this.driver.wait(condition.fn, 10000);*

*});*

*When('I click on “Logout”-Button', function(){*

*return this.driver.findElement({id:"logout"}).then(function(element) {*

*return element.click();});*

*});*

*Then('I should see the Confirmation', function(){*

*var condition = seleniumWebdriver.until.titleContains("IBM W3 ID");*

*return this.driver.wait(condition.fn, 10000);*

*});*

*});*

# Literaturverzeichnis

**Bass et. al. (2015)**

Bass, L., Weber, I., Zhu L.: DevOps: A software architect's perspective. 2015. Old Tappan, New Jersey: Addison-Wesley.

**Bell und Beer (2014)**

Bell, P., Beer B.: Introducing GitHub: A Non-Technical Guide. 1. Aufl., 2014. Sebastopol: O’Reilly Media.

**Beck (2000)**

Beck, K.: Extreme Programming: Die revolutionäre Methode für Softwareentwicklung in kleinen Teams. 2000. München: Addison-Wesley Verlag.

**Beck (2003)**

Beck, K.: Test-Driven Development by Example. Westford: Pearson Education.

**Beck und Andres (2005)**

Back, K., Andres, C.: Extreme Programming Explained: Embrace Change. 2. Aufl., 2005. Crawfordsville: Pearsoned.

**Boehm (1981)**

Boehm,B. W.: Software Engineering Economics. 1. Aufl., 1981. Englewood Cliffs: Prentice-Hall Inc..

**Brandt- Pook und Kollmeier (2008)**

Brandt-Pook, H., Kollmeier R.: Softwareentwicklung kompakt und Verständlich: Wie Softwaresysteme entstehen. 2. Aufl., 2008. Wiesbaden: Springer Vieweg.

**Davis und Daniels (2015)**

Davis, J., Daniels, K.: Effective DevOps: Building a culture of collaboration, affinity, and tooling at scale. 1. Aufl., 2015. Sebastopol: O'Reilly Media.

**Emrich und Marit (2016)**

Emrich, M., Marit, C.: JavaScript: Aller Anfang ist leicht. 1. Aufl., 2016. Nürnberg: Webmasters Press.

**Gloger und Margetich (2014)**

Gloger, B., Margetich, J.: Das Scrum-Prinzip: Agile Organisationen aufbauen und gestalten. Stuttgart: Schäffer-Poeschek Verlag.

**Hüttermann (2012)**

Hüttermann, M.: DevOps for developers: Integrate development and operations, the agile way. 2012. New York: Apress, Springer.

**Koch (2011)**

JavaScript: Einführung, Programmierung und Referenz. 6. Aufl., 201. Heidelberg: dpunkt Verlag.

**Rubin (2014)**

Rubin, S.,K.: Essential Scrum: Umfassendes Scrum-Wissen aus der Praxis. 1. Aufl., 2014. Heidelberg: Hüthig JEhle Rehm GmbH.

**Springer (2013)**

Springer, S.: Node.js: Das umfassende Handbuch.1. Aufl., 2013, Bonn: Galileo Press.

**Swartout (2014)**

Swartout, P.: Continuous Delivery and DevOps: A Quickstart Guide. 2014. Birmingham: Packt Publishing.

**Schwaber und Beedle (2003)**

Schwaber, K., Beedle, M.: Agile Software Developtment with Scrum. 2003. Upper Saddle River: Pearson Education.

**Verona (2016)**

Verona, J.: Practical DevOps: Harness the power of DevOps to boost your skill set and make your IT organization perform better. 1. Aufl., 2016. Birmingham: Packt Publishing.

**Wolf und Bleek (2011)**

Wolf, H. und Bleek, W.-G.: Agile Softwareentwicklung: Werte, Konzepte und Methoden. 2. Aufl., 2011. Heidelberg: dpunkt-Verl.

**Wolff (2015)**

Wolff, E.: Continuous delivery: Der pragmatische Einstieg. 2. Aufl., 2015. Heidelberg: Dpunkt.

**Wynne und Hellesøy (2012)**

Wynee, M., Hellesøy, A.: The Cucumber Book: Behaviour-Driven Development for Testers and Developers. 2. Aufl. Pragmatic Programmers LLC.

**Dees et. al. (2013)**

Dees, I., Wynee, M., Hellesøy, A.: Cucumber Recipes: Automate Anything with BBD Tools and Techiques. 1. Aufl. Pragmatic Programmers LLC.

**Wrobel (2015)**

Wrobel, G.: JavaScript Tools: Besserer Code durch eine professionelle Programmierumgebung. 2015. Open Source Press.

# Internetquellen

**agilemanifesto.org (2011)**

agilemanifesto.org: Manifesto for Agile Software Development. Zugriff am 02.10.2016,

von: http://agilemanifesto.org/iso/de/manifesto.html. Gespeichert unter: CD\Internetquellen\agilemanifesto.org (2011).

**Anforderungsmanagement**

Anforderungsmanagement: Funktionale, nicht funktionale Anforderungen. Zugriff am 09.01.2017,

von: http://www.anforderungsmanagement.ch/in\_depth\_vertiefung/funktionale\_nicht\_  
funktionale\_anforderungen/. Gespeichert unter: CD\Internetquellen\  
Anforderungsmanagement.

**Caum (2013)**

Caum, C.: Continuous Delivery Vs. Continuous Deployment: What's the Diff. Artikel vom 30.08.2013, Zugriff am 29.10.2016,

von: https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff. Gespeichert unter: CD\Internetquellen\Caum (2013).

**Dawson (2016)**

Dawson, B.: Was ist eigentlich DevOps?. Artikel vom 22.08.2016, Zugriff am 15.10.2016,   
von: https://jaxenter.de/was-ist-devops-45376. Gespeichert unter: CD\Internetquellen\Dawson (2016).

**Edwards (2010)**

Edwards, D.: What is DevOps?. Artikel vom 23.02.2010, Zugriff am 17.10.2016,

von: http://dev2ops.org/2010/02/what-is-devops/. Gespeichert unter: CD\Internetquellen\Edwards(2010).

**Erwin (2013)**

Erwin, T., Marlinghaus, S.: Continuous Monitoring: Risiken erkennen, Transparenz schaffen. Artikel vom 15.10.2013, Zugriff am: 07.11.2016

von: http://www.computerwoche.de/a/risiken-erkennen-transparenz-schaffen,2547630. Gespeichert unter: CD\Internetquellen\Erwin(2013).

**Fowler (2013)**

Fowler, M.: Continuous Delivery. Artikel vom 30.05.2013, Zugriff am 16.10.2016,

von: http://martinfowler.com/bliki/ContinuousDelivery.html. Gespeichert unter: CD\Internetquellen\Fowler(2013).

**Gründerszene**

Gründerszene: Agile Softwareentwicklung. Zugriff am 03.10.2016 ,

von: http://www.gruenderszene.de/lexikon/begriffe/agile-softwareentwicklung. Gespeichert unter: CD\Internetquellen\Gruenderszene.

**Hellesøy (2014)**

Hellesøy, A.: The world’s most misunderstood collaboration tool. Artikel vom 03.03.2014, Zugriff am 13.01.2017.

von: https://cucumber.io/blog/2014/03/03/the-worlds-most-misunterstood-collaboration-tool. Gespeichert unter: CD\Internetquellen\Hellesoy(2014).

**IBM Deutschland**

IBM Deutschland: IBM in Deutschland - Über uns. Zugriff am 02.10.2016,

von: http://www-05.ibm.com/de/ibm/unternehmen/index.html. Gespeichert unter: CD\Internetquellen\IBM\_Deutschland.

**IBM GBS**

IBM GBS: Consulting. Zugriff am 01.10.2016,

von: http://www-935.ibm.com/services/de/gbs/consulting/. Gespeichert unter: CD\Internetquellen\IBM\_GBS.

**IBM Unternehmensstruktur**

IBM: Unternehmensstruktur. Zugriff am 02.10.2016,

von: http://www.ibm.com/de/ibm/unternehmen/struktur/. Gespeichert unter: CD\Internetquellen\IBM\_Unternehmensstruktur.

**Macvittie (2016)**

Macvittie, L.: Application Services and the CD Pipeline, Artikel vom 25.02.2016, Zugriff am 20.11.2016,

von: https://f5.com/about-us/blog/articles/application-services-and-the-cd-pipeline. Gespeichert unter: CD\Internetquellen\Macvittie(2016).

**Martin (2014)**

Martin, B.: DevOps, Agile Development, Continuous Delivery und ITIL: Was gibt es für Zusammenhänge und wie man sie nutzen kann, Artikel vom 09.10.2014, Zugriff am 28.11.2016,

von: http://blog.itil.org/2014/10/itil/devops-agile-development-continuous-delivery-und-itil-was-gibt-es-fuer-zusammenhaenge-und-wie-man-sie-nutzen-kann/. Gespeichert unter: CD\Internetquellen\Martin(2014).

**Neumann (2011)**

Neumann, A.: 10 Jahres Agiles Manifest zur Geburt agiles Softwareentwicklung. Artikel vom 12.02.2011, Zugriff am 05.10.2016,

von: http://www.heise.de/developer/meldung/10-Jahre-Agiles-Manifest-zur-Geburt-agiler-Softwareentwicklung-1188299.html. Gespeichert unter: CD\Internetquellen\Neumann(2011).

**North (2006)**

North, D.: Introducing BDD. Artikel vom 03.2006. Zugriff am 13.01.2017.

von: https://dannorth.net/2006/10/20/article-introducing-behaviour-driven-development/.Gespeichert unter: CD\Internetquellen\North(2006).

🡪 Ursprünglich für Better Software Magazine

**Onepage**

Onepage: DevOps. Zugriff am 17.10.2016,

von: https://de.onpage.org/wiki/DevOps. Gespeichert unter: CD\Internetquellen\Onepage.

**Peschlow (2012)**

Peschlow, P.: Die DevOps-Bewegung. Artikel aus Javamagazin, vom 01.12.2012. Zugriff am 10.10.2016,

von: https://www.codecentric.de/kompetenzen/publikationen/die-devops-bewegung. Gespeichert unter: CD\Internetquellen\Peschlow(2012).

**Prowareness**

Prowareness: Continuous Delivery: Das Agility Ladder Framework. Zugriff am 26.01.2017,

von: https://www.prowareness.de/continuous-delivery-it/. Gespeichert unter: CD\Internetquellen\Prowareness

**Royce (1970)**

Royce, W.: Managing the Development of Large Software Systems. Artikel aus Proceedings, IEEE WESCON, vom 08.1970, Zugriff am 02.10.2016,

von: http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf. Gespeichert unter: CD\Internetquellen\Royce(1970).

**Schwaber und Sutherland (2016)**

Schwaber, K., Sutherland, J.: The Scrum Guide: Der gültige Leitfaden für Scrum: Die Spielregeln. Von 2016. Zugriff am 13.01.2017,

von: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf. Gespeichert unter: CD\Internetquellen\Schwaber\_Sutherland(2016).

**Tee**

Tee, J.: Coontinuous Development: The glue holding DevOps, TDD and Agile methods together. Zugriff am 13.01.2017.

von: http://www.theserverside.com/feature/Continuous-Development-The-glue-holding-DevOps-TDD-and-Agile-together. Gespeichert unter: CD\Internetquellen\Tee.

**Testing Excellence**

Testing Excellence: Types of Software Testing – Complete List. Zugriff am 28.11.2016,

von: http://www.testingexcellence.com/types-of-software-testing-complete-list/. Gespeichert unter: CD\Internetquellen\Testing\_Excellence.

**Tutorialspoint**

Tutorialspoint: Behavior Driven Development – Quick Guide. Zugriff am 15.01.2017,

von: https://www.tutorialspoint.com/behavior\_driven\_development/index.htm. Gespeichert unter: CD\Internetquellen\Tutorialspoint.

# Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der in den Fußnoten und im Literaturverzeichnis angegebenen Quellen angefertigt habe.

Kiel, den 23. Februar 2017

*──────────────────*

*Mareike Muus*

1. Vgl. IBM Unternehmensstruktur [↑](#footnote-ref-1)
2. Vgl. IBM Deutschland [↑](#footnote-ref-2)
3. Quelle: Eigene Erstellung in Anlehnung an IBM Unternehmensstruktur [↑](#footnote-ref-3)
4. Vgl. IBM GBS [↑](#footnote-ref-4)
5. Vgl. Wolf und Bleek (2011), S. 8 [↑](#footnote-ref-5)
6. Vgl. Boehm (1981), S. 35 [↑](#footnote-ref-6)
7. Vgl. Royce (1970) [↑](#footnote-ref-7)
8. Vgl. Boehm (1981), S. 37 [↑](#footnote-ref-8)
9. Quelle: Eigene Erstellung in Anlehnung am Boehm (1981), S. 36 [↑](#footnote-ref-9)
10. Vgl. XXX [↑](#footnote-ref-10)
11. Vgl. Brandt-Pook und Kollmeier (2008), S. 24 [↑](#footnote-ref-11)
12. Agilesmanifesto.org [↑](#footnote-ref-12)
13. Vgl. Beck (2000), S. 29 ff. [↑](#footnote-ref-13)
14. siehe *Anhang I: Prinzipien von XP* [↑](#footnote-ref-14)
15. Vgl. Beck und Andres (2005), S. 14 [↑](#footnote-ref-15)
16. Vgl. Beck (2000), S. 140 ff. [↑](#footnote-ref-16)
17. siehe *Anhang II: Primärpraktiken von XP* [↑](#footnote-ref-17)
18. siehe *Anhang III*: Folge Praktiken von XP [↑](#footnote-ref-18)
19. Quelle: Eigene Erstellung in Anlehnung an Wolf und Bleek (2011), S. 156 [↑](#footnote-ref-19)
20. Vgl. Beck (2000), S. 89 [↑](#footnote-ref-20)
21. Vgl. Wolf und Bleek (2011), S. 160 [↑](#footnote-ref-21)
22. Vgl. *Anhang II: Primärpraktiken von XP* [↑](#footnote-ref-22)
23. Vgl. Anforderungsmanagement [↑](#footnote-ref-23)
24. Vgl. Tutorialspoint [↑](#footnote-ref-24)
25. Übersetzung: Testgetriebene Entwickelung, häufig wird auch „testgetriebener Entwurf“ genutzt [↑](#footnote-ref-25)
26. Vgl. Wolf und Bleek (2011), S. 99 [↑](#footnote-ref-26)
27. Beck (2003), S. X [↑](#footnote-ref-27)
28. Quelle: Eigene Erstellung in Anlehnung an Wolf und Bleek (2011), S. 99 [↑](#footnote-ref-28)
29. Vgl. Solís und Wang (2011) [↑](#footnote-ref-29)
30. Vgl. Wynne und Hellesøy (2012), S. 3 [↑](#footnote-ref-30)
31. Vgl. Solís und Wang (2011) [↑](#footnote-ref-31)
32. Stakeholder sind alle Personen, die von der unternehmerischen Tätigkeit gegenwärtig oder zukünftig betroffen sind. [↑](#footnote-ref-32)
33. Vgl. North (2006) [↑](#footnote-ref-33)
34. Quelle: Eigene Erstellung in Anlehnung an North (2006) [↑](#footnote-ref-34)
35. Quelle: Eigene Erstellung in Anlehnung an Solís und Wang (2011) [↑](#footnote-ref-35)
36. Vgl. Solís und Wang (2011) [↑](#footnote-ref-36)
37. Quelle: Eigene Erstellung in Anlehnung an North [↑](#footnote-ref-37)
38. Vgl. Hellesøy (2014) [↑](#footnote-ref-38)
39. Vgl. Tee [↑](#footnote-ref-39)
40. Vgl. North (2006) [↑](#footnote-ref-40)
41. Vgl. Peschlow (2012) [↑](#footnote-ref-41)
42. Quelle: Eigene Erstellung in Anlehnung an Hüttermann (2012), S. 20 [↑](#footnote-ref-42)
43. Vgl. Bass et al. (2015), S. 27 [↑](#footnote-ref-43)
44. Vgl. Verona (2016), S. 1 (aus dem Englischen übersetzt) [↑](#footnote-ref-44)
45. Vgl. Minich (2013) (aus dem Englischen übersetzt) [↑](#footnote-ref-45)
46. Vgl. Hüttermann (2012), S. 7 f. [↑](#footnote-ref-46)
47. Vgl. Dawson (2016) [↑](#footnote-ref-47)
48. Vgl. Wolff (2016), S. 238 [↑](#footnote-ref-48)
49. Vgl. Kapitel 3.3 Der Konflikt innerhalb der IT-Organisation [↑](#footnote-ref-49)
50. Vgl. Swartout (2012), S. 22 [↑](#footnote-ref-50)
51. Vgl. Swartout (2012), S. 54 f [↑](#footnote-ref-51)
52. Vgl. Davis und Daniels (2016), S. 42 ff [↑](#footnote-ref-52)
53. Vgl. Verona (2016), S. 39 ff. [↑](#footnote-ref-53)
54. Vgl. Bass et. al. (2015), S. 11 [↑](#footnote-ref-54)
55. Vgl. Onepage [↑](#footnote-ref-55)
56. Vgl. Aiello und Sachs (2016), S. 214 [↑](#footnote-ref-56)
57. Vgl. Aiello und Sachs (2016), S. 222 [↑](#footnote-ref-57)
58. Vgl. Onepage [↑](#footnote-ref-58)
59. Vgl. Kapitel 3.4.1 Kernelemente von DevOps [↑](#footnote-ref-59)
60. Quelle: Eigene Erstellung in Anlehnung an Martin (2014) [↑](#footnote-ref-60)
61. Übersetzung: kontinuierliche Integration [↑](#footnote-ref-61)
62. Vgl. Edwards (2010) [↑](#footnote-ref-62)
63. Quelle: Eigene Erstellung in Anlehnung an Verona (2016), S.14 [↑](#footnote-ref-63)
64. Ein Zweig des Programmcodes, der vom übrigen Programmcode gelöst wurde, um einen einzelnen Arbeitszweig zu erhalten. Der Hauptzweig des Programms wird Master-Branch genannt. [↑](#footnote-ref-64)
65. Übersetzung: kontinuierliche Auslieferung [↑](#footnote-ref-65)
66. Beschreibt den Versionsstand einer Software. [↑](#footnote-ref-66)
67. Vgl. Wolff (2016), S. 13 [↑](#footnote-ref-67)
68. Quelle: Eigene Erstellung in Anlehnung an Wolff (2016), S. 24 [↑](#footnote-ref-68)
69. Vgl. Kapitel 3.5.1 Continuous Integration (CI) [↑](#footnote-ref-69)
70. Vgl. Kapitel 3.2 Behaviour-Driven Development [↑](#footnote-ref-70)
71. Vgl. Test Exellence [↑](#footnote-ref-71)
72. Vgl. Prowareness [↑](#footnote-ref-72)
73. Übersetzung: kontinuierliche Einführung [↑](#footnote-ref-73)
74. Vgl. Fowler (2013) [↑](#footnote-ref-74)
75. Vgl. Kapitel 3.5.2 Continuous Delivery [↑](#footnote-ref-75)
76. Vgl. Caum (2013) [↑](#footnote-ref-76)
77. Quelle: Eigene Erstellung in Anlehnung an Macvittie (2016) [↑](#footnote-ref-77)
78. Vgl. Wolff (2016), S. 14 ff. [↑](#footnote-ref-78)
79. Vgl. Davis und Daniels (2016), S. 39 [↑](#footnote-ref-79)
80. Übersetzung: kontinuierliches Testen [↑](#footnote-ref-80)
81. Vgl. Kapitel 3.5.2 Conitnuous Delivery (CD) [↑](#footnote-ref-81)
82. Vgl. Kapitel 3.5.3 Continuous Deployment [↑](#footnote-ref-82)
83. XXX [↑](#footnote-ref-83)
84. Vgl. Aiello und Sachs (2016), S. 35 [↑](#footnote-ref-84)
85. Vgl. Testing Excellence [↑](#footnote-ref-85)
86. Übersetzung: kontinuierliche Überprüfung [↑](#footnote-ref-86)
87. Vgl. Bass et. al. (2015), S. 130 ff. [↑](#footnote-ref-87)
88. Vgl. Erwin (2013) [↑](#footnote-ref-88)
89. Quelle: Eigene Erstellung [↑](#footnote-ref-89)
90. Vgl. Bass et. al. (2015), S. 144 [↑](#footnote-ref-90)
91. Cucumber.io [↑](#footnote-ref-91)
92. Wynne und Hellesy (2012), S. 8 [↑](#footnote-ref-92)
93. Quelle: Eigene Erstellung in Anlehnung an Wynne und Hellesøy (2012), S. 8 [↑](#footnote-ref-93)
94. Single Sign-On hat den Vorteil, dass ein Benutzer nach einer einmaligen Authentifizierung jegliche Services nutzen kann. [↑](#footnote-ref-94)
95. Ein Intranet ist ein unternehmensinternes Computernetzwerk. [↑](#footnote-ref-95)
96. Quelle: Eigene Erstellung [↑](#footnote-ref-96)
97. Es wird an dieser Stelle kein Gebrauch davongemacht, dass Gherkin auch auf Deutsch einsetzbar ist, da die Unternehmenssprache der IBM Englisch ist und das Tool international verwendet wird. [↑](#footnote-ref-97)
98. Vgl. *Anhang IV* [↑](#footnote-ref-98)
99. Quelle: Eigene Erstellung [↑](#footnote-ref-99)
100. Quelle: Eigene Erstellung [↑](#footnote-ref-100)
101. Vgl. Kapitel 3.2 Behaviour-Driven Development [↑](#footnote-ref-101)
102. Quelle: Eigene Erstellung [↑](#footnote-ref-102)