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?

Autorin:	Mareike Muus
Matrikelnummer:	924027
Adresse:	Veilchenweg 37 23626 Ratekau
E-Mail:	mareike.muus@student.fh-kiel.de
Telefon:	017664919310

Betreuende Hochschullehrerin: Prof. Dr. Doris Weßels

Abgabedatum: 23.02.2017

Inhaltsverzeichnis

Inhaltsverzeichnis ..... **I**

Gliederung ..... **II**

Abkürzungsverzeichnis ..... **V**

Abbildungsverzeichnis ..... **VI**

Anhangsverzeichnis ..... **VII**

Programmcodeelementverzeichnis ..... **VIII**

Tabellenverzeichnis ..... **IX**

1 Einleitung **1**

2 Vorstellung des Unternehmens **4**

3 Einführung in die theoretischen Grundlagen **7**

4 Praktische Umsetzung von Behaviour-Driven Development **33**

5 Experteninterviews **46**

6 Zusammenfassung und Ergebnisse **51**

7 Anhang **54**

Literaturverzeichnis **74**

Internetquellen **76**

Eidesstattliche Versicherung **80**

Gliederung

Inhaltsverzeichnis ..... **I**

Gliederung ..... **II**

Abkürzungsverzeichnis ..... **V**

Abbildungsverzeichnis ..... **VI**

Anhangsverzeichnis ..... **VII**

Programmcodeelementverzeichnis ..... **VIII**

Tabellenverzeichnis ..... **IX**

1 Einleitung **1**

1.1 Motivation und Problemstellung *1*

1.2 Ziel der Arbeit und Forschungsfragen *1*

1.3 Aufbau der Arbeit *2*

1.4 Abgrenzung des Themengebiets *3*

2 Vorstellung des Unternehmens **4**

2.1 IBM – International Business Machines *4*

2.2 IBM – Deutschland *4*

2.3 IBM GBS – Global Business Services *5*

2.4 IBM – DevOps Center of Competence Europe *5*

3 Einführung in die theoretischen Grundlagen **7**

3.1 Die Softwareentwicklung *7*

3.1.1 Klassische Softwareentwicklung 7

3.1.2 Agile Softwareentwicklung 9

3.2 Behaviour-Driven Development *11*

3.2.1 Bedeutung und Herkunft 12

3.2.2 Vorgehensweise bei BDD 14

3.2.3 Vor- und Nachteile 17

3.3 Der Konflikt innerhalb der IT-Organisation *17*

3.4 DevOps im Unternehmen *19*

3.4.1 Defintion von DevOps ... 19
3.4.2 Kernelemente von DevOps ... 21
3.4.3 Agilität von DevOps ... 23
3.5 Die DevOps Praktiken ... 23
3.5.1 Continuous Integration ... 24
3.5.2 Continuous Delivery ... 25
3.5.3 Continuous Deployment ... 27
3.5.4 Continuous Testing ... 28
3.5.5 Continuous Monitoring ... 30
3.6 Vor- und Nachteile von DevOps ... 31
3.7 Gegenüberstellung von BDD und DevOps ... 31
4 Praktische Umsetzung von Behaviour-Driven Development ... 33
4.1 Entwicklung von Testfällen mit Cucumber ... 33
4.2 Vorstellung des Projekts ... 35
4.3 Anforderungen an das Tool ... 35
4.4 Selenium ... 37
4.5 Ordnerstruktur der Tests ... 37
4.6 Browser-Steuerung ... 38
4.7 Definition von Features und Szenarien ... 39
4.8 Erläuterung der Step Definition ... 41
4.9 Ausführung von Cucumber ... 44
4.10 Bewertung der Durchführung ... 45
5 Experteninterviews ... 46
5.1 Vorstellung der Gesprächspartner ... 46
5.2 Analyse der Interviews ... 47
5.2.1 Requirements Engineering ... 47
5.2.2 SAFe und BDD ... 48
5.2.3 Rollenverteilung ... 49
5.2.4 DevOps und BDD ... 49
5.3 Meinungsbild der Experten zu BDD ... 50
6 Zusammenfassung und Ergebnisse ... 51

6.1 Beantwortung der Forschungsfragen 51

6.2 Fazit 52

6.3 Ausblick 53

7 Anhang 54

Literaturverzeichnis 74

Internetquellen 76

Eidesstattliche Versicherung 80

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
RDNG	Rational DOORS next Generation
IBM	International Business Machines
IT	Informationstechnik
GBS	Global Business Services
RQM	Rational Quality Manager
RTC	Rational Team Concert
SAFe	Scaled Agile Framework
Ops	IT-Betrieb (abgekürzt von Operations)
TDD	Test-Driven Development
XP	eXtreme Programming

Abbildungsverzeichnis

Abbildung 1: Die Unternehmensstruktur der IBM 5

Abbildung 2: Das Wasserfallmodell nach Boehm 8

Abbildung 3: Die XP-Praktiken im Überblick 10

Abbildung 4: Der XP-Prozess 11

Abbildung 5: Der Red-Green-Refactor Cycle 14

Abbildung 6: Die Isolation innerhalb der IT-Organisation 18

Abbildung 7: Die Einbindung der Einheiten der IT-Organisation in DevOps 20

Abbildung 8: Die Kernelemente von DevOps 22

Abbildung 9: Der Mehrwert von DevOps 22

Abbildung 10: Die Reichweiten der verschiedenen DevOps Praktiken 24

Abbildung 11: Der Continuous Integration-Prozess 25

Abbildung 12: Die CD-Pipeline 26

Abbildung 13: Die Continuous Deployment-Pipeline im Vergleich 27

Abbildung 14: Der CT-Prozess 29

Abbildung 15: Der CM-Prozess 30

Abbildung 16: Die Funktionsweise von Cucumber 34

Abbildung 17: Das Tool im Überblick 36

Abbildung 18: Der Überblick der Ordnerstruktur 37

Abbildung 19: Die Übersicht der fehlenden Schritte in Cucumber 40

Abbildung 20: Die Anzeige einer fehlenden Funktion 41

Abbildung 21: Das erfolgreiche Testresultat 44

Abbildung 22: Der Mehrwert durch die Einführung von BDD 50

Anhangsverzeichnis

Anhang I: Prinzipien von XP 54

Anhang II: Primärpraktiken von XP 57

Anhang III: Folgepraktiken von XP 59

Anhang IV: Weitere User-Stories und Szenarien 62

Anhang V: Interview Leitfaden 67

Programmcodeelementverzeichnis

Programmcodeelement 1: Beispiel für ein Feature und Szenario in Cucumber 34

Programmcodeelement 2: Supportcode 38

Programmcodeelement 3: Feature 39

Programmcodeelement 4: Background und Szenarien 40

Programmcodeelement 5: Aufruf der Treiber 41

Programmcodeelement 6: Given-Schritt 42

Programmcodeelement 7: When-Information mit der Eingabe des richtigen Passworts 42

Programmcodeelement 8: Then-Information für den erfolgreichen Login 43

Programmcodeelement 9: When-Information mit der Eingabe eines falschen Passworts 43

Programmcodeelement 10: Then-Information für den fehlgeschlagenen Login 43

Tabellenverzeichnis

Tabelle 1: Vorstellung der wichtigsten Testarten 12

Tabelle 2: Beispiel für eine User-Story 15

Tabelle 3: Aufbau einer User-Story 15

Tabelle 4: Beispiel für ein Szenario 16

Tabelle 5: Vergleich von BDD und DevOps 32

1 Einleitung

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

1.1 Motivation und Problemstellung

Die heutige Branche der Informationstechnik (IT) ist geprägt durch ihre Schnelllebigkeit. Kunden bestehen auf möglichst kurze Reaktionszeiten des Softwareentwicklungsteams und bei Bedarf auch Flexibilität in Bezug auf Änderungen. 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. Kulturelle Aspekte aber auch die Automatisierung von Prozessen wird im Bereich DevOps außerdem fokussiert. Um eine fehlerfreie Software zu erstellen, 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. Diese Praktik untersucht die Software auf Grundlage ihrer Verhaltensweisen. Es werden Automatisierungsprozesse eingesetzt und die Kommunikation zwischen den einzelnen Parteien bildet den Kern des Vorgehens.

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

1.2 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-Ansatz und DevOps-Praktiken?
2. Ist Behaviour-Driven Development mit Cucumber im Bereich Continuous Testing möglich?
3. Wie beurteilen projekterfahrene Experten Behaviour-Driven Development mit Cucumber?

1.3 Aufbau der Arbeit

Zur Einleitung beschreibt das erste Kapitel die Problemstellung und deren Hintergründe, mit denen 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 findet die Vorstellung des Unternehmens International Business Machines statt. Die Strukturen des Unternehmens werden erläutert und es wird genauer auf das Team eingegangen, das das Projekt für diese Ausarbeitung bearbeitet hat.

Im dritten Kapitel werden die theoretischen Grundlagen erläutert. Die klassische und agile Softwareentwicklung werden anhand von unterschiedlichen Vorgehensmodellen erklärt. 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 vierte Kapitel konzentriert sich auf die praktische Anwendung von Behaviour-Driven Development mit Cucumber. 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 Rahmen des fünften Kapitels wird sich auf Interviews mit praxiserfahrenen IBM Experten bezogen. Die Bereiche, die im Interview fokussiert betrachtet wurden, werden in diesem Kapitel analysiert. Im Anschluss daran erfolgt eine Zusammenfassung der Meinungsbilder der Experten.

Das abschließende sechste Kapitel beschäftigt sich mit der Zusammenfassung der Ausarbeitung. Es findet außerdem eine Beantwortung der Forschungsfragen statt. Im Rahmen eines Fazits wird die Fragestellung des Titels „Erfüllt Cucumber Ansätze beider Praktiken?“ beantwortet. Außerdem wird ein Ausblick über Themengebiete von BDD und Cucumber gegeben, die noch untersucht werden müssen.

1.4 Abgrenzung des Themengebiets

Es findet keine quantitative Untersuchung des Themengebiets, sondern lediglich die qualitative Überprüfung statt. Dies bedeutet, dass der Einsatz von BDD mit dem Tool Cucumber nur anhand eines Beispiels erprobt wird. Außerdem wird festgehalten, dass keine weiteren Analysen zu zusätzlichen Funktionsweisen des Tools Cucumber erfolgen, sondern nur der Bereich des Testens konzentriert betrachtet wird. Es findet keine Analyse zu weiteren Strukturen, in die Cucumber eingegliedert werden könnte, statt, sondern lediglich die Untersuchung im Rahmen von DevOps.

2 Vorstellung des Unternehmens

Im Folgenden wird das Unternehmen vorgestellt, in dem die praktische Anwendung umgesetzt wurde. Die unterschiedlichen Bereiche des Unternehmens werden erläutert.

2.1 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]

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.

2.2 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]

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

[1] Vgl. IBM Unternehmensstruktur

[2] Vgl. IBM Deutschland

Abbildung 1: Die Unternehmensstruktur der IBM[3]

2.3 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]

2.4 IBM – DevOps Center of Competence Europe

Das DevOps Center of Competence (CoC) Europe 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“(RTC), „Rational DOORS next Generation“ (RDNG) und „Rational Quality Manager“ (RQM). Die Arbeitsablaufsplanung sowie das Management in diesem Bereich wird durch das RTC abgedeckt. Außerdem werden Build-Prozesse sowie das Source Code Management durch das RTC abgedeckt. Im RDNG wird das Requirements Management angeboten und das RQM wird zum Anlegen des Testmanagements genutzt. Die Tools bilden so sämtliche Tätigkeiten des

[3] Quelle: Eigene Erstellung in Anlehnung an IBM Unternehmensstruktur

[4] Vgl. IBM GBS

Softwarelebenszyklus ab und werden deshalb auch als Collaborative LifeCycle Management Tools bezeichnet.

Seit September 2016 ist diese Produktpalette durch das DevOps CoC um Open Source und Third Party Produkte erweitert worden, um den Ansprüchen von Projekten im DevOps Umfeld zu genügen.

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

3 Einführung in die theoretischen Grundlagen

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

3.1 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] 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 eXtreme Programming (XP) vorgestellt.

3.1.1 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] 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

[5] Vgl. Wolf und Bleek (2011), S. 8

[6] Vgl. Boehm (1981), S. 35

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. 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] Durch die Validierung wird der Nutzen geprüft und über die Verifizierung findet die Überprüfung der Vorgaben der Anforderungen statt.[8] In der folgenden Abbildung 2 ist das Wasserfallmodell grafisch dargestellt.

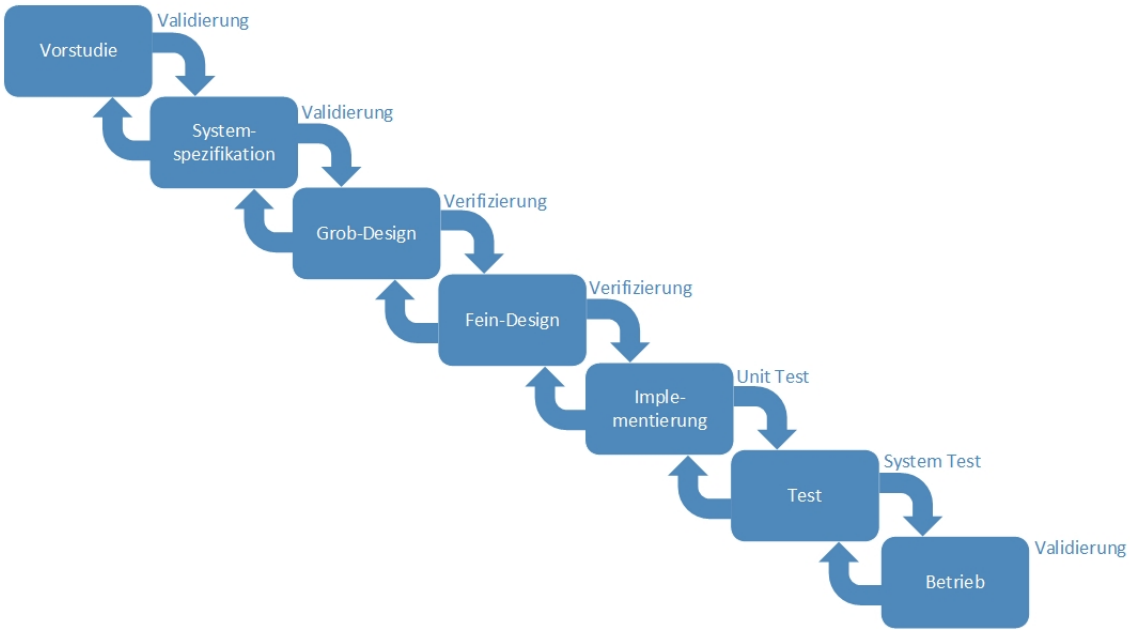

Abbildung 2: Das Wasserfallmodell nach Boehm[9]

Die Modelle von Boehm und Royce unterscheiden sich in der zuvor erläuterten Qualitätskontrolle, welche Royce nicht eingeplant hat.

Das Modell nach Royce setzte sich in der Praxis aufgrund der mangelnden Qualitätskontrolle nicht durch.[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.

[7] Vgl. Royce (1970)

[8] Vgl. Boehm (1981), S. 37

[9] Quelle: Eigene Erstellung in Anlehnung am Boehm (1981), S. 36

[10] Vgl. Onepage Wasserfallmodell

Das Modell nach Boehm besticht durch Einfachheit, jedoch sind an dieser Stelle auch die Grenzen des Vorgehensmodells gesetzt. Durch die erst spät angesiedelte Testphase sind nachträgliche Änderungen oft notwendig, da Probleme der Software auftreten. So entstehen große Aufwände sowohl im monetären als auch im zeitlichen Rahmen, da durch einen Rücksprung in den Phase alle folgenden Phasen nochmals bearbeitet werden müssen. So ist das Modell nur zu empfehlen, wenn alle Anforderungen zu Beginn des Projektverlaufs bekannt sind.

3.1.2 Agile Softwareentwicklung

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

„Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

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

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.“[11]

Diese Grundsätze beschreiben die Flexibilität der agilen Softwareentwicklung.

Um die Ausgestaltung der agilen Softwareentwicklung hervorzuheben, wird im Folgenden 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.[12] 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[13] herzustellen.[14]

[11] Agilesmanifesto.org

[12] Vgl. Beck (2000), S. 29 ff.

[13] siehe Anhang I

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.[15]

Bei der Anwendung von XP werden Primärpraktiken[16] und bei erfahrenen Teams auch Folgepraktiken[17] ausgewählt. Die sinnvolle Kombination ergibt sich dabei aus den definierten Werten und Prinzipien. Die Abbildung 3 stellt die Primärpraktiken im äußeren und die Folgepraktiken im inneren Kreis dar.

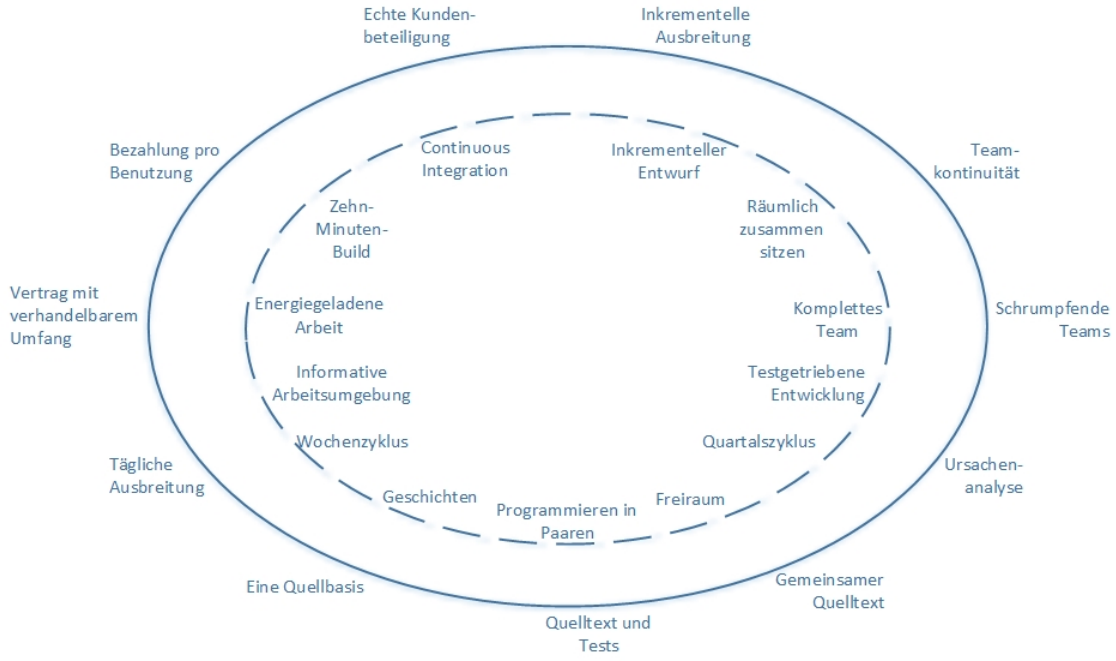

Abbildung 3: Die XP-Praktiken im Überblick[18]

Die Anforderungen werden in Form von User-Stories durch den Kunden formuliert. Diese sind Beschreibungen von dem, was das System leisten soll.[19] 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

[14] Vgl. Beck und Andres (2005), S. 14

[15] Vgl. Beck (2000), S. 140 ff.

[16] siehe Anhang II

[17] siehe Anhang III

[18] Quelle: Eigene Erstellung in Anlehnung an Wolf und Bleek (2011), S. 156

[19] Vgl. Beck (2000), S. 89

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.[20] Die folgende Abbildung 4 verdeutlicht diesen Prozess.

Abbildung 4: Der XP-Prozess[21]

3.2 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 Folgepraktik genutzt.[22]

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

[20] Vgl. Wolf und Bleek (2011), S. 160

[21] Quelle: Eigene Erstellung

[22] siehe Anhang II

3.2.1 Bedeutung und Herkunft

Software wird getestet, um sicherzustellen, dass diese einwandfrei an den Kunden ausgeliefert wird. Durch Testen werden also die Mängel in der Software vor der Auslieferung an den Kunden festgestellt. Es werden im Folgenden beispielhaft die vier wichtigsten Testarten vorgestellt.

Testart	**Beschreibung**
Unit Test	Der Test wird als erstes durchgeführt und testet den Programmcode auf die Richtigkeit.
Akzeptanztest	Dieser Test beschäftigt sich mit der Überprüfung der Funktionen der Software anhand der vom Kunden erstellten Anforderungen.
Integration Test	Dieser Test stellt die Zusammenwirkung der einzelnen Softwareelemente sicher.
System Test	Es werden sämtliche Anforderungen, die vom Kunden erstellt wurden, geprüft und das gesamte erstellte System daraufhingehend getestet.

Tabelle 1: Vorstellung der wichtigsten Testarten[23]

Es ist anzumerken, dass Anforderungen sich in funktionale und nichtfunktionale Anforderungen aufgliedern. Die funktionalen Anforderungen konzentrieren sich auf die zu erfüllenden Leistungen der Software. 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.[24]

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

[23] Quelle: Eigene Erstellung in Anlehnung an Testing Excellence

[24] Vgl. Anforderungsmanagement

Auslieferung der Software entdeckt, führt dies zu enormen Kosten und einem hohen Zeitaufwand, wie bei dem Wasserfallmodell[25].[26]

Um diese ernormen Aufwände zu reduzieren, empfiehlt die agile Vorgehensweise XP in einer der Folgepraktiken vor dem Schreiben des Programmcodes zu testen. Die Testfälle müssen also geschrieben werden, bevor die Software entwickelt wird. Dieses Vorgehen wird Test-Driven Development[27] (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. Erstellung/ 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.[28]

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, so dass dieser den Test besteht, folgt die Überarbeitung. Dopplungen sollten nicht mehr im Programmcode vorhanden sein und alle Standards sollten umgesetzt sein.[29] Dieser Kreislauf ist im Folgenden anhand der Abbildung 5 dargestellt.

[25] Vgl. Kapitel 3.1.1 klassische Softwareentwicklung

[26] Vgl. Tutorialspoint

[27] Übersetzung: Testgetriebene Entwickelung, häufig wird auch „testgetriebener Entwurf“ genutzt

[28] Vgl. Wolf und Bleek (2011), S. 99

[29] Beck (2003), S. X

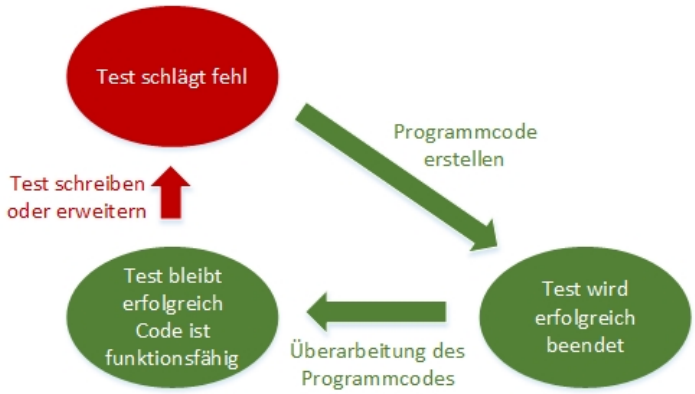

Abbildung 5: Der Red-Green-Refactor Cycle[30]

3.2.2 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.[31] Dabei wird die Kommunikation zwischen den unterschiedlichen Beteiligten an einem Softwareentwicklungsprojekt betont.[32] Durch diesen Aspekt ist BDD in die agile Softwareentwicklung als Prinzip einzuordnen.[33]

Die grundlegende Funktionsweise von BDD baut auf den unterschiedlichen Verhaltensweisen einer Software, die von den Stakeholdern[34] 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.[35] Ein Beispiel für eine User-Story ist in der Tabelle 2 dargestellt.

[30] Quelle: Eigene Erstellung in Anlehnung an Wolf und Bleek (2011), S. 99

[31] Vgl. Solís und Wang (2011)

[32] Vgl. Wynne und Hellesøy (2012), S. 3

[33] Vgl. Solís und Wang (2011)

[34] Stakeholder sind alle Personen, die von der unternehmerischen Tätigkeit gegenwärtig oder zukünftig betroffen sind.

[35] Vgl. North (2006)

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 2: Beispiel für eine User-Story[36]

Die User-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 User-Story?
Was möchte der Stakeholder tun?
Welcher Mehrwert ergibt sich aus dem Handeln für den Stakeholder?

Tabelle 3: Aufbau einer User-Story[37]

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, die „Then“-Information, handelt es sich um das Ergebnis, dass die Software zurückliefern soll, nachdem der Benutzer die „When“-Schritte ausgeführt hat. Also welches Verhalten wird erwartet, wenn der Benutzer etwas in einer bestimmten Situation tut.[38] Die folgende Tabelle 4 verdeutlicht dieses anhand eines Beispiels.

[36] Quelle: Eigene Erstellung in Anlehnung an North (2006)

[37] Quelle: Eigene Erstellung in Anlehnung an Solís und Wang (2011)

[38] Vgl. Solís und Wang (2011)

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 4: Beispiel für ein Szenario[39]

Daraus ergibt sich, dass bei BDD die Abnahmekriterien bereits vor der eigentlichen Programmierung erstellt werden.[40] Die Business-Verantwortlichen müssen von den Entwicklern unterstützt werden und anders herum, um alle Kriterien umzusetzen. So wird der Softwareentwicklungsprozess gemeinsam erfolgreich durchgeführt.[41]

Der nächste Schritt zur Automatisierung ist die Erstellung 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 in der natürlichen Sprache wird deutlich, dass im Rahmen von BDD auf die Verwendung des Wortes „Test“ verzichtet wird. Durch diese Faktoren soll sichergestellt werden, 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.[42]

[39] Quelle: Eigene Erstellung in Anlehnung an North

[40] Vgl. Hellesøy (2014)

[41] Vgl. Tee

[42] Vgl. North (2006)

3.2.3 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 werden.
- 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 das System tun soll, zu beantworten, als die Frage, wie eine bestimmte Anforderung implementiert werden soll.
- Die Erfüllung der Abnahmekriterien ist messbar, durch die Durchführung von BDD.

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 ausschließlich ihre Funktionen direkt in Algorithmen ausdrücken, sondern müssen diese zunächst in natürlicher Sprache definieren.

3.3 Der Konflikt innerhalb der IT-Organisation

Die IT-Organisation gliedert sich in die Softwareentwicklung (Development) und den IT-Betrieb (Operations) 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 von Anforderungen und die damit verbundene Erweiterung der Software. Von der Softwareentwicklung wird erwartet, dass alle gestellten Anforderungen möglichst schnell umgesetzt werden. So werden seitens der Entwicklung häufig neue Funktionen entwickelt und Änderungen in der Software vorgenommen.[43]

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 6.

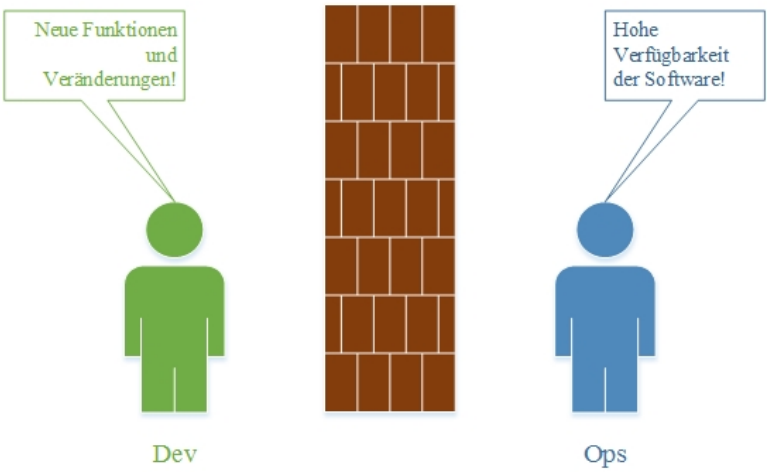

Abbildung 6: Die Isolation innerhalb der IT-Organisation[44]

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 an den IT-Betrieb weitergegeben werden kann, damit dieses Team die Software in der Produktivumgebung einsetzt.

Nachdem der IT-Betrieb dies umgesetzt hat, ist die Verfügbarkeit der Software häufig nicht mehr gegeben, da Fehler, welche in der Testumgebung noch nicht deutlich waren, nun erst auffällig 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

[43] Vgl. Peschlow (2012)

[44] Quelle: Eigene Erstellung in Anlehnung an Hüttermann (2012), S. 20

gemeinsamen Ziel gearbeitet. Diese entstehende Uneinigkeit und die daraus resultierenden Konflikte werden als „Blame Game“ bezeichnet.[45]

3.4 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.

3.4.1 Defintion von DevOps

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 sowohl der technische Hintergrund als auch die nichttechnischen kulturellen Aspekte verstanden werden.“[46] Ein Mitarbeiter der IBM definiert DevOps in einem Blog dagegen etwas anders. „DevOps ist die unternehmerische Fähigkeit, die kontinuierliche Bereitstellung von Software zu nutzen, um den Zeitpunkt bis zum Kunden-Feedback zu reduzieren und dem Kunden durch schnell verfügbare Software neue Marktchancen zu ermöglichen.“[47]

Um eine eigene Definition für DevOps aufzustellen, ist es notwendig, die grundlegenden 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.[48] Dies verlangt vor allem die heutige Marktsituation. Die Kunden erwarten aber nicht nur frühzeitige Lieferungen, sondern ebenfalls eine kurzfristige Reaktion auf Änderungen. Somit ist auch die Integration einer Rückmeldungsschleife zwischen Kunden und dem Projektteam unverzichtbar.

Es wird kein feststehender Prozess vorgeschrieben, sondern der Fokus liegt bei der Zusammenarbeit der Beteiligten, sowohl der eingesetzten Prozesse und Werkzeuge als auch der

[45] Vgl. Bass et al. (2015), S. 27

[46] Vgl. Verona (2016), S. 1 (aus dem Englischen übersetzt)

[47] Vgl. Minich (2013) (aus dem Englischen übersetzt)

[48] Vgl. Hüttermann (2012), S. 7 f.

Mitarbeiter.[49] 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.[50] Jegliche Barrieren innerhalb der IT-Organisation sollen reduziert und so die reibungslose Zusammenarbeit garantiert werden. Dies verbildlicht die Abbildung 7.

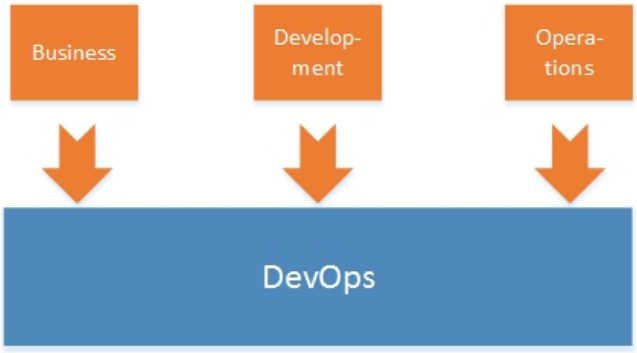

Abbildung 7: Die Einbindung der Einheiten der IT-Organisation in DevOps

Diese Inhalte von DevOps führen zu der folgenden Definition, die im Rahmen dieser Arbeit gelten soll:

„DevOps bildet die Vereinigung der unterschiedlichen Einheiten der IT-Organisation ab. Von diesem so entstehenden funktionsübergreifenden Team werden Methoden eingesetzt, die zu häufigeren Releasen und somit häufigerem Feedback der Stakeholder führen."

[49] Vgl. Dawson (2016)

[50] Vgl. Wolff (2016), S. 238

3.4.2 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.[51] 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.[52] 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.[53] 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.[54]

Durch den Einsatz von standardisierten Tools wird die Arbeit erleichtert.[55] 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 entstehen, wird die Kontrolle trotz der Geschwindigkeit innerhalb der Erstellung gewährleistet. [56]

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.[57]

[51] Vgl. Kapitel 3.3 Der Konflikt innerhalb der IT-Organisation

[52] Vgl. Swartout (2012), S. 22

[53] Vgl. Swartout (2012), S. 54 f

[54] Vgl. Davis und Daniels (2016), S. 42 ff

[55] Vgl. Verona (2016), S. 39 ff.

[56] Vgl. Bass et. al. (2015), S. 11

[57] Vgl. Onepage DevOps

Die beschriebene Kernelemente werden in der Abbildung 8 nochmals zusammengefasst dargestellt.

Abbildung 8: Die Kernelemente von DevOps[58]

So ergibt sich aus diesen Kernelementen sowie aus der Definition von DevOps auch der Mehrwert für die Unternehmen. Dieser liegt in den kürzeren Zeitabständen zwischen den Releases und dem damit verbundenen häufigerem Feedback. Außerdem wird der Engpass, der durch den zuvor beschriebenen Intressenkonflikt, im Bereich des IT-Betriebs entsteht, durch die Automatisierungsprozesse vermieden. Ebenfalls wird die Qualität der Software durch die Automatisierung erhöhrt, da weniger Fehler gemacht werden können. Auf dieser Basis verringert sich außerdem die Anzahl der Systemausfälle. Die folgende Abbildung 9 verdeutlicht die unterschiedlichen Aspekte des Mehrwerts nochmals.

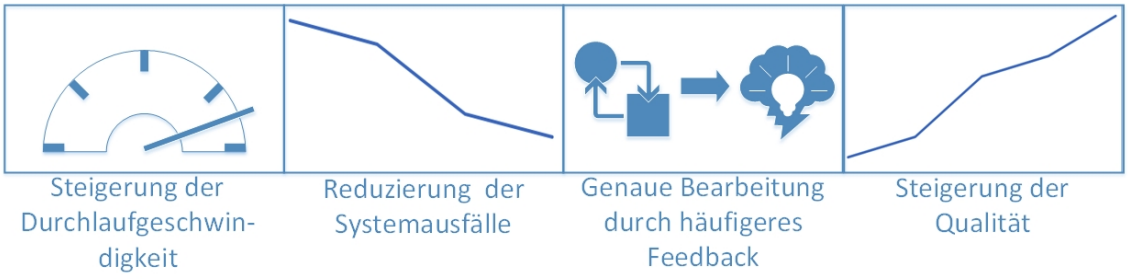

Abbildung 9: Der Mehrwert von DevOps[59]

[58] Quelle: Eigene Erstellung

[59] Quelle: Eigene Erstellung in Anlehnung an Xoomtrainings (2015)

3.4.3 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.[60] Somit ist eindeutig, dass agile Softwareentwicklung eigentlich nur durch den Einsatz von DevOps vollständig umgesetzt werden kann.[61] 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.[62]

3.5 Die DevOps Praktiken

Es werden fünf Praktiken bei DevOps zur Ausgestaltung des gesamten Softwareentwicklungsprozesses angewendet. Kern dieser Praktiken sind Automatisierung und Standardisierung.[63]

In der Abbildung 10 wird die Reichweite der einzelnen Praktiken grafisch dargestellt und im Anschluss erläutert.

[60] Vgl. Aiello und Sachs (2016), S. 214

[61] Vgl. Aiello und Sachs (2016), S. 222

[62] Vgl. Onepage DevOps

[63] Vgl. Kapitel 0 Kernelemente von DevOps

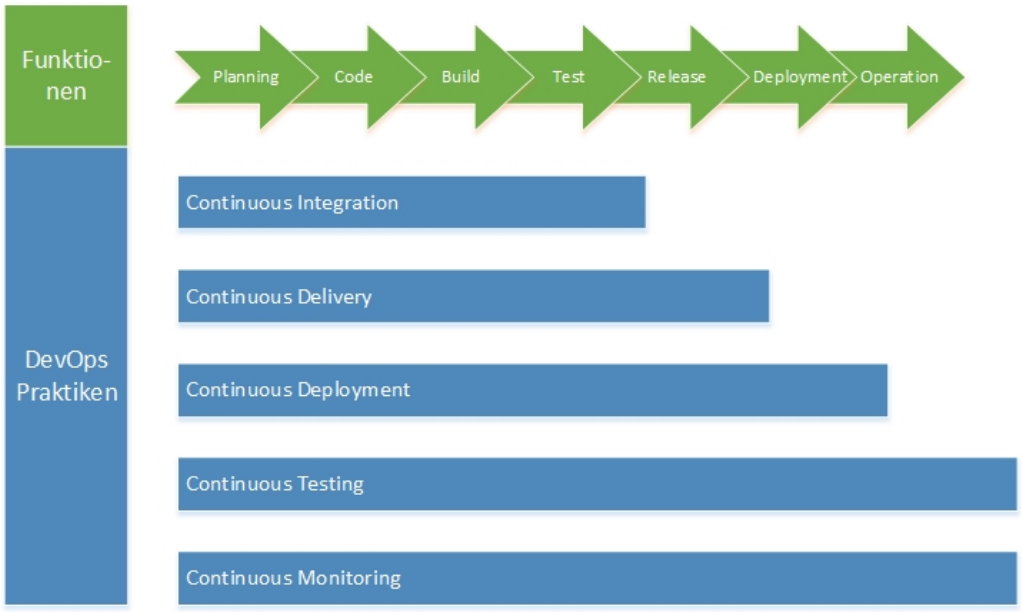

Abbildung 10: Die Reichweiten der verschiedenen DevOps Praktiken[64]

3.5.1 Continuous Integration

Der Prozess von Continuous Integration[65] (CI) beschreibt das Vorgehen der Integration der neuen Programmcodeelemente 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 einen Build. Ein Build ist dabei die Übersetzung des Programmcodes in die ausführbare Software. Nach der Fertigstellung des Builds wird die Komplierbarkeit 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 einen Entwickler weitergibt. Ist das Ergebnis positiv, kann der Programmcode freigegeben werden. Wenn der CI-Server allerdings ein negatives Ergebnis meldet, muss ein Entwickler den Fehler beheben und einen neuen Build anstoßen.[66] Dieser Ablauf ist in der Abbildung 11 dargestellt.

[64] Quelle: Eigene Erstellung in Anlehnung an Martin (2014)

[65] Übersetzung: kontinuierliche Integration

[66] Vgl. Edwards (2010)

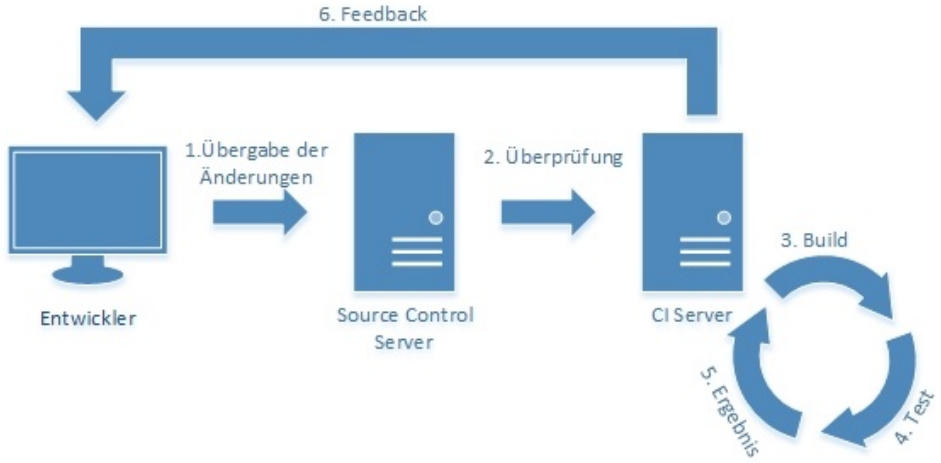

Abbildung 11: Der Continuous Integration-Prozess[67]

Durch CI wird somit die Problemstellung der reibungslosen Zusammenarbeit innerhalb der Entwicklung gelöst. Mehrere Entwickler können gleichzeitig an einem Branch[68] oder mehreren Branches des Programmcodes arbeiten. Durch die kontinuierliche Integration und automatische Testfunktion ist die Einbindung eines Branches 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.

3.5.2 Continuous Delivery

Continuous Delivery[69] (CD) verfolgt das Ziel, den Prozess bis zu dem neuen Release[70] der Software zu optimieren. Ursprünglich wurde die Einführung neuer Software oft am Wochenende durchgeführt.[71] Es wurde zwar versucht, eine lauffähige Software bereitzustellen, dennoch war dies oft nicht gegeben. Durch manuelle Bearbeitung wurde dieser sehr komplexe Prozess noch langwieriger und ebenfalls fehleranfälliger, folglich wurden Release nur selten in der Produktion eingesetzt.

[67] Quelle: Eigene Erstellung in Anlehnung an Verona (2016), S.14

[68] Ein Zweig des Programmcodes, der vom übrigen Programmcode gelöst wurde, um einen einzelnen Arbeitszweig zu erhalten. Der Hauptzweig des Programms wird Master-Branch genannt.

[69] Übersetzung: kontinuierliche Auslieferung

[70] Beschreibt den Versionsstand einer Software.

[71] Vgl. Wolff (2016), S. 13

Durch die Einführung von CI werden Fehler zwar 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 Artefakt-Repository möglich. Sämtliche Dateien, die im Build erzeugt wurden, werden in diesem Repository gespeichert.

Regressionen sollten in der Softwareentwicklung vermieden werden, diese treten ein, wenn ein neues Softwarerelease erstellt wurde und dieses in das vorhandene System eingebunden wird. Dadurch werden zuvor fertiggestellte Elemente der Software erneut verändert, deshalb müssen diese Elemente nochmals getestet werden. So entsteht ein kontinuierlich wachsender Testprozess, welcher nur durch Automatisierungen zu bewältigen ist.

Die folgende Abbildung 12 zeigt die CD-Pipeline, diese gliedert sich in fünf Phasen auf.

Abbildung 12: Die CD-Pipeline[72]

Der erste Schritt dieser Pipeline wird „Commit“[73] bezeichnet und beschreibt den CI-Prozess der Erstellung der Builds und Unit Tests.[74] In der folgenden Phase „Akzeptanztest“ wird die Interaktion des Releases über die Benutzeroberfläche getestet. Bei der Automatisierung dieser Tests besteht auch die Möglichkeit, die Testfälle auf Grundlage der Anforderungen in natürlicher Sprache zu formulieren.[75] 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

[72] Quelle: Eigene Erstellung in Anlehnung an Wolff (2016), S. 24

[73] Normalerweise wird in der Softwareentwicklung mit “Commit” das freigeben von Änderungen im Programmcode bezeichnet, dies ist an diesem Punkt nicht gemeint.

[74] Vgl. Kapitel 3.5.1 Continuous Integration (CI)

[75] Vgl. Kapitel 3.2 Behaviour-Driven Development

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.[76] Nachdem diese Phase abgeschlossen ist, findet ein Feedback mit dem Kunden statt, wodurch die Zufriedenheit gemessen wird.[77]

Sollte diese Pipeline an einer Phase scheitern, so ist die Software zu überprüfen. Der Fehler ist 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.

3.5.3 Continuous Deployment

Der Prozess von Continuous Deployment[78] knüpft sich an die CD-Pipeline an. So führt bei Continuous Deployment jede Änderung im Programmcode automatisch zu einer Installastion in der Produktivumgebung.[79] Somit befasst sich Continuous Deployment mit der Automatisierung des letzten Schritts der CD-Pipeline.[80] Dieser Unterschied wird in der folgenden Abbildung 13 verdeutlicht.[81]

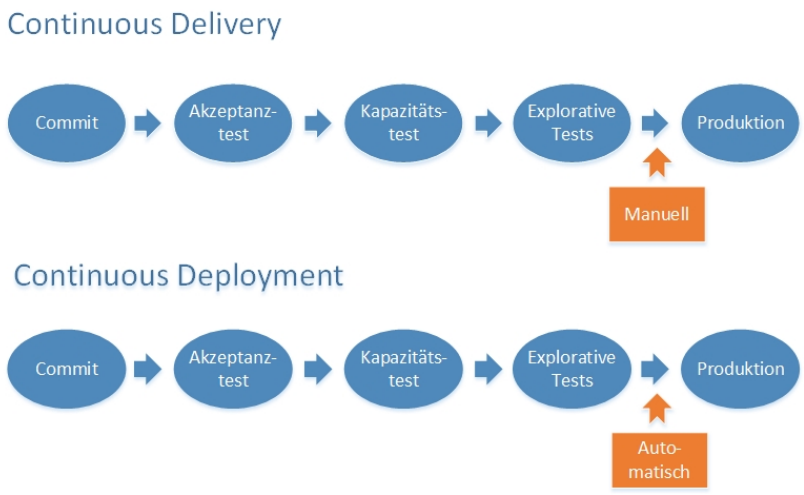

Abbildung 13: Die Continuous Deployment-Pipeline im Vergleich[82]

[76] Vgl. Test Exellence

[77] Vgl. Prowareness

[78] Übersetzung: kontinuierliche Einführung

[79] Vgl. Fowler (2013)

[80] Vgl. Kapitel 3.5.2 Continuous Delivery

[81] Vgl. Caum (2013)

[82] Quelle: Eigene Erstellung in Anlehnung an Macvittie (2016)

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 kann ein vorheriger Versionsstand der Software automatisch wiederhergestellt werden. Ein Rollforward dagegen sieht die schnelle Beseitigung des Fehlers, durch die automatischen Freigabe einer neuen fehlerfreien Version vor.[83]

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.[84]

3.5.4 Continuous Testing

Bei der vierten DevOps Praktik handelt es sich um Continuous Testing[85] (CT).

In der CD-Pipeline[86] sowie in der Continuous Deployment-Pipeline[87] wird bereits deutlich, in welchem Detailgrad das Testen 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. Im Softwareentwicklungsprozess ist das Testen wie in der Abbildung 14 dargestellt, angesiedelt.

[83] Vgl. Wolff (2016), S. 14 ff.

[84] Vgl. Davis und Daniels (2016), S. 39

[85] Übersetzung: kontinuierliches Testen

[86] Vgl. Kapitel 3.5.2 Conitnuous Delivery (CD)

[87] Vgl. Kapitel 3.5.3 Continuous Deployment

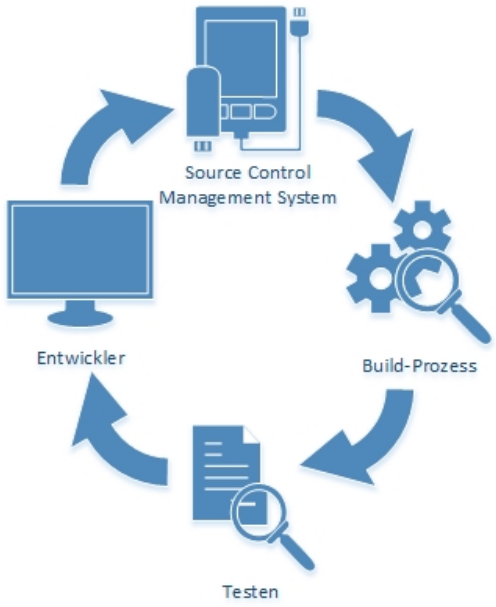

Abbildung 14: Der CT-Prozess[88]

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.[89]

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.[90]

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.[91]

[88] Quelle: Eigene Erstelung

[89] Vgl. Witte (2015), S.158

[90] Vgl. Aiello und Sachs (2016), S. 35

[91] Vgl. Testing Excellence

3.5.5 Continuous Monitoring

Continuous Monitoring[92] (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.[93] 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.[94] Dieser Prozess wird in der Abbildung 15 verdeutlicht.

Abbildung 15: Der CM-Prozess[95]

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.[96] Verläuft die Überwachung der IT-Systeme konstant, so werden Veränderungen jeglicher Art verzeichnet und es kann auf diese individuell reagiert werden.

[92] Übersetzung: kontinuierliche Überprüfung

[93] Vgl. Bass et. al. (2015), S. 130 ff.

[94] Vgl. Erwin (2013)

[95] Quelle: Eigene Erstellung

[96] Vgl. Bass et. al. (2015), S. 144

3.6 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 durch die Vermeidung von manueller Arbeit weiter reduziert.
- Die Kundenzufriedenheit wird außerdem durch die Stabilität der Verfügbarkeit der Software weiter gesteigert.

Nachteile:

- Es entstehen durch die notwendigen Umstellungen hohe Kosten im Bereich der 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 und in welchem Maße diese in das betreffende Unternehmen einzuführen sind.

3.7 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 anversiert 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 Prozessen ein Kernelement. Es soll möglichst wenig manuelle Arbeit verrichtet werden. Durch dieses Vorgehen werden Fehler vermieden und das Risiko minimiert.

Behaviour-Driven Development	**DevOps**
Kommunikation zwischen den Beteiligten während des Entwicklungsprozesses.	Arbeit innerhalb eines funktionsübergreifenden Teams.
Das Ziel ist die Erstellung einer fehlerfreien Software.	Das Ziel ist die frühzeitig verfügbare fehlerfreie Software.
Automatisierte Überprüfungen der zuentwickelnden Software durch den Einsatz von Tools.	Erstellung von möglichst vielen automatisierten Tests durch den Einsatz von Tools. Manuelle Tätigkeiten sollen vermieden werden.
Durch die kontinuierliche Durchführung der Überprüfungen der Abnahmekriterien entsteht ein häufiges Feedback.	Es wird ein häufiges Feedback durch häufige Auslieferungen an den Kunden erreicht.

Tabelle 5: Vergleich von BDD und DevOps

Diese Punkte machen deutlich, dass BDD und DevOps sehr gut harmonieren. DevOps bildet dabei einen umfangreicheren Rahmen, in den BDD im Bereich von CT eingegliedert werden kann. Somit sollte an den gemeinsamen Zielen beider Praktiken festgehalten werden und so die Zusammenarbeit weiter unterstützt werden.

4 Praktische Umsetzung von Behaviour-Driven Development

Im Folgenden Kapitel wird BDD anhand eines Projekts der IBM vorgestellt.

4.1 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.[97]

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 Schritte „Given“, „When“ und „Then“ in einer Datei gespeichert. Diese Datei enthält somit nur natürliche Sprache. Im Programmcodeelement 1 ist 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
```

[97] Cucumber.io

```
When he clicks on the button „Contact"

Then he would see the contact details
```

Programmcodeelement 1: Beispiel für ein Feature und Szenario in Cucumber

Im Anschluss 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 durch den Ausdruck „{VARIABLE:stringInDoubleQuotes}" einer Variablen zugeordnet. Die Variable kann dann in die folgende Funktion übergeben werden, außerdem ist eine mehrfache Verwendung möglich. Durch dieses Vorgehen wird das Ausschreiben von Textelementen erspart und die Programmierung erleichtert. Außerdem erfolgt eine sinnvolle Navigation innerhalb des Programmcodes durch festgelegte 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.[98]

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

Der Ablauf der Erstellung der unterschiedlichen Elemente ist in der folgenden Abbildung 16 dargestellt.

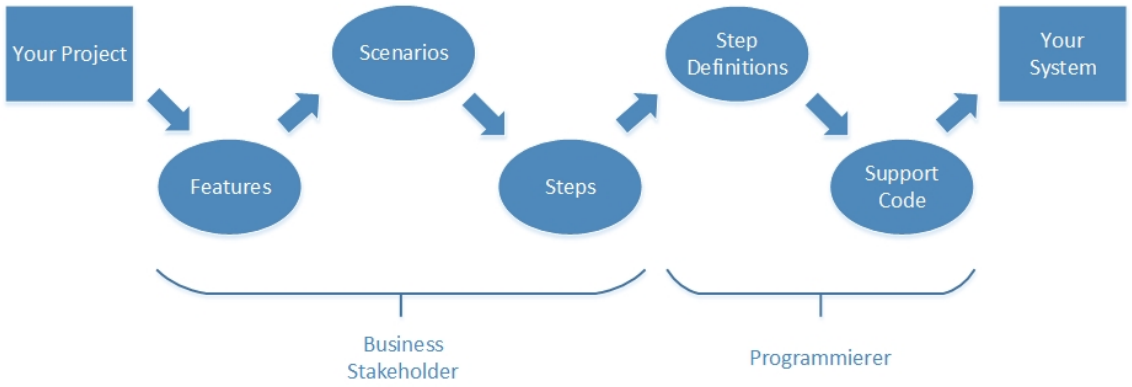

Abbildung 16: Die Funktionsweise von Cucumber[99]

Wenn das Tool alle Schritte 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 Blau als übersprungen oder in Gelb als nicht auffindbar markiert.

[98] Wynne und Hellesy (2012), S. 8

[99] Quelle: Eigene Erstellung in Anlehnung an Wynne und Hellesøy (2012), S. 8

4.2 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 die folgenden Rollen unterschieden: keine, Berater, Architekt, Release Manager und Implementation Spezialist.

Bevor dieses Projekt begonnen hat, wurde die Auswertung der Skills mühsam per Microsoft Excel-Dateien gepflegt. Jeder Mitarbeiter erhielt einen Fragebogen in Form einer Microsoft 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.

4.3 Anforderungen an das Tool

Die Anforderungen an das Tool wurden durch Gespräche mit den Stakeholdern in einer Microsoft Word-Datei gesammelt und festgehalten. Eine weitere Strukturierung der Anforderungen erfolgte zu diesem Zeitpunkt nicht. Es wurde anhand von Beispielen festgemacht, welche Anforderungen priorisiert werden. So soll der Login über den IBM-internen Single Sign-On[100] Service durchgeführt werden. Das Besondere an diesem ist die Notwendigkeit des Intranets[101] 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. Eine andere Möglichkeit

[100] Single Sign-On hat den Vorteil, dass ein Benutzer nach einer einmaligen Authentifizierung jegliche Services nutzen kann.

[101] Ein Intranet ist ein unternehmensinternes Computernetzwerk.

ist, dass der Mitarbeiter bereits Daten eingegeben hat. In diesem Fall werden ihm die Daten 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. Der Mitarbeiter soll seine Daten löschen können, dies funktioniert über einen „Delete“-Button. Nach dem Anklicken des Buttons sind sämtliche Daten des Benutzers von der Datenbank entfernt.

Die folgende Abbildung 17 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.

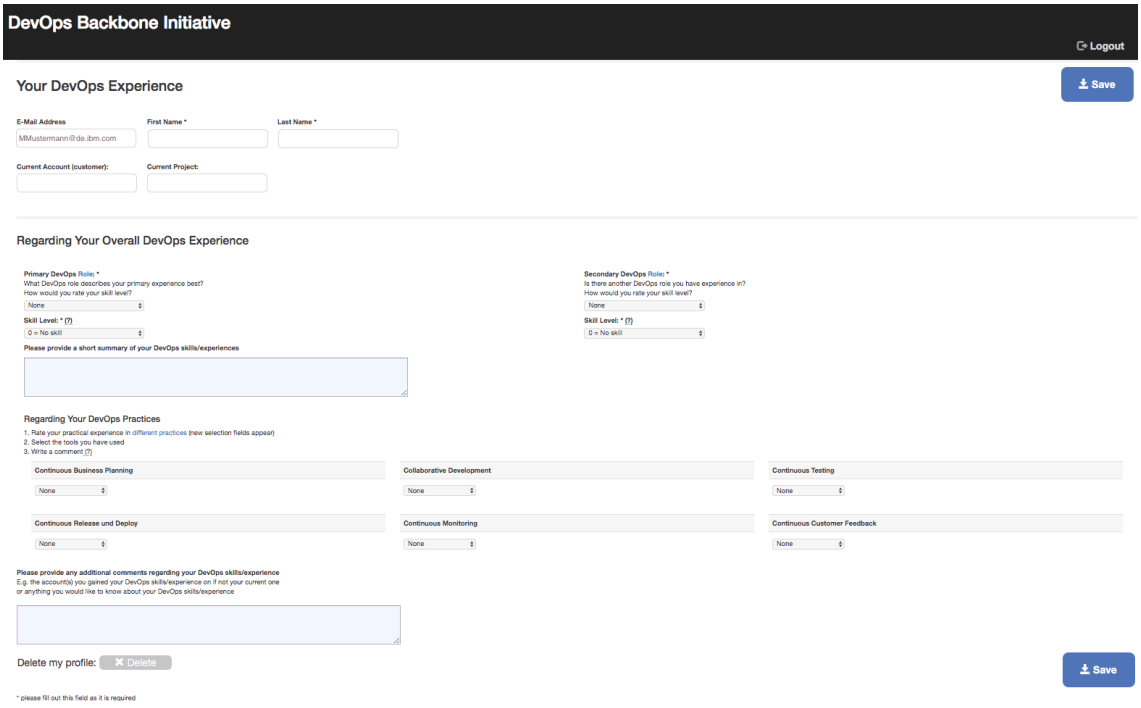

Abbildung 17: Das Tool im Überblick[102]

[102] Quelle: Eigene Erstellung

4.4 Selenium

Selenium ist eines der bekanntesten OpenSource Testtools. Es wird in erster Linie für die Automatisierung von Web-Anwendungen für Testzwecke genutzt. So kann Selenium Formulare einer Website automatisch ausfüllen, Checkboxen anklicken, in Dropdown Menüs auswählen und Links in Dokumenten anklicken.[103] Durch Integration von Selenium in Cucumber ist die Bedienung des Webbrowsers möglich. Die Einbindung kann problemlos über den Programmcode erfolgen und ist für das verwendete Beispiel unverzichtbar.

4.5 Ordnerstruktur der Tests

Cucumber greift auf die Dateien der Features zu, um dann den Bezug zu den Step Definitions zu erstellen. Damit dies gelingt, ist die Struktur der Dateitypen relevant. Die Dateien, die die Features beinhalten, werden mit der Dateiendung „.feature“ gekennzeichnet. Die anderen benötigten Dateien, also sowohl der Supportcode als auch die Step Definitions, werden nach der Programmiersprache typisiert, dies bedeutet bei dieser Durchführung „JavaScript“, also die Dateiendung „.js“.

In der folgenden Abbildung 18 wird die Struktur der Ordner und Dateien dargestellt, um einen Überblick der Tests zu geben.

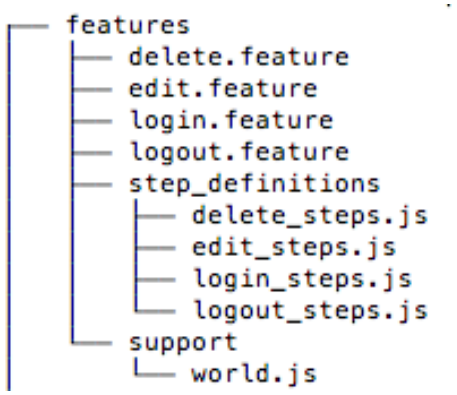

Abbildung 18: Der Überblick der Ordnerstruktur[104]

[103] Vgl. Selenium (2017)

[104] Quelle: Eigene Erstellung

4.6 Browser-Steuerung

Bevor die Überprüfungen der Website durchgeführt werden können, muss festgelegt werden, in welchem Browser diese gestartet werden sollen. Dies wird im Programmcodeelement 2 dargestellt.

```
//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); //20000 Millisekunden = 20
	Sekunden
	});
```

Programmcodeelement 2: Supportcode

Der Treiber des Browsers muss angesprochen werden, damit die Durchführung stattfinden kann. Um Cucumber die Tests automatisch durchführen zu lassen, 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. Da beim Login die Ladezeiten der Website mehr als die Standardlaufzeit von Cucumber betragen, muss diese von fünf Sekunden bis auf 20 Sekunden erweitert werden.

4.7 Definition von Features und Szenarien

Die unstrukturierten Anforderungen aus der Mircosoft Word-Datei wurden im Folgenden zu User-Stories und Szenarien formuliert. Als Beispiel wird der Login-Vorgang verwendet. Weitere Ausführungen werden im Anhang vorgestellt. [105]

Die Login-Funktion ist notwendig, damit ein Benutzer seine Skills bearbeiten kann und kein Zugriff und keine Bearbeitung durch andere Personen möglich ist. Hieraus ergibt sich das Feature wie im Programmcodeelement 3 dargestellt.

```
//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
```

Programmcodeelement 3: Feature

Für dieses Feature ergeben sich zwei Szenarien, der erfolgreiche Login und der fehlgeschlagene Login. Der Benutzer muss also auf der Website der „DevOps Backbone Intiative“ sein, um sich einloggen zu können. Der Login ist über die IBM-interne User ID und dem Passwort möglich. Nach der Eingabe muss der „Sign In“-Button angeklickt werden, um das Profil des Benutzers angezeigt zu bekommen. Bei einer Falscheingabe des Passworts wird eine Fehlermeldung angezeigt. Da in beiden Szenarien die Voraussetzung besteht, dass die Website besucht wurde und die Eingabe der User ID erfolgte, kann diese Informationen in einen „Background“-Schritt erläutert werden. Alle weiteren Informationen sind dann wieder dem jeweiligen Szenario zugeordnet. Im Programmcodeelement 4 sind der Background und die Szenarien dargestellt.

```
    Background:

        Given as an 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
```

[105] 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.

```
        When I click on "Sign In"

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

    Scenario: Failed 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
```

Programmcodeelement 4: Background und Szenarien

Wenn nun Cucumber gestartet wird und somit der erste Testlauf beginnt, wird wie in der folgenden Abbildung 19 dargestellt, das Feature mit den Szenarien und Schritten angezeigt. Die Fragezeichen in den Zeilen der einzelnen Informationen geben Aufschluss darüber, dass das Tool hierzu noch keine Step Definitions findet.

```
        -MBP:~        $ ./node_modules/.bin/cucumber-js
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

  Scenario: Successful Login on “DevOps Backbone Initiative”-Website
  ? Given As a IBM Employee I am on the Website "DevOps Backbone Initiative"
  ? When I fill in my User ID
  ? 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
  ? Given As a IBM Employee I am on the Website "DevOps Backbone Initiative"
  ? When I fill in my User ID
  ? When I fill in a wrong password
  ? When I click on “Sign In”
  ? Then I should see an Error Message
```

Abbildung 19: Die Übersicht der fehlenden Schritte in Cucumber[106]

Um dann die Step Definitions zu erstellen und somit die Funktionsfähigkeit der einzelnen Schritte einzufügen, gibt Cucumber eine Hilfestellung. Cucumber zeigt an, welche Funktionen im Rahmen der Step Definition eingefügt werden müssen, damit der Test positiv verläuft. Beispielhaft ist die in der nächsten Abbildung 20 für den Step der „Given“-Information formuliert. Anhand dieser Vorgaben ist die Umsetzung mittels des Vorgehens des Red-Green-Refactor Cycles[107] sehr einfach möglich.

[106] Quelle: Eigene Erstellung

[107] Vgl. Kapitel 3.2 Behaviour-Driven Development

```
Warnings:

1) Scenario: Successful Login on "DevOps Backbone Initiative"-Website - features/Login.feature:10
   Step: Given As a IBM Employee I am on the Website "DevOps Backbone Initiative" - features/Login.feature:7
   Message:
     Undefined. Implement with the following snippet:

       Given('As a IBM Employee I am on the Website {arg1:stringInDoubleQuotes}', function (arg1, callback) {
         // Write code here that turns the phrase above into concrete actions
         callback(null, 'pending');
       });
```

Abbildung 20: Die Anzeige einer fehlenden Funktion[108]

4.8 Erläuterung der Step Definition

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

Als Erstes wird im Programmcodeelement 5 das Starten des Treibers für Selenium durchgeführt. 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');
```

Programmcodeelement 5: Aufruf der Treiber

Im Programmcodeelement 6 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.ne
        t/");
```

[108] Quelle: Eigene Erstellung

```
        });
```

Programmcodeelement 6: Given-Schritt

Im Programmcodeelement 7 wird die User ID automatisch in das Feld namens “username” eingetragen. Dasselbe wird danach mit dem Passwort durchgeführt. Außerdem erfolgt der Login durch das Anklicken des Buttons „Sign In“.

```
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("correctpassword");
        });
        });
When('I click on “Sign In”', function () {
        return this.driver.findElement({id:
        "btn_signin"}).then(function(element) {
        return element.click();
        });
        });
```

Programmcodeelement 7: When-Information mit der Eingabe des richtigen Passworts

Als Resultat soll dann das Profil in Programmcodeelement 8 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);

});
```

Programmcodeelement 8: Then-Information für den erfolgreichen Login

Da auch das Szenario existiert, dass ein Benutzer ein falsches Passwort eingibt, setzt das Programmcodeelement 9 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");

});

});
```

Programmcodeelement 9: When-Information mit der Eingabe eines falschen Passworts

Es soll dann auf der Login-Seite eine Fehlermeldung ausgegeben werden, die über eine Funktion im Programmcodeelement 10 geprüft wird, welche den Titel der Website überprüft. Wird 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);

});

});
```

Programmcodeelement 10: Then-Information für den fehlgeschlagenen Login

Damit ist die Step Definiton für den Login Prozess abgeschlossen. Sollten 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.[109]

4.9 Ausführung von Cucumber

Nach dem Starten des Tools werden die Dateien, in denen die Features und Szenarien gespeichert sind, mit der Endung „.feature“ automatisch aufgerufen. Das Tool erstellt das Mapping zu den Step Definitions, welche als JavaScript Dateien gespeichert sind. Nach der Ausführung des Support Codes wird der Test gestartet.

In der Abbildung 21 ist der erfolgreich verlaufene Test dargestellt.

```
        -MBP:~        $ ./node_modules/.bin/cucumber-js
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

  Scenario: Successful Login on “DevOps Backbone Initiative”-Website
  ✔ Given As a IBM Employee I am on the Website "DevOps Backbone Initiative"
  ✔ When I fill in my User ID
  ✔ 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
  ✔ Given As a IBM Employee I am on the Website "DevOps Backbone Initiative"
  ✔ When I fill in my User ID
  ✔ When I fill in a wrong password
  ✔ When I click on “Sign In”
  ✔ Then I should see an Error Message

2 scenarios (2 passed)
10 steps (10 passed)
0m12.741s
```

Abbildung 21: Das erfolgreiche Testresultat[110]

Es sind nochmals alle Szenarien und Schritte 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 Schritten insgesamt durchführt wurden und wie alnge die gesamte Durchlaufzeit betrug.

[109] siehe Anhang IV

[110] Quelle: Eigene Erstellung

4.10 Bewertung der Durchführung

Die Arbeit mit Cucumber stellt sich als sehr leicht und schnell erlernbar dar. Über die Vorgehensweise vom Red-Green-Refactor Cycle erfährt der Benutzer, welche Schritte notwendig sind, um eine funktionsfähige Überprüfung zu erstellen. Für den unerfahrerenen Cucumber Benutzer gibt es außerdem durch die aktive Community gibt es viele Möglichkeiten der Hilfestellung, wie beispielsweise Foren.

Das Vorgehen, dass die Erwartungen der Stakeholder durch die Feature eindeutig dargelegt werden, sorgt dafür, dass jegliche Missverständnisse nahezu unmöglich sind. 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.[111] Außerdem bietet Cucumber durch die Verwendung der natürlichen Sprache die Möglichkeit der Kommunikation auf derselben Ebene.

[111] Vgl. Rupp und die Sophisten (2014), S. 23

5 Experteninterviews

Im Verlauf des folgenden Kapitels soll eine vorherrschende Experten-Meinung zur Thematik BDD gegeben werden. Dafür werden erst die Interviewpartner vorgestellt und im weiteren Verlauf die geführten Interviews analysiert. Die Interviews wurden aufgrund der geografischen Entfernung per Telefon geführt. Die dazugehörigen Leitfäden befinden sich im Anhang.[112]

5.1 Vorstellung der Gesprächspartner

Als Experten wurden zwei Mitarbeiter des Teams DevOps CoC Europe und ein Mitarbeiter eines anderen Teams befragt. Alle drei Interviewpartner haben unterschiedliche Rollen innerhalb der IBM. So ist Markus Holzinger als IT-Architekt eingestellt, Bianca Heinemann als Business Analyst und Katrin Heymann als Strategy Consultant für Business Processes.

Der Unterschied zwischen diesen Rollen liegt darin, dass der Architekt für den Aufbau einer Systemarchitektur verantwortlich ist. Dafür werden Kenntnisse in den unterschiedlichen Systemen, Anwendungen und Methoden vorausgesetzt.[113]

Die Rolle des Business Analysten dagegen ist besonders konzentriert auf die Schnittstelle zwischen den IT- und Business-Beteiligten. Somit arbeitet Frau Heinemann mit sämtlichen Stakeholdern eines Projektes zusammen und beschäftigt sich mit der Analyse der Arbeit zwischen den Parteien. Das Erstellen von Diagrammen in der Unified Modeling Language und das Identifizieren von Use Cases gehört dabei zum Tagesgeschäft.[114]

Als Strategy Consultant für Business Processes beschäftigt sich Frau Heymann vor allem mit den Vorstellungen der Unternehmensvorstände. Bei dieser Rolle geht es um die Umsetzung der Gesamtstrategien innerhalb eines Unternehmens und aller dafür benötigten Schritte. So führt Frau Heymann beispielsweise Prozessanalysen durch und beschäftigt sich mit der Verbesserung dieser Prozesse.

[112] siehe Anhang V

[113] Vgl. IBM Intranet

[114] Vgl. IBM Intranet

5.2 Analyse der Interviews

Im Folgenden werden die Kernaussagen der Interviews analysiert. Diese Aussagen haben sich durch die Intensive Besprechung während der Interviews ergeben. Das Requirements Engineering bildet eine dieser grundlegenden Aussagen. Diese Disziplin im Softwareentwicklungprozess ist sehr wichtig, da durch Fehler oder Ungenauigkeiten in diesem Prozess oft hohe zusätzliche Aufwände entstehen. Um die Anwendung von BDD innerhalb der IBM einzuführen, wäre eine Möglichkeit laut Herrn Holzinger, BDD in das Scaled Agile Framework (SAFe), das in der Ausführung genauer erläutert wird, einzufügen. Ein weiterer wichtiger Punkt, welcher sich während der Interviews mit Frau Heymann und Frau Heinemann herauskristallisiert hat, ist die Rollenverteilung innerhalb von Projekten. Außerdem wurde besonders durch Herrn Holzinger die Thematik DevOps diskutiert und bildet damit den vierten Punkt dieser Analyse.

5.2.1 Requirements Engineering

Das Requirements Engineering ist die Disziplin der Anforderungserhebung sowie der Verwaltung der Anforderungen. In diesem Rahmen werden ebenfalls die Abnahmekritieren für die Akzeptanztests entlang der Anforderungen festgelegt. Dies stellt laut Frau Heinemann eine Problemquelle dar, denn wenn diese nicht genau und präzise formuliert werden, werden falsche oder ungenaue Abnahmekriterien formuliert, wodurch der Kunde dann eventuell nicht das gewünschte Produkt erhält. Dieses Problem führt oft zum deutlichen Überschreiten der monetären sowie zeitlichen Vorgaben. Außerdem ist die Kundenzufriedenheit eine der wichtigsten Kennzahlen für Unternehmen, um Vertrauen herzustellen und so Aufträge zu sichern.

BDD leistet an dieser Stelle nach der Meinung von Frau Heinemann große Unterstützung, da die Kriterien gemeinsam entlang der User-Stories und Szenarien formuliert werden. So wird entlang einer Schablone vorgegangen, so dass sichergestellt wird, keine Aspekte zu überspringen. Außerdem ist laut Frau Heymann das schriftliche Festhalten an dieser Stelle wichtig, da so der genaue Rahmen festgelegt ist.

Frau Heinemann bemerkte außerdem, dass durch die Einhaltung derselben natürlichen Sprache sichergestellt wird, dass keine sprachlichen Missverständnisse entstehen.

5.2.2 SAFe und BDD

In vielen Projekten der IBM wird dieses Framework[115] integriert. Unter anderem auch in dem Projekt, in dem Herr Holzinger seit einiger Zeit beschäftigt ist, deshalb brachte er dieses Framework auch mit BDD in Verbindung.

SAFe ist durch Scale Agile Incorporated entwickelt worden und beschreibt einen agilen Vorgehensrahmen, welcher sich durch die individuelle Skalierbarkeit und Modularität auszeichnet. SAFe unterstützt dabei die Ansätze des agilen Manifests. Neben dem agilen Vorgehen findet SAFe seinen zweiten Ursprung im Lean Management.[116] Lean Management ist ein Management System, welches besonders auf die strategische Ausrichtung abzielt und die Unternehmenskultur beinhaltet. Es wird bei der Durchfühung von Lean Management besonders auf die Verkürzung der Durchlaufzeit und dabei auf die Erhaltung der Qualität geachtet.[117] Dafür werden die folgenden fünf Lean Prinzipien unterschieden:

1. *„Definiere den Wert aus Sicht des Kunden.*
2. *Verstehe den Wertstrom und eliminiere Verschwendungen.*
3. *Organisiere den verbleibenden Wertstrom hin zu einem synchronisierten Fluss.*
4. *Lass den Kunden abrufen, was er braucht, und liefere es ihm genau zum richtigen Zeitpunkt (ziehendes System).*
5. *Strebe nach Perfektion („Kaizen").“*[118]

SAFe auf Basis der Ansätze Lean Management und Agilität legt laut Hernn Holzinger den grundsätzlichen Aufbau des Projekts dar. Es wird beschrieben, wie die funktionsübergreifenden Teams vorgehen sollen, um ein für den Kunden wertschöpfendes Programm zu erstellen. Je nach Größe des Unternehmens wird SAFe 3.0 und SAFe 4.0 unterschieden. SAFe gibt zu detailierten Vorgängen, wie beispielsweise das Testen, keine Auskünfte. Somit wäre für Herrn Holzinger auch eine Eingliederung von BDD im Bereich des Akzeptanztests in diesen Prozess denkbar. Eine weitere Untersuchung zu diesem Bereich ist dafür erforderlich, jedoch ist davon auszugehen, dass BDD in diesem Bereich ein Alleinstellungsmerkmal für die IBM sein würde.

[115] Ein Framework stellt eine Grundstruktur und somit einen Rahmen dar.

[116] Vgl. Scaled Agile Framework

[117] Vgl. Lean-Management

[118] Lean-Management – Prinzipien

5.2.3 Rollenverteilung

Die Schwerpunkte der unterschiedlichen Rollen, die in einem Projekt vertreten werden, ist besonders im Punkt der Kommunikation zu bedenken. Dies ist eine der Kernaussagen aus dem Interview mit Frau Heymann.

Es werden unter den Mitarbeitern eines Projekts IT-Architekten, Tester, Entwickler, Analysten und einige weitere Rollen unterschieden, dazu kommen noch die Mitarbeiter, die sich auf das Management des Projekts und die Organisation konzentrieren, die sogenannten Projektleiter. Frau Heymann bezog sich vor allem auf die verschiedenen Sichtweisen, die durch die unterschiedlichen Personen in einem Projekt vertreten werden. Laut Frau Heymann entstehen diese aber durch die verschiedenen Rollen, allerdings wird die Anforderungsanalyse dadurch erschwert. Frau Heymann unterstrich diese Aussage am Beispiel eines Entwicklers. Der Entwickler ist sehr in seinem Fachgebiet verankert, dadurch hat er ein sehr eingeschränktes Blickfeld auf das Gesamtprojekt. Für den Kunden ist diese Denkweise nicht nachzuvollziehen, weshalb Probleme bei der Anforderungserhebung durch einen Entwickler sehr wahrscheinlich sind. Trotz der eigentlich gleichen natürlichen Sprache ist die unterschiedliche Herangehensweise kritisch zu sehen. Frau Heymann empfielt deshalb eine Einhaltung der Rollen, so dass nicht Entwickler die Definition der User-Stories übernehmen, sondern diese Aufgabe weiterhin im Zuständigkeitsbereich der Business Analysten bleibt.

5.2.4 DevOps und BDD

Herr Holzinger erkannte im Interview die Möglichkeiten von BDD im Rahmen von DevOps. Die Automatisierung im Bereich BDD findet seiner Meinung nach, vor allem auf der Seite der Entwicklung statt. Es werden die Akzeptanztests erleichtert, wodurch wird in der Entwicklung eine kürzere Durchlaufzeit erreicht wird. Herr Holzinger erkannte außerdem die Aufwandsminimierung durch die Automatisierung der Abnahmekritieren.

Der Kunde profitiert nach Auffassung von Herrn Holzinger ebenfalls von der kürzeren Durchlaufzeit und seine Zufriedenheit steigt. Außerdem entsteht keine Verärgerung auf der Seite des Kundens durch die Erfüllung der falschen oder ungenauen Abnahmekriterien. Diesen Punkt haben alle drei befragten Experten dargelegt.

In einer sehr schnelllebigen Technologie, wie es die App-Entwicklung ist, sieht Herr Holzinger besonders einen Vorteil beim Einsatz von BDD im Bereich DevOps: Die Erhöhung der Testqualität und die Automatisierung im Bereich BDD würde die Zielstellung von DevOps der schnell verfügbaren, fehlerfreien Software weiter unterstützen und stärken.

5.3 Meinungsbild der Experten zu BDD

BDD ist eine Praktik, die das Requirements Management erleichtern würde, inwieweit dies abhängig von der Größe oder Komplexität von Projekten ist, ist allerdings fraglich. Die Meinung, dass der Nutzen von BDD expontentiell zur Projektgröße steigen könnte, da die Komplexität durch BDD reduziert wird, wurde durch die praxiserfahreren Experten vertreten. Daneben wurde durch die Experten auch infrage gestellt, ob komplexe Abhängigkeiten überhaupt mit BDD darstellbar sind. Besonderen Zuspruch fanden die standardisierten Vorlagen User-Stories und Szenarien, welche von den Interviewpartnern Katrin Heymann und Bianca Heinemann auch als größter Mehrwert von BDD angesehen wurden. Für dieses Meinungsbild, ist sicher die Erfahrung in der jeweiligen Rolle der IBM Mitarbeiter verantwortlich. Markus Holzinger, der in der technischen Rolle angestellt ist, beschrieb die Schablonen eher als Mittel zum Zweck der Automatisierung und sah in der Automatisierung den größten Mehrwert von BDD, da so Einsparungen generiert werden können.

Abbildung 22: Der Mehrwert durch die Einführung von BDD[119]

[119] Quelle: Eigene Erstellung

6 Zusammenfassung und Ergebnisse

Für das Verständnis dieser Arbeit wurden eingangs die Problemstellung der Schnellebigkeit der Softwareentwicklung und die daraus resultierenden Probleme erläutert. Da sich die spätere Umsetzung von BDD auf ein Projekt der IBM bezieht und im weiteren Verlauf auch Experten von IBM befragt wurden, wurde das Unternehmen vorgestellt.

Der Einstieg in die theoretischen Grundlagen fand durch die Abgrenzung von der klassischen zu agilen Softwareentwicklung statt. BDD wurde erläutert. Im Anschluss fand eine Einführung in die Grundlagen von DevOps statt. Der Zusammenhang von den beiden Praktiken wurde erläutert. Die gemeinsamen Ziele wie die Automatisierung von Tests und die Kommunikation zwischen den Beteiligten wurden in den Vordergrund gestellt.

In der praktischen Umsetzung von BDD wurde das Tool Cucumber, das zur Erstellung und Durchführung der Tests genutzt wurde, vorgestellt. Im Anschluss wurden die Anforderungen an das zu testende Tool erst erläutert, dann anhand der Beschreibungssprache Gherkin aufgenommen und in Features und Szenarien festgehalten. Es wurde gezeigt, wie durch die Programmierung von Step Definitions und einer Browserautomatisierung die Tests automatisch durchgeführt werden.

Nach der praktischen Umsetzung fand eine Befragung von drei IBM-Spezialisten aus dem Bereich DevOps nach einem festgelegten Interwieleitfaden statt. Die Kernaussagen der Interviews wurden dann genauer analysiert und die Meinungen der Experten zusammengefasst.

6.1 Beantwortung der Forschungsfragen

Die Beantwortung der eingangs gestellten Forschungsfragen ist mit den Ergebnissen dieser Ausarbeitung möglich.

1. Besteht Einklang zwischen den Behaviour-Driven Development-Ansätze und DevOps-Praktiken?

   Der Einklang des Ansatzes von BDD und der DevOps-Praktiken ist eindeutig gegeben. Die Ziele beider Praktiken überschneiden sich. Frühzeitig verfügbare möglichst fehlerfreie Software soll durch beide Vorgehen entwickelt werden. Kulturelle Aspekte wie Zusammenarbeit werden ebenfalls bei beiden favorisiert. DevOps bietet allerdings einen größeren Rahmen als BDD, so wäre die Intregration von BDD in diesen Rahmen sinnvoll.

2. Ist Behaviour-Driven Development mit Cucumber im Bereich Continuous Testing möglich?

Der Einsatz von BDD im Bereich Continuous Testing ist empfehlenswert, da beide Praktiken sich mit dem Test auseinandersetzen. Der Akzeptanztest wird bei der Durchführung im Rahmen von Continuous Testing automatisiert. Die Beschreibung der Abnahmekriterien in natürlicher Sprache ist an dieser Stelle ebenfalls vorgesehen.[120] Somit wäre BDD ein optimaler Ansatz für die Akzeptanztests im Rahmen von Continuous Testing.

3. Wie beurteilen projekterfahrene Experten Behaviour-Driven Development?

BDD ist aus der Sicht von Experten besonders für die Anforderungserhebung und für die Erstellung der Abnahmekriterien sinnvoll. Das Muster der User-Stories und Szenarien würde laut Meinung der Experten eine Vorlage bieten, durch die die Erhebung der Anforderungen erleichtert werden würde. Durch diese genaue Erhebung und Strukturierung der Anforderungen wird außerdem sichergestellt, dass keine Missverständnisse entstehen und somit eine höhere Kundenzufriedenheit erreicht wird.

Außerdem ist BDD im Rahmen von DevOps sinnvoll, da es die Entwicklung unterstützt und durch weitere Automatisierungsprozesse zu einer kürzeren Durchlaufzeit führen würde.

6.2 Fazit

Das Ziel dieser Arbeit war die Untersuchung von BDD im Rahmen von DevOps in dem Bereich Continuous Testing.

Es lässt sich nach der Bearbeitung festhalten, dass sich ein Zusammenhang durch überschneidene Zielstellungen ergibt.[121] Auch in der Umsetzung der Zielstellung sind Parallelen zwischen BDD und DevOps zu erkennen. Es wird die Zusammenarbeit aller Beteiligten und die damit verbundene Kommunikation in DevOps als auch in BDD favorisiert.

Als Ergebnis der durchgeführten Interviews bleibt festzuhalten, dass durch den Einsatz von BDD in der Praxis in unterschiedlichen Bereichen Einsparungen generiert werden können. Auch die Experten sind der Ansicht BDD in DevOps integrieren zu können.

[120] Vgl. Kapitel 3.5.2 Continuous Delivery

[121] Vgl. Kapitel 3.7 Gegenüberstellung von DevOps und BDD

Auffällig in den Interviews war, dass bei den projekterfahrenen Experten weder das Tool Cucumber noch die Vorgehensweise von BDD bekannt war, obwohl diese bereits seit 2006 publiziert wird. Die ursprüngliche Form TDD war den Experten allerdings bekannt.

6.3 Ausblick

Eine quantitative Überprüfung der Fragestellungen ist vor der Einführung in die Praxis notwendig. Die damit verbundene Untersuchung der Anwendung von BDD im größeren Rahmen. Dieser Rahmen sollte unterschiedliche Projekte umfassen, um die Anwendung von BDD bei höherer Komplexität und längerer Laufzeit zu überprüfen.

Auch eine tiefergehende Untersuchung des Bereichs von Requirements Engineering im Rahmen von BDD ist weiter zu untersuchen, um die Meinung der Experten in den Interviews zu unterstützen und den Mehrwert von BDD an diesem Punkt zu unterstreichen.

Außerdem sollte eine Untersuchung zur Eingliederung von BDD in den agilen Vorgehensrahmen SAFe durchgeführt werden. Diese würde dann die Grundlage bieten, um den Testprozess innerhalb von SAFe genauer zu definieren.

7 Anhang

Anhang I: Prinzipien von XP [122]

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 im Hinblick auf ihren persönlichen Vorteil arbeiten, sondern auch zukunftsorientiert und teamorientiert handeln.
Selbstähnlichkeit	Dieses Prinzip zielt auf die Wiederverwendung 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.

[122] Quelle: Eigene Erstellung in Anlehnung an Wollf und Bleek (2011), S. 151 ff.

Reflektion	Die Beteiligten sollen sich regelmäßig Gedanken über aktuell und zukünftig geleistete Arbeit machen. Umgesetzt wird dieses Prinzip durch spezielle Teammeetings.
Fluss	Ein Fluss entsteht in der Entwicklung durch stetige Auslieferung der Software und parallele Durchführung von Entwicklungstätigkeiten.
Gelegenheit	Das Team soll lernen, durch Probleme Gelegenheiten zu nutzen und Neues zu lernen.
Redundanz	Die Dopplung von Kompetenzen ist mit diesem Prinzip abgedeckt, so wird sich bei der Paarprogrammierung auf zwei Programmierer mit derselben Fähigkeit verlassen.
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 [123]

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 Releases fertiggestellt.
Freiraum	Freiräume sind für Entwickler die Phasen, in denen sie sich weiterbilden sollen. Dies ist unbedingt notwendig, da die Technologie zu schnelllebig ist.

[123] Quelle: Eigene Erstellung in Anlehnung an Wollf und Bleek (2011), S. 153 ff.

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 [124]

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 Releases 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 eingeplanen 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 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.

[124] Quelle: Eigene Erstellung in Anlehnung an Wollf und Bleek (2011), S. 156 ff.

Gemeinsamer Quelltext	Jeder Entwickler kann den gesamten Programmcode verändern. Wenn das Team nicht gemeinsam arbeitet, entstehen 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.ne
            t/");

            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.ne
      t/");
      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.ne
        t/");

        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);
        });
});
```

Anhang V: Interview Leitfaden

Die Leitfäden werden im Folgenden mit Fragen und den zusammengefassten Antworten dargestellt. Die Interviews erfolgten per Telefon und es ergab sich immer eine lockere Gesprächsatmosphäre.

Gesprächspartner	**Termin**
Bianca Heinemann	09.02.2017, 13:00 – 14:00 Uhr

Einleitung:

1. Was sind Ihre Haupttätigkeiten aktuell?

   Ein Business Analyst setzt sich hauptsächlich mit der Analyse der User Stories sowie der Vorbereitung und Aufbereitung des Kundenwunsches auseinander. Die Überprüfung der Bearbeitungsmöglichkeiten der Anforderungen findet statt. Die Anforderungen werden an die Programmierer verständlich übergeben.

2. Über welche Skills und Erfahrungen im Bereich Akzeptanztest verfügen Sie?

   Die Abnahmekriterien werden durch Frau Heinemann geschrieben und anhand dieser die Akzeptanztests automatisiert. Die Schwierigkeit besteht darin, im Vorweg die Anforderungen präzise formuliert zu haben. Wie das Beispiel eines Autos, das Beleuchtung benötigt, aber ein Scheinwerfer in der Mitte nicht genügt, da zwei Scheinwerfer vorgeschrieben sind, verdeutlicht. Außerdem gibt es beim Auto unterschiedliche Abhängigkeiten, die bei der Konzipierung der Lichtanlange zu beachten sind (z.B. Nebelscheinwerfer nur wenn Abblendlicht angeschaltet ist). Genau diese Abhängigkeiten müssen auch alle in den Abnahmekriterien abgeprüft werden, um den Akzeptanztest präzise durchzuführen.

3. Haben Sie bereits Erfahrungen mit Behaviour-Driven Development?

   Es bestehen keine Erfahrungen im Bereicht Behaviour-Driven Development.

Hauptteil:

4. Ist die Durchführung von Akzeptanztests, wie in Behaviour-Driven Development beschrieben, Ihrer Meinung nach sinnvoll?

Durchaus sehr sinnvoll, besonders die Art und Weise der Aufnahme der Anforderungen innerhalb der Schablonen der User-Stories und Szenarien erscheint als sehr sinnvoll.

5. Welche Vorteile ergeben sich aus dieser speziellen Art der Durchführung von Tests und der Entwicklung Ihrer Meinung nach?

Die Anforderungsaufnahme wird strukturierter durchgeführt. Es entsteht eine Art Leitfaden, der Missverständnisse vermeiden hilft. Außerdem ist auch die Dokumentation sinnvoll, falls es im Nachhinein zu Auseinandersetzungen zwischen den Parteien kommt.

6. Welchen Mehrwert sehen Sie für die Praxis in diesen Tests? Erkennen Sie eine Verbindung zu DevOps – Continuous Testing?

Automatisierung bietet immer einen Vorteil und ist auch in DevOps eingegliedert, daher sollte eine Verbindung möglich sein.

7. Bestehen Erfahrungen mit dem Tool Cucumber?

Es besteht kein Wissen über das Tool.

8. Welche Vor- und Nachteile ergeben sich bei der Arbeit mit diesem Tool?

Es wäre noch zu untersuchen, welche Schnittstellen Cucumber bietet, um festzustellen, ob eine Anbindung an Planungstools wie beispielsweise Jira möglich sind.

9. Im Vorfeld habe ich Ihnen User-Stories und die dazugehörigen Szenarien für das Tool (Website) der DevOps Backbone Initiative gesendet, welcher Wert ergibt sich aus den formulierten User-Stories?

Die strukturierte Erhebung der Anforderungen bietet den größten Wert, allerdings wird in diesen Testfällen nie ein messbarer Wert wie beispielsweise Reaktionszeit ermittelt. Dies wäre noch einzugliedern.

Schluss:

10. Sie sind als Business Analyst bei IBM tätig. Würde sich ein Mehrwert für Ihre Rolle ergeben?

Die Schablone wäre tatsächlich der größte Mehrwert aktuell, dennoch ist es fraglich, inwieweit sich dieses Vorgehen für Großprojekte mit höherer Komplexität eignet.

11. Wie verläuft die Testautomatisierung im agilen Vorgehen innerhalb der IBM?

Es wird immer mehr automatisiert, das ist vor allem für die Zusammenarbeit auf internationaler Ebene sehr sinnvoll, da so Zeitverschiebungen oder kulturelle Aspekte an Bedeutung verlieren können.

Gesprächspartner	**Termin**
Katrin Heymann	10.02.2017, 10:15 – 11:00 Uhr

Einleitung:

1. Was sind Ihre Haupttätigkeiten aktuell?

Als Strategy Consultant geht es um die Analyse der Funktionen innerhalb des Unternehmens. Die Prozessdokumentation über Interviews gehört derzeit zu den Haupttätigkeiten, durch diese werden die angewendeten Methoden dann überarbeitet.

2. Über welche Skills und Erfahrungen im Bereich Akzeptanztest verfügen Sie?

Die Erfahrungen im Testen beziehen sich eher auf das Management des gesamten Prozesses als explizit auf den Akzeptanztest.

3. Haben Sie bereits Erfahrungen mit Behaviour-Driven Development?

Es bestehen keinerlei Erfahrung im Bereich Behaviour-Driven Development

Hauptteil:

4. Ist die Durchführung von Akzeptanztests, wie in Behaviour-Driven Development beschrieben, Ihrer Meinung nach sinnvoll?

Derzeit wird in der Praxis ein Abgleich mit dem Kunden gemacht, welche Anforderungen nun genau festgehalten wurden, dieser würde durch BDD vereinfacht stattfinden können, da dieselbe Sprache bereits garantiert ist. Dennoch ist die Rollenverteilung hierbei auch wichtig, es ist nicht möglich, einen Mitarbeiter mit einer anderen Denkweise wie beispielsweise einen Entwickler diesen Prozess mit dem Kunden durchführen zu lassen.

5. Welche Vorteile ergeben sich aus dieser speziellen Art der Durchführung von Tests und der Entwicklung Ihrer Meinung nach?

Es wird durch BDD eine Übersicht über das gesamte Projekt oder eher die gesamten Anforderungen gegeben, diese ist gut, um den Überblick im Projekt zu bewahren.

6. Welchen Mehrwert sehen Sie für die Praxis in diesen Tests? Erkennen Sie eine Verbindung zu DevOps – Continuous Testing?

Automatisierungen bilden meist einen großen Mehrwert für die Praxis.

7. Bestehen Erfahrungen mit dem Tool Cucumber?

Es bestehen keinerlei Erfahrungen mit dem Tool.

8. Welche Vor- und Nachteile ergeben sich bei der Arbeit mit diesem Tool?

Keine Angabe

9. Im Vorfeld habe ich Ihnen User-Stories und die dazugehörigen Szenarien für das Tool (Website) der DevOps Backbone Initiative gesendet, welcher Wert ergibt sich aus den formulierten User-Stories?

Die Tests geben einen guten Überblick über die Funktionen des Tools, dennoch ist immer zu bedenken, wer diese Anforderungen aufnimmt, damit diese so formuliert werden können. Die Fragestellung „Wer kommuniziert mit wem?“ ist dabei immer sehr wichtig und auch trotz derselben Sprachbasis zu bedenken.

Schluss:

10. Sie sind als Strategy Consultant bei IBM tätig. Würde sich ein Mehrwert für Ihre Rolle ergeben?

Der Mehrwert zeigt sich in der strukturellen Anforderungserhebung und dem Überblick der Anforderungen, da solche Gesamtübersicht für ein Projekt bei Gesprächen mit Unternehmensvorständen von Bedeutung ist.

11. Wie verläuft die Testautomatisierung im agilen Vorgehen innerhalb der IBM?

Die Testautomatisierung nimmt deutlich an Bedeutung zu, das kontinuierliche Testen wird in Form einer Customer Relationship Management Methode umgesetzt.

Gesprächspartner	**Termin**
Markus Holzinger	10.02.2017, 11:00 – 12:00 Uhr

Einleitung:

1. Was sind Ihre Haupttätigkeiten aktuell?

Derzeit wird sich vor allem innerhalb eines Projekts auf die Entwicklung der CLM Tools konzentriert. Hierbei geht es nicht nur um die technische Bereitstellung der Tools, sondern auch um die Abbildung der vorhandenen Prozesse innerhalb der Tools.

2. Über welche Skills und Erfahrungen im Bereich Akzeptanztest verfügen Sie?

Es wurden bereits unterschiedlichen Rollen im Bereich des Testens ausgeübt.

3. Haben Sie bereits Erfahrungen mit Behaviour-Driven Development?

Im Bereich BDD bestehen keine Erfahrungen.

Hauptteil:

4. Ist die Durchführung von Akzeptanztests, wie in Behaviour-Driven Development beschrieben, Ihrer Meinung nach sinnvoll?

Der Auftraggeber muss die Anforderungen vorgeben und das Projektteam muss nachweisen, dass genau diese Anforderungen vom Auftraggeber gestellt wurden, somit ist das Vorgehen von BDD mit den User-Stories und Szenarien sinnvoll.

5. Welche Vorteile ergeben sich aus dieser speziellen Art der Durchführung von Tests und der Entwicklung Ihrer Meinung nach?

Durch die Automatisierung von Tests wird der Aufwand reduziert und somit auch die Durchlaufgeschwindigkeit erhöht. Es kann außerdem durch die Standardisierung die Qualität der Anforderungserhebung steigen, damit ist auch die Reduzierung von Missverständnissen zwischen den Beteiligten eingeschlossen. Außerdem steigt an diesem Punkt dann auch die Kundenzufriedenheit.

6. Welchen Mehrwert sehen Sie für die Praxis in diesen Tests? Erkennen Sie eine Verbindung zu DevOps – Continuous Testing?

Die Entwicklung wird effizienter und schneller durch diese Form des Testens. Die Eingliederung in DevOps ist durch die Automatisierung problemlos möglich. Besonders für die App Entwicklung wäre dieses Vorgehen von Vorteil, da dies ein sehr schnelllebiger Geschäftsbereich ist.

7. Bestehen Erfahrungen mit dem Tool Cucumber?

Es bestehen keinerlei Erfahrungen mit dem Tool.

8. Welche Vor- und Nachteile ergeben sich bei der Arbeit mit diesem Tool?

Die Einbindung von Selenium ist äußerst sinnvoll und das Mapping, das Cucumber vollzieht ist, für jeden nachvollziehbar.

9. Im Vorfeld habe ich Ihnen User-Stories und die dazugehörigen Szenarien für das Tool (Website) der DevOps Backbone Initiative gesendet, welcher Wert ergibt sich aus den formulierten User-Stories?

Es bestehen strukturierte Anforderungen die automatisch überprüft werden können. Dies stellt einen Mehrwert für das Projekt dar.

Schluss:

10. Sie sind als IT-Architekt bei IBM tätig. Würde sich ein Mehrwert für Ihre Rolle ergeben?

Derzeit wird SAFe durchgeführt, innerhalb diesem gibt es keinerlei Auskünfte, wie das Testen stattfinden soll. Die Einführung von BDD in diesem Prozess würde einen großen Vorteil bieten.

11. Wie verläuft die Testautomatisierung im agilen Vorgehen innerhalb der IBM?

Keine Angabe

Literaturverzeichnis

Bass et. al. (2015)

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

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..

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.

Hüttermann (2012)

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

North (2006)

North, D.: Introducing BDD. Artikel aus Better Software Magazine von 03.2006.

Royce (1970)

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

Rupp und die Sophisten(2014)

Rupp, C., die Sophisten: Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil. 5.Aufl., 2014. München: Carl Hanser Verlag.

Solís und Wang (2011)

Solís, C., Wang, X.: A Study of the Characteristics of Behaviour Driven Development. Artikel aus 37 th Euromicro Conference on Software Engineering and Advanced Applications, IEEE Computer Society, S. 383-387. Von 2011.

Swartout (2014)

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

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.

Witte (2015)

Witte, F.: Testmanagement und Softwaretest: Theoretische Grundlagen und praktische Umsetzung. 1. Aufl., 2015. Wiesbaden: Springer Vieweg.

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.

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).

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.

Lean-Management

Lean-Management: Was ist Lean?. Zugriff am 12.02.2017,

von: http://www.lean-management.biz. Gespeichert unter: CD\Internetquellen\Lean-Management.

Lean-Management – Prinzipien

Lean-Management – Prinzipien: Lean Prinzipien. Zugriff am 12.02.2017

von: http://www.lean-management.biz/3.html. Gespeichert unter: CD\Internetquellen\Lean-Management_Prinzipien.

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).

Onepage Wasserfallmodell

Onepage Wasserfallmodell: Wasserfallmodell. Zugriff am 01.02.2017,

von: https://de.onpage.org/wiki/Wasserfallmodell. Gespeichert unter: CD\Internetquellen\Onepage_Wasserfallmodell

Onepage DevOps

Onepage: DevOps. Zugriff am 17.10.2016,

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

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

Scaled Agile Framework

Scaled Agile Framework: Whitepaper: SAFe 4.0 Introduction. Zugriff am 10.02.2017. von: http://www.scaledagileframework.com/videos-and-presentations/. Gespeichert unter: CD\Internetquellen\Scaled_Agile_Framework.

Selenium (2017)

Selenium: Introduction. Artikel vom 10.02.2017, Zugriff am 11.02.2017.

von: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp#introducing-selenium. Gespeichert unter: CD\Internetquellen\Selenium(2017).

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.

Xoomtrainings (2015)

Xoomtrainings: An Introduction to DevOps. Artikel vom 24.10.2015, Zugriff am 11.02.2017, von: http://www.xoomtrainings.com/blog/an-introduction-to-devops. Gespeichert unter: CD\Internetquellen\Xoomtrainings.

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