Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Fritz Oscar Stephan Berngruber

Integration von DevOps in den laufenden Projektbetrieb am Beispiel des delegs-Forschungsprojekts

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Fritz Oscar Stephan Berngruber

Integration von DevOps in den laufenden Projektbetrieb am Beispiel des delegs-Forschungsprojekts

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Dr. Carola Lilienthal

Eingereicht am: 16.05.2017

Fritz Oscar Stephan Berngruber

Thema der Arbeit
Integration von DevOps in den laufenden Projektbetrieb
am Beispiel des delegs-Forschungsprojekts

Stichworte
DevOps, Continuous Deployment, Time to Market, Container-Technologie, Infrastruktur

Kurzzusammenfassung
In dieser Arbeit wird am Beispiel eines kleinen Forschungsprojekts, in dem eine verteilte Webanwendung weiterentwickelt und gewartet wird, eine Lösungsstrategie zur Integration von DevOps in dessen laufenden Projektbetrieb vorgestellt. Hierzu wird eine komplette Analyse des Projekts vorgenommen, auf deren Basis eine umfassende Lösungsstrategie erstellt wird. Neben der Integration von DevOps stellt die Lösungsstrategie ein verteiltes System vor, mit dem ein bruchloses Deployment neuer Anwendungsversionen möglich wird.
Das Ergebnis dieser Arbeit lässt sich zwar nicht grundsätzlich verallgemeinern, beschreibt aber, wie enorm die Time to Market verkürzt werden kann und das trotz erheblicher Hürden im untersuchten Forschungsprojekt.

Fritz Oscar Stephan Berngruber

Title of the paper
The integration of DevOps into an ongoing project
using the research project delegs as an example

Keywords
DevOps, Continuous Deployment, Time to Market, Container Technology, Infrastructure

Abstract
Using a small research project as an example, which is maintaining and further developing a distributed web application, this thesis describes a solution strategy for integrating DevOps into an ongoing project. A complete analysis of the project, based on comprehensive solution strategy development, is done. Apart from describing the integration of DevOps, the solution strategy shows a concept for a distributed system which enables the seamless deployment of new application versions.
The results here cannot easily be generalized but it does show, despite the significant hurdles in the examined project, how much the time to market can be reduced.

Inhaltsverzeichnis

Danksagung

An dieser Stelle möchte ich allen Menschen herzlich danken, die mich während der Bearbeitung meiner Bachelorarbeit auf die eine oder andere Art unterstützt haben.

Mein Dank gilt meinem *Professor Dr. Stefan Sarstedt* für die Möglichkeit, bei ihm diese Arbeit zu schreiben und für die hervorragende Betreuung während der gesamten Zeit.

Ich danke den Geschäftsführern der Firma *Workplace Solutions GmbH (WPS)*
Dr. Carola Lilienthal und *Dr. Guido Gryczan* für die Möglichkeit,
diese Arbeit im delegs-Projekt zu schreiben,
Dr. Carola Lilienthal außerdem dafür, dass sie sich bereiterklärt hat,
Zweitgutachterin für diese Arbeit zu sein.

Mein Dank gilt auch allen Mitarbeitern des zentralen Hamburger Teams des delegs-Projekts, ehemaligen und gegenwärtigen. Dort speziell *Jörn Koch*, *Katrin Mrohs* und *Adrian Metzner* für die Unterstützung bei fachlichen und technischen Fragen.

Außerdem danke ich den folgenden Menschen, die für mich Texte Korrektur gelesen haben: *meiner Mutter*, *Jason Wilmans* und *Chrystal Krichel*.

Für interessante Anregungen, gute Gespräche fachlicher Natur und das Teilen von Erfahrungen danke ich den Mitgliedern der *SIG DEVOPS* der WPS,
speziell *Clemens Heppner*, *Marco Ballhausen*, *Johannes Bumüller*, *Florian Stosch* und *Jens Barthel* und außerdem *Malte Eckhoff*, Letzterer ist kein Mitglied der SIG.

Mein besonderer Dank gilt *meiner Partnerin Rike*, mit der ich seit Beginn dieser Arbeit glücklich zusammen bin und die mich während der gesamten Zeit sehr unterstützt hat.
Du bist toll.

Teil I

Einleitung

1 Über diese Arbeit

Spätestens seit dem Einsatz von Clouds in der Softwareentwicklung und angetrieben durch das Aufblühen der agilen Bewegung ist das Thema DevOps in den Fokus der IT-Welt gerückt. Als Vermittler zwischen der Softwareentwicklung und dem Deployment-Prozess greift DevOps ganzheitlich in diese beiden Prozesse ein und verändert sie nachhaltig. Die Verteilung von Aufgaben und Verantwortungen und die Zusammenarbeit von Entwicklern (*Devs*) und Operations-Team (*Ops*) werden teilreformiert. Die Grenzen zwischen den beiden Teams verwischen.
Durch den Einsatz von DevOps entstehen neue Anforderungen an die zu entwickelnde Software, ihre Architektur wird entsprechend angepasst. Ziel von DevOps ist es dabei, die Produktivität von Projektteams zu erhöhen und die *Time to Market* zu verringern.[1]

In dieser Arbeit wird eine ganzheitliche Lösungsstrategie vorgestellt, die das Softwareprojekt, das im Rahmen des laufenden Forschungsprojekts *delegs* durchgeführt wird, in ein DevOps getriebenes Projekt umwandelt. In dem Softwareprojekt wird die Anwendung *delegs-Editor* gewartet und weiterentwickelt. Beim delegs-Editor handelt es sich um eine verteilte Webanwendung, die es ermöglicht, kontrastive Lehrmaterialien speziell für die Vermittlung von Gebärdenschrift zu erstellen.

Die in dieser Arbeit vorgestellte umfangreiche Analyse dient als Basis für die Lösungsstrategie. Die Strategie greift dabei in alle Bereiche der Softwareentwicklung ein und sieht bei deren Umsetzung sowohl Optimierungen im Entwicklungsprozess als auch im Deployment-Prozess vor. Außerdem wird in dieser Arbeit aufgezeigt, welche Vorteile durch eine Umsetzung der Lösungsstrategie erreicht werden, speziell der erwartete Zeitgewinn wird aufgezeigt.

Das Softwareprojekt ist besonders deshalb interessant, weil es Teil eines kleinen Forschungsprojekts ist. Es hat eine Teamstärke von maximal acht technischen Mitarbeitern und es hat mit starken personellen Fluktuationen zu kämpfen, die dadurch bedingt sind, dass für die Arbeit am Projekt nur Teilzeitkräfte zur Verfügung stehen, die jederzeit ausfallen oder abgezogen werden können.

[1] *Time to Market* ist die Zeit, die vergeht zwischen dem Entwurf und dem Zurverfügungstellen eines neuen Features beim Endnutzer.

2 Ziel und Fragestellung

In diesem Kapitel werden die Fragestellung an diese Arbeit und das Ziel der Arbeit definiert. Außerdem wird das Thema eingegrenzt.

Die Fragestellung an diese Arbeit lautet:

> *Wie groß wäre der zu erwartende Zeitgewinn durch die Umsetzung einer Lösungsstrategie zur Umwandlung eines Softwareprojekts in ein DevOps getriebenes Projekt, wenn das Projekt über nur wenig Personal verfügt und wenn es zusätzlich mit starken personellen Fluktuationen zu kämpfen hat, die dadurch bedingt sind, dass für die Arbeit am Projekt nur Teilzeitkräfte zur Verfügung stehen, die jederzeit ausfallen können?* [2]

Aus der Fragestellung ergibt sich das folgende Ziel dieser Arbeit:

> *Es soll eine Lösungsstrategie für die Integration von DevOps in den laufenden Projektbetrieb des delegs-Projekts entwickelt werden, sodass Letzteres durch die Umsetzung der Lösungsstrategie zu einem DevOps getriebenen Projekt umgewandelt werden kann.*
> *Auf Basis der Lösungsstrategie und einer anfänglichen Umsetzung dieser soll dann die oben genannte Fragestellung beantwortet werden.*

2.1 Abgrenzung

Die verschiedenen Aspekte der Arbeit müssen sehr genau betrachtet werden, um das Thema zu durchdringen und der Fragestellung in ihrer Gesamtheit gerecht zu werden. Gegeben durch die Größe des DevOps-Themas kann im Rahmen dieser Arbeit nicht auf alle Themen und Teilthemen explizit eingegangen werden.

[2] *Wenig Personal* meint im Rahmen dieser Fragestellung weniger als acht technische Mitarbeiter.

Folgend eine Auflistung von Themen, die nicht vertieft werden:
IT-Sicherheit, Microservices, Monitoring und Logging, agile Vorgehensmodelle, GitFlow, detaillierte Inhalt von Test-Suites, Automatisierung von Performance- und GUI-Tests, das Eingehen auf Fehlerfälle, Provisionierung von Umgebungen, teilweise die Beschreibung von Tools und teilweise auch schon nur die Auswahl des richtigen Tools.

Im Verlaufe dieser Arbeit wird immer wieder direkt an betroffenen Stellen auf notwendige Abgrenzungen hingewiesen.

Der Autor dieser Arbeit empfiehlt dem Leser bei Interesse an den entsprechenden Themen die Lektüre der gängigen Fachliteratur.

3 Aufbau der Bachelorarbeit

In diesem Kapitel wird zunächst der Aufbau dieser Arbeit beschrieben. Es wird auf die Herangehensweise - *die Methodik* - eingegangen und erläutert, wie die Fragestellung dieser Arbeit bearbeitet wurde. Dann beschreibt der Autor, wie er bei der Erstellung dieser Arbeit vorgegangen ist und schließt das Kapitel mit einigen Anmerkungen.

3.1 Aufbau dieser Arbeit

Diese Arbeit gliedert sich in vier Teile.

In Teil I Einleitung wird auf die Fragestellung und das Ziel der Arbeit eingegangen. Es werden der Aufbau der Arbeit, die Methodik und die Herangehensweise an die Arbeit beschrieben und wie sich der Fokus im Laufe der Arbeit, auf Kosten der Umsetzung, stärker hin zur Erarbeitung des Konzept verschoben hat.

Teil II Fachlicher Rahmen gibt eine Einführung in das delegs-Forschungsprojekt und bringt dem Leser die Philosophie und die Schlüsselwerte von DevOps näher. Außerdem wird auf einige Mechanismen und Vorgehensweisen eingegangen, die eingesetzt werden können, um ein Projekt nach DevOps aufzuziehen.

In Teil III Analyse & Lösungsstrategie wird der Ist-Zustand des delegs-Editors, seine Entwicklung und sein Deployment beschrieben und ausgewertet. Es werden die Anforderungen an die Lösungsstrategie erhoben und die Lösungsstrategie selbst vorgestellt.

In Teil IV Auswertung, Ausblick, Fazit wird die Lösungsstrategie evaluiert und es wird ein Ausblick gegeben, wie die begonnen Arbeit fortgeführt werden könnte. Dort wird unter anderem auf die Umsetzung der Lösungsstrategie eingegangen.
Den Abschluss der Arbeit bildet das Fazit, in dem die erarbeiten Ergebnisse zusammengefasst und der ursprünglichen Fragestellung gegenübergestellt werden.

3.2 Methodik

Die Herangehensweise an diese Arbeit ist eine in der Softwareentwicklung typische Methodik. Es wird eine Analyse vorgenommen und auf deren Basis eine Lösungsstrategie entwickelt.

Als Fallbeispiel wird das Softwareprojekt des delegs-Projekts gewählt, das die delegs-Editor-Anwendung wartet und weiterentwickelt.

In der Analyse wird der Ist-Zustand vom Projekt fachlich und technisch beschrieben und danach ausgewertet. Es werden dabei sowohl die Bereiche Entwicklungsprozess und Wartung als auch der Deployment-Prozess genauer betrachtet. Außerdem wird eine Erhebung von Anforderungen an die Lösungsstrategie vorgenommen. Diese werden indirekt aus der Fragestellung bzw. konkret aus der DevOps-Philosophie abgeleitet.
Anhand der Auswertung der Analyse wird dann eine Lösungsstrategie entwickelt.
Zum Schluss wird noch die Lösungsstrategie ausgewertet und es wird der direkte Bezug zur Fragestellung hergestellt.

3.3 Vorgehen bei der Erstellung dieser Arbeit

In diesem Abschnitt beschreibt der Autor der Arbeit sein Vorgehen bei der Erstellung dieser Arbeit und wie unter anderem die Lösungsstrategie erarbeitet wurde.

Grundlagenliteratur für die hier vorliegende Arbeit ist das Buch *DevOps: A Software Architect's Perspective* von *L. Bass*, *I. Weber* und *L. Zhu*.[3] Aus der Lektüre dieses Buches und den dadurch angestoßenen Recherchen im Internet entstanden viele der Inhalte, die später in die Beschreibung von DevOps einfließen sollten.

In dieser Zeit der tieferen Auseinandersetzung mit dem Thema begann ich nach und nach, den Umfang des Arbeitsthemas zu erahnen. Meine Vermutung, eine komplette Umsetzung der Lösungsstrategie sei nicht im Rahmen meiner Arbeit zu bewerkstelligen, verhärtete sich. Daher fiel früh meine Entscheidung, die Umsetzung entgegen einer DevOps konformen Vorgehensweise mit der Erfüllung der Anforderung A01 - *Deployment wird bruchlos durchgeführt* zu beginnen, um sicherzustellen, dass zumindest diese Anforderung umgesetzt wird.[4] Den tatsächlichen Umfang der begonnenen Arbeit konnte ich zu diesem Zeitpunkt allerdings noch nicht einschätzen. Wäre das möglich gewesen, hätte ich die Ausrichtung der Arbeit noch anpassen können.

Ich führte zunächst Interviews mit dem Team des delegs-Projekts durch und begann dann mit der Analyse des Projekts. Dazu überarbeitete ich die inhaltliche Struktur des Ist-Zustands und konkretisierte diese.
Mein Vorgehen bei der Analyse war, wie auch in den anderen Teilen dieser Arbeit, ein agiles. Immer wieder entstanden beim Schreiben an einem Abschnitt Ideen zu anderen Themenberei-

[3] Das Buch ist im Literaturverzeichnis zu finden, siehe Bass u. a. (2015).
[4] Die Anforderung A01 wurde vom Projektleiter an diese Arbeit gestellt. Details dazu im Kapitel 7 Anforderungen.

chen der Arbeit. Aufgrund der Abhängigkeit mancher Bereiche zueinander entwickelte ich diese teilweise parallel.
Nachdem ich die Analyse zu mehr als zwei Dritteln erarbeitet hatte, nahm ich die Anforderungen an die Lösungsstrategie auf. Bei Betrachtung der Fragestellung und der daraus resultierenden Aufgabe, delegs in ein DevOps getriebenes Projekt umzuwandeln, entwickelte ich die Idee, Anforderungen aus DevOps und dessen Philosophie zu extrahieren.

Mit Beginn der Entwicklung der Lösungsstrategie wurde mir klar, dass es noch eine weitere Quelle für Anforderungen gab: die von mir gehaltenen Interviews. Daraufhin nahm ich die Interviews in die Arbeit auf, wertete sie noch einmal gezielter aus und extrahierte aus ihnen vorläufige Anforderungen. Diese wurden danach von mir evaluiert, mit bereits erhobenen Anforderungen abgeglichen und entweder in die Menge der konkreten Anforderungen aufgenommen oder verworfen.

Nun konnte ich beginnen, die Lösungsstrategie für das bruchfreie Deployment zu entwickeln. Dafür stellte ich, anders als anfangs geplant, die Entwicklung einer neuen Deployment-Pipeline vorerst zurück. Um das bruchfreie Deployment zu realisieren, entwickelte ich ein Konzept für eine neues verteiltes System. Dies erwies sich als nicht einfach. Anfangs war ich versucht, ein hundertprozentig bruchloses Deployment zu realisieren. Ich begriff aber, dass für den Kontext, in dem der delegs-Editor eingesetzt wird, ein solches System viel zu komplex werden würde. Es ging also darum, eine weniger komplexe Lösungsstrategie zu entwickeln.
Als ich die Ausarbeitung der Lösungsstrategie bereits zur Hälfte vorangetrieben hatte, wurde mir der volle Umfang des Themas dieser Arbeit bewusst. Aufgrund der fortgeschrittenen Zeit und der bereits investierten Arbeit hielt ich es aber nicht für sinnvoll, das Thema zu beschränken oder die Arbeit anders auszurichten.

Da es mir wichtig war, eine *in sich geschlossene Arbeit* abzuliefern, entschied ich mich, weiter an der Lösungsstrategie und ihrer Verschriftlichung zu arbeiten - auf Kosten der Umsetzung.
Die Einführung von GitFlow im technischen Team ist die einzige Umsetzung, die ich während der Bachelorarbeit vorgenommen habe. Diese haben das technische Team und ich gemeinsam durchgeführt.[5] Zeitlich gesehen hat dies parallel zur Entwicklung der Lösungsstrategie stattgefunden.
Eine Übersicht zur Umsetzung und wie diese vorgenommen werden kann, habe ich in einem ersten Ansatz im Ausblick beschrieben.

[5] Der Autor dieser Arbeit war zu diesem Zeitpunkt Mitglied des technischen Teams. Details zu den Teams des delegs-Projekts im Kapitel 4 Das delegs-Forschungsprojekt.

3.4 Anmerkungen des Autors

Dieser Abschnitt beinhaltet Anmerkungen zu den Themen *Fachbegriffe im IT-Bereich* und *Gleichstellung von Gender*.

Mehrdeutige Begriffe im Fachbereich IT

Da es im Fachbereich IT Begriffe gibt, die nicht eindeutig definiert sind, empfiehlt es sich, einen Blick in das Glossar zu werfen. Die dort aufgeführten Definitionen gelten für die gesamte Bachelorarbeit.

Der Einsatz englischer Wörter in der Fachsprache IT

In der Fachsprache des Fachbereichs IT kommen vermehrt englische Wörter zum Einsatz, wie beispielsweise bei der Benennung von Prozessen (*Deployment*, *Delivery*, etc.). Im Anhang a. Umgang mit englischen Fremdwörtern in der deutschen Rechtschreibung ist ein Artikel der Webseite des Duden zitiert. Er wird in dieser Arbeit als Referenz für den Umgang mit englischen Fremdwörtern genutzt. Der Artikel gibt allerdings keinen eindeutigen Umgang vor und so wird versucht, die am besten lesbare Variante für alle Vorkommnisse konstant einzusetzen, was natürlich auf der subjektiven Einschätzung des Autors beruht.

Gleichstellung von Gender

Der Autor weist explizit darauf hin, dass in dieser Arbeit aus Gründen der Lesbarkeit von der Verwendung einer gendergerechten Sprache abgesehen wird. Jede genannte Rolle, Arbeitsaufgabe oder Ähnliches ist ohne Beschränkung auf ein bestimmtes Gender zu verstehen und kann und sollte entsprechend besetzt werden.
Ein Team wird stark durch die Vielfältigkeit seiner Mitglieder.

Teil II

Fachlicher Rahmen

4 Das delegs-Forschungsprojekt

In diesem Kapitel wird das delegs-Forschungsprojekt, kurz *delegs-Projekt*, beschrieben. Die Projektpartner werden vorgestellt, das Betätigungsfeld des Projekts wird beschrieben, es wird kurz auf dessen Geschichte eingegangen. Der Hauptfokus liegt dabei auf dem Standort Hamburg, an dem auch die delegs-Editor-Anwendung entwickelt wird.

Das delegs-Projekt ist ein laufendes Forschungsprojekt, das von vier gleichgestellten Partnern gemeinsam durchgeführt wird. Die vier Partner sind die *Universität Hamburg*, die *Fortbildungsakademie der Wirtschaft (FAW)*, die Firma *Workplace Solutions GmbH (WPS)* und *Dr. Hans-Günther Ritz*, sie werden unten genauer beschrieben.

Das Projekt wird durch das *Bundesministerium für Arbeit und Soziales (BMAS)* gefördert und hauptsächlich am Universitätsstandort Hamburg (Standort Hamburg) und an diversen Standorten der FAW in Deutschland durchgeführt.

Ziel des Projekts ist es, die Kommunikationsmöglichkeiten gehörloser Menschen zu verbessern, indem ihnen Möglichkeiten geboten werden, die deutsche Schriftsprache zu erwerben. Sie sollen dadurch größere Chancen auf dem Arbeitsmarkt und im Arbeitsalltag erhalten. Für deutsche Gehörlose ist die deutsche Schriftsprache eine Fremdsprache. Stützen sie sich nur auf ihre Muttersprache, die deutsche Gebärdensprache, laufen sie Gefahr, "[...] ein immer schwerwiegenderes Defizit" aufzubauen, da die "berufliche Kommunikation [...] immer stärker auf Medien und Techniken [...]" basiert, die in Deutschland deutsche "Schriftsprachkompetenz voraussetzen." (Hänel-Faulhaber (2015), S. 2).
Um dieses Ziel zu erreichen, werden am Standort Hamburg kontrastive Lehrmaterialien in deutscher Gebärdensprache (DGS) und deutscher Schriftsprache unter der Verwendung von Gebärdenschrift erstellt.[6] Außerdem werden dort Weiterentwicklung und Wartung der Webanwendung, dem *delegs-Editor*, vorangetrieben. Die Anwendung ermöglicht das Erstellen von Teilen der oben genannten Lehrmaterialien. Daneben bietet der delegs-Editor eine Plattform zum Austausch von Materialien, die im Editor erstellt wurden. Die Entwicklung der Lehrmaterialien und des delegs-Editors finden beide am Standort Hamburg statt.
Ursprünglich ist die Entwicklung des delegs-Editors 2009 einem studentischen Projekt entsprun-

[6] Bei Gebärdenschrift handelt es sich um Zeichensysteme zur Verschriftlichung der zur Kommunikation eingesetzten Gebärdenschprache.

gen, das im Rahmen des Softwaretechnik-Seminars am Fachbereich Informatik der Universität Hamburg durchgeführt wurde. Die Entwicklung des Editors wurde dann 2010 in das vom *Europäischer Sozialfonds* geförderte *delegs-Projekt* integriert. 2014 ging das Forschungsprojekt in das heutige vom BMAS geförderte Projekt über.

Im delegs-Projekt werden auch Kurse für Gehörlose entworfen und angeboten, in denen diese sowohl die deutsche Schriftsprache als auch die deutsche Gebärdenschrift erlernen können. Auch Weiterbildungsmaßnahmen für Lehrer solcher Kurse werden angeboten. Kurse und Weiterbildungen finden sowohl am Standort Hamburg als auch an diversen Standorten der FAW statt.

Das delegs-Projekt betreibt eine eigene Webseite, über die unter anderem auf die vom Lehrteam erstellten Lehrmaterialien und auf den delegs-Editor zugegriffen werden kann. Außerdem befinden sich auf ihr Angebote der Kurse und Workshops, Informationen zur Gebärdenschrift, kleine Lernspiele und weitere Informationen zu den Themen *delegs-Projekt* und *Gebärdensprachen*. Die Website ist zu erreichen unter *delegs.de*, der delegs-Editor unter *delegs.de/delegseditor*.

Das Team des Forschungsprojekts setzt sich zusammen aus einem Lehrteam, das die fachliche Kompetenz stellt und einem Tech-Team, das die delegs-Editor-Anwendung wartet und weiterentwickelt. Zusammen bilden sie das zentrale, in Hamburg arbeitende Team. Außerhalb von Hamburg gibt es zusätzlich Dozenten der FAW, die an den diversen Standorten der FAW Schulungen in Gebärdenschrift und deutscher Schriftsprache anbieten.

Das Forschungsprojekt wird am Standort Hamburg auch zu Ausbildungszwecken genutzt. Ein Großteil der im Tech-Team arbeitenden Entwickler sind studentische Hilfskräfte, zeitweise arbeiten dort auch Praktikanten. Es wird immer wieder Studenten ermöglicht, ihre Abschlussarbeiten im delegs-Projekt zu schreiben. In eben diesem Kontext ist auch diese Arbeit entstanden.

Die vier Projektpartner

Zwischen den vier Partner im Forschungsprojekt delegs gibt es eine enge Kooperation, ansonsten arbeiten die Partner allerdings in ihren Bereichen autonom. Es gibt keine Projektleitung, die alle Partner führt, die Projektanteile werden von den Partnern selbst geleitet. An dieser Stelle wird kurz auf die einzelnen Partner eingegangen.

Fortbildungsakademie der Wirtschaft gGmbH (FAW)

Die FAW ist mit diversen Standorten in der Bundesrepublik Deutschland unter der Leitung von *Anja Englert* vertreten. Dozenten der FAW bieten dort Schulungen für Gehörlose an, in denen diesen die deutsche Schriftsprache und die deutsche Gebärdenschrift vermittelt werden soll. Der Großteil der Dozenten ist selbst auch gehörlos.
Die FAW ist der Hauptantragsteller für das aktuelle delegs-Projekt.

Universität Hamburg

Die Universität Hamburg ist mit der *Fakultät für Erziehungswissenschaft, Schulpädagogik, Sozialpädagogik, Behindertenpädagogik und Psychologie in Erziehung und Unterricht* im Projekt vertreten. Der Projektpart der Universität Hamburg mit dem dort arbeitenden *Pädagogischen Team* wird von *Prof. Dr. Barbara Hänel-Faulhaber* geleitet.

Am Standort Hamburg arbeitet das zentrale Hamburger Team, das *zentrale Team*, das sich aus einem *Lehrteam* und einem *Tech-Team* zusammensetzt. Im Folgenden eine kurze Übersicht über die beiden Teams.

Das **Lehrteam** ist Teil des pädagogischen Teams und setzt sich zusammen aus zwei Lehrern, die beide Lehrmaterialien entwickeln und Schulungen vorbereiten und durchführen. Es werden von ihnen Lehrmaterialien in Schriftform und auch in Form von Videos erstellt. Ein Teammitglied des Lehrteams ist selbst gehörlos, was dem Team unter anderem den Vorteil der Nähe zur Gehörlosengemeinschaft bringt.
Das Lehrteam arbeitet zusammen mit dem Tech-Team an der Weiterentwicklung des delegs-Editors. Es stellt fachliche Anforderungen an die Entwicklung des Editors und priorisiert diese. So steuert das Lehrteam die fachliche Weiterentwicklung des Editors mit, testet den Editor und gibt Fehler in der Anwendung an das Tech-Team weiter. In den Zuständigkeitsbereich des Lehrteams fallen auch das Verwalten und Pflegen der Inhalte der delegs-Projektwebseite.

Der Großteil der Entwickler im **Tech-Team** sind studentische Hilfskräfte. Es setzt sich zusammen aus einem festangestellten Mitarbeiter der WPS und vier Studenten aus den Fachbereichen der Informatik. Unter der Führung des technischen Projektleiters, *Herrn Jörn Koch*, arbeiten die Entwickler weitestgehend selbstständig am delegs-Editor. Sie kümmern sich um die Wartung, Weiterentwicklung und das Deployment der Anwendung sowie fast alle anfallenden administrativen Aufgaben. Der festangestellte Mitarbeiter arbeitet im Projekt als Vierfünftelkraft, die studentischen Hilfskräfte hingegen jeweils ein bis zweieinhalb Tage die Woche. Bei Veröffentlichung dieser Arbeit ist der Autor selbst seit knapp einem Jahr als studentische Hilfskraft Entwickler des Tech-Teams im delegs-Projekt.

Workplace Solutions GmbH (WPS)

Die WPS ist im Bereich der Softwareentwicklung in den verschiedensten Feldern tätig. Sie agiert sowohl beratend als auch aktiv als Entwickler von Software für diverse Kunden und in Zusammenarbeit mit diesen und entwickelt eigene Softwareprodukte. Ebenso veröffentlichen die WPS und ihre Mitarbeiter Fachliteratur und Fachbeiträge im IT-Bereich. In Zusammenarbeit mit der Universität Hamburg, aus der die Firma ursprünglich entstanden ist, unternimmt die WPS Forschungsprojekte, zu denen auch das delegs-Projekt gehört. Für den Projektpart der WPS hat *Dr. Guido Gryczan* die Projektleitung für das delegs-Projekt.[7] Geminsam mit *Prof. Dr.-Ing. Heinz Züllighoven* und *Dr. Carola Lilienthal* ist er Geschäftsführer der WPS. Außerdem ist er Professor für Softwaretechnik an der Universität Hamburg. Zusammen mit dem technischen Projektleiter, Jörn Koch, leitet er die Entwicklung der delegs-Editor-Anwendung.
Herr Jörn Koch ist festangestellter Mitarbeiter der WPS und führt und berät das zentrale Team, speziell das Tech-Team und deren Arbeit am delegs-Editor.

Dr. Hans-Günther Ritz

Dr. Hans-Günther Ritz hat im delegs-Projekt die Rolle eines sozialwissenschaftlicher Beraters. Er ist in alle im Projekt stattfinden Aktivitäten involviert und gestaltet das Projekt aktiv mit.

[7] Wird in dieser Arbeit vom *Projektleiter* gesprochen, bezieht sich dies immer auf den Projektleiter des Projektparts der WPS, Herrn Dr. Guido Gryczan.

5 Über DevOps und seine Philosophie

In diesem Kapitel wird ein Einblick in das Thema *DevOps* gegeben.
Das Kapitel stützt sich dabei zu großen Teilen auf das Buch *DevOps : a software architect's perspective* von Bass u. a. (2015).
Um den Umfang der Arbeit zu begrenzen, wird neben einer kurzen Einführung von DevOps nur auf einige Schlüsselthemen eingegangen. Diese sind die Arbeitsweise, das DevOps-Team mit seinen Rollen und weitere wichtige Aspekte. Außerdem werden die folgenden, für diese Arbeit wichtigen Themen erörtert: *Continuity*, *Deployment-Prozess und Pipeline*, *Cloud*, *die Integration von DevOps in den laufenden Projektbetrieb* und eine *Abwägung von Kosten und Nutzen*.

DevOps hat sich, aus der agilen Bewegung und später parallel zu ihr entwickelt. Es kommt selbst einer Bewegung gleich, ist dabei aber weniger genau definiert, als es im agilen Bereich wie beispielsweise bei *Scrum* der Fall ist.[8] Es gibt kein Rahmenwerk, das vorgegeben ist, um ein Projekt nach DevOps Manier durchzuführen. Hingegen sind von DevOps eine Menge von Praktiken zusammengestellt, die helfen sollen, das oberste Ziel von DevOps zu erreichen: eine Verkürzung der *Time to Market*.

DevOps greift in fast alle Bereiche der Softwareentwicklung ein, inklusive des Deployments. Die von DevOps zusammengestellten Praktiken sind nichts Neues in der IT-Welt. Daher gibt es vermutlich mehr Softwareprojekte, die geplant oder ungeplant DevOps-Praktiken integriert haben als solche, die vollständig DevOps-Projekte sind. Denn Letzteres braucht Analyse, Planung und Umsetzung. Das gezielte Integrieren von DevOps in ein laufendes Projekt bedeutet erneuten Aufwand.

5.1 Der Begriff *DevOps*

Um den Begriff *DevOps* leichter erklären zu können, wird an dieser Stelle zunächst kurz auf das *(Software-)Deployment* eingegangen. Der *Deployment-Prozess* ist Teil des Softwareentwicklungsprozesses und setzt an der Stelle ein, an der eine Änderung an der Codebasis

[8] *Scrum* ist ein agiles Vorgehensmodell, ein Rahmenwerk, das wenige, prägnante Regeln vorgibt wie ein Softwareprojekt durchgeführt werden soll.

beispielsweise in Form eines neuen Features durch Softwareentwickler fertiggestellt wurde. Das Deployment selbst ist das *Bereitstellen* dieser Änderung als eine neuen Softwareversion in Test- oder Produktivumgebungen.[9,10] Zum Deployment-Prozess gehören auch das Integrieren von Code in die Codebasis und in Teilen auch der Produktivbetrieb und die Wartung von bereitgestellter Software. Außerdem fallen in den Bereich des Deployments die Aktualisierung, Konfiguration und das Warten der unterliegenden Infrastruktur sowie Tracking und Versionieren von Änderungen, Überwachung (*Monitoring*) von laufenden Systemen, und Weiteres.[11] Der Deployment-Prozess weist seine eigene Komplexität und Tiefe auf und kann sehr zeitintensiv sein.

Der Begriff *DevOps* setzt sich zusammen aus den Wörtern *Development* und *Operations*. Wobei Development (*Dev*) alle Facetten der Softwareentwicklung und die daran beteiligten Arbeitskräfte einschließt. Die im Bereich Operations (*Ops*) arbeitenden Fachkräfte hingegen unterstützen den Entwicklungsprozess, das Bereitstellen, das Testen und den Betrieb von neuen Versionen und stellen und warten die dafür benötigten Hardware- und Software-Ressourcen. In ihren Aufgabenbereich fällt auch der Anwender-Support für produktiv eingesetzte Soft- und Hardware. Die Aufgaben von IT-Support, (System-)Administratoren und möglichen Subgruppen fasst DevOps alle unter der Rolle des *Operators* zusammen. Es wird dort keine Unterscheidung gemacht. Operators können sowohl dediziert für ein Projekt zuständig sein als auch für mehrere, bis hin zu allen Projekte eines Betriebs.

5.2 Arbeitsweise von DevOps

DevOps versucht die Aufgaben und Prozesse, die traditionell in *Development* und *Operations* aufgeteilt waren, so zusammenzubringen, dass eine engere Verzahnung zwischen ihnen entsteht. Grenzen verwischen und nun übernehmen auch Softwareentwickler Operationsaufgaben und sind für diese mitverantwortlich. Dadurch werden die Entwickler dazu animiert, schon bei der Aufnahme der Anforderungen Operationsthemen zu bedenken und einzubringen. Zusätzlich werden auch Operators aktiv an Teilen der Softwareentwicklung beteiligt. Sie werden schon zu Beginn eines Projekts in das Erarbeiten der Anforderungen involviert und können

[9] Wenn der Begriff *Version* in dieser Arbeit benutzt wird, ist immer eine Veränderung an der bestehenden Software gemeint und das unabhängig davon, ob der neue Entwicklungsstand, der aus den Veränderungen entstandenen ist, als eine neue Version gekennzeichnet wird.

[10] In dieser Arbeit wird teilweise vom 'Bereitstellen von Software' gesprochen, wenn *das Vornehmen des Deployments* bezeichnet werden soll. Im deutschsprachigen Projektalltag wird hier allerdings statt der Nutzung des Wortes '*Bereitstellen*', entweder in umgangssprachlichem *Denglisch* vom '*Deployen*' (von *deploy*, dt.: *verteilen*) oder vom '*Verteilen*' von Software gesprochen.

[11] In dieser Arbeit ist, wenn von *Hardware* gesprochen wird, immer auch virtualisierte Hardware gemeint. Sollte hingegen rein *physikalische Hardware* gemeint sein, wird diese explizit so benannt.

dadurch früh Einfluss auf die Architektur der Software und deren Entwicklungszyklus nehmen. Der Einfluss der Operators auf die Entwicklung führt dazu, dass ein reibungsloserer Deployment-Prozess möglich ist. Die Entlastung des Deployment-Prozesses durch DevOps führt auch zu einer Entlastung der Operators. DevOps erreicht dies durch eine weitest mögliche Automatisierung von Teilen der Arbeitsaufgaben der Operators und natürlich auch durch die oben genannte Umschichtung von Operatoraufgaben auf die Entwickler. Auch unterstreicht DevOps die Wichtigkeit von funktionierenden Operations und verstärkt die Kommunikation und Zusammenarbeit zwischen Entwickler und Operator. Es entsteht eine engere Rückkopplung zwischen Entwicklung und Deployment-Prozess. Auf der Export-Import Bank Konferenz 2014 in Washington, DC spricht Elon Musk darüber, wie seine Firma *Tesla* genau diese Rückkopplung bei der Produktion von Autos erreicht und wie er den Produktionsprozess verbessern konnte. Sein anschauliches Beispiel, lässt sich leicht auf die Softwareentwicklung übertragen. Nach Elon Musk ist es wichtig dass man

> "[...] die Herstellung [von Autos] nicht als langweiligen Prozess sieht, in dem Kopien [eines Ingenieursdesigns] hergestellt werden, sondern [man erkennt], dass das Herstellungssystem, dass das Auto herstellt, selbst eine sehr komplexe Maschine ist. Und genauso wie Innovation in das Design des Autos gebracht wird, kann und sollte man Innovation in die Maschine bringen, die die Maschine herstellt. Man kann so viele neue, coole Wege finden ein Auto herzustellen. Ich bin überzeugt von einer engen Rückkopplung zwischen der Entwicklung durch Ingenieure und der Produktion. Wenn die Produktion weit von der Entwicklung weg ist, verliert man diese Rückkopplung. Und so versteht jemand, der ein Auto auf eine bestimmte Art entworfen hat nicht, wie schwierig es ist, dieses bestimmte Design umzusetzen. Wenn aber das Stockwerk des Werks [, in dem das Auto hergestellt wird,] 50 Fuß[12] vom Schreibtisch [des Ingenieurs] entfernt ist, kann man einfach raus gehen [in das Werk] und es sehen. [...] Und sie [die Ingenieure] können einfach ins Gespräch kommen mit den Leuten aus dem Werk. Und genauso haben die Leute aus dem Werk großartige Ideen, wie man das Auto verbessern kann. Wenn diese [Leute aus dem Werk] zu weit weg sind, können sie das nicht mit den Ingenieuren kommunizieren, die es [das Auto] entworfen haben. Ich glaube, etwas das oft nicht beachtet wird ist, dass eine enge bidirektionale Rückkopplung zwischen den Ingenieuren und der Produktion sehr hilfreich ist, um ein Auto besser zu machen, effizienter zu werden und die [Produktions-]Kosten zu senken." (Musk Elon (2014))

[12] Maßeinheit für Länge: 1 Fuß entspricht 30,48 cm, 50 Fuß entsprechen 15,24 m

Elon Musk spricht über die Erkenntnis der Werte des Entwicklungs- und Produktionsprozesses. Er unterstreicht, wie wichtig es ist, dem Design und der Weiterentwicklung der Maschine, die das Produkt herstellt, die gleiche Aufmerksamkeit und Wertschätzung entgegen zu bringen wie auch dem Produkt selbst. Analog dazu kann man bei DevOps die Wertschätzung für den kompletten Softwareentwicklungsprozess und speziell die Deployment-Pipeline finden. DevOps definiert die Deployment-Pipeline als ein "eigenes Stück Softwareprodukt" , deren Design, (Weiter-)Entwicklung und Wartung denselben Qualitätsstandards unterliegt, wie auch das Softwareprodukt, deren Entwicklung sie unterstützt. Laut Musk wird aller Aufwand, der in die Maschine gesteckt wird, die das Produkt herstellt mit dem Ziel aufgebracht, ein besseres Produkt zu entwickeln und dabei effizienter zu sein, auch was die Produktionskosten betrifft. Die gleiche Zielsetzung verfolgt auch DevOps. Und wie auch Musk, setzt DevOps auf enge Zusammenarbeit und dadurch enge Rückkopplung zwischen Entwicklung und Deployment-Prozess.
DevOps beeinflusst die Entwicklung von Softwaresystemen in der Gesamtheit.

5.3 Das DevOps-Team

DevOps in die Entwicklung einer Software einfließen zu lassen, bedeutet anzuerkennen, dass Operators und der Deployment-Prozess von gleicher Wichtigkeit sind wie Entwickler und Entwicklungsprozess. Ein Umdenken für alle Beteiligten ist nötig: Weg von alten Ideen über begrenzte Fähigkeiten und Aufgabenbereiche von Rollen wie Softwareentwickler oder Administrator. Hin zu einer engeren Zusammenarbeit, einer geteilten Vision und gemeinsamen Zielen in einem Team aus Entwicklern und Operators. Hin zu *einem* Team, das zusammen die Verantwortung für das zu entwickelnde Endprodukt trägt.
Auf der einen Seite übernehmen Softwareentwickler partiell Verantwortung am Deployment-Prozess, unter anderem für einen gewissen Zeitraum nachdem ein Release in den Produktivbetrieb eingebracht wurde. Sie nehmen so aktiv an der Überwachung und Fehlererkennung teil. Durch das Bereitstellen von Skripten, die Vorgänge während des Deployment-Prozesses automatisieren, wird der Deployment-Prozess Entwicklern zugänglicher gemacht. Die Skripte werden von Entwicklern und Operators zusammen entwickelt. Hierdurch kann das Verständnis der Skripte auf Entwicklerseite erhöht werden und die Expertise der Entwickler im Schreiben von Code in das Aufsetzen der Skripte mit einfließen. Treten während des Deployment-Prozesses Fehler auf, die traditionell von Operators behoben worden wären, werden diese nun zur Behandlung auch an die Entwickler weitergegeben und von Entwicklern und Operators zusammen behoben.

Auf der anderen Seite werden Operators schon während der frühen Phase der Softwareentwicklung in die Anforderungsermittlungen eingebunden. Sie sollen Anforderungen an die zu entwickelnde Software einbringen, die den späteren Deployment-Prozess erleichtern. Denn "Operations haben [beispielsweise] eine Menge von Anforderungen die das Logging und Monitoring betreffen. So sollten zum Beispiel Logging-Nachrichten für einen Operator verständlich und auswertbar sein." Operators sollten also für die Anforderungsanalyse als "[...] Stakeholder erster Klasse [...]" gesehen werden. (Bass u. a. (2015), S.5)
Auch ist es sinnvoll Operators zu schulen, sodass diese bestimmte Fehler der bereitzustellenden Software verstehen können und beispielsweise mit einer entsprechender Anpassung von Konfigurationen auf diese reagieren können.
Die oben genannte Neuverteilung von Aufgabenbereichen führt dazu, dass Fehler schneller erkannt und behoben werden können und ermöglicht manche Fehlkonfigurationen von vornherein zu unterbinden.

Rollen im DevOps-Team

DevOps definiert verschiedene Rollen, die es auszufüllen gilt. Die zu verteilenden Rollen und ihre Aufgaben sind im Folgenden aufgelistet und erläutert:

- **Teamleiter:**
  Der Teamleiter ist dafür verantwortlich, das Team zusammenzuhalten. Er bewahrt es vor äußeren und teaminternen Problemen, sodass dieses ohne Unterbrechungen arbeiten kann. Ähnlich wie beim *Scrum Master* aus dem Vorgehensmodell *Scrum* handelt es sich beim Teamleiter um eine Rolle, die keine Autorität über das Team ausübt. Das Team ist trotz Teamleiter selbstregulierend.

- **Teammitglied:**
  Teammitglieder sind alle am Prozess der Entwicklung und des Deployments beteiligten Personen.

- **Service-Owner:**
  Der Service-Owner hat eine ähnliche Rolle wie der *Product Owner* in *Scrum*. Er ist die Schnittstelle zwischen dem Team und der 'Außenwelt', so kommuniziert er mit Stakeholdern und Kunden. Er kennt das architektonische Bild des Gesamtsystems und priorisiert die Aufgaben für die kommenden Iterationen.

- **Reliability Engineer:**
  Der Reliability Engineer ist zuständig für das Überwachen der vom Team produktiv

bereitgestellten neuen Versionen der Software. Er ist der Verantwortliche, der kontaktiert wird, sollten Probleme während des Produktivbetriebs auftreten. Zu seinen Aufgabenbereichen gehört ebenso der partielle Produktivbetrieb.[13] Tritt ein Problem auf, wird der Reliability Engineer versuchen es zu beheben. Teilweise braucht er dazu die Hilfe seines Teams. Ein Reliability Engineer hat ausgezeichnete Kenntnisse über die betreute Software haben und ist zu jederzeit abrufbar.

- **Gatekeeper:**
  Wenn Änderungen an einer Software produktiv gehen sollen, muss diese durch alle Schritte der Deployment-Pipeline wandern.[14] Die Aufgabe des Gatekeepers ist es zu entscheiden, ob die Software in den nächsten Schritt der Pipeline übergeben werden kann.
  Gatekeeper ist eine Rolle, die gut (teil-)automatisiert werden kann, so zum Beispiel in einer Pipeline, in der automatisierte Schritte hintereinander durchgeführt werden.

- **DevOps Engineer:**
  Der DevOps-Engineer ist zuständig für die Auswahl und Zusammenstellung von Configuration Management-Tools und Deployment-Tools. Er weiß, wie diese konfiguriert und bedient werden.

5.4 Automatisierung, Monitoring & Wiederholbarkeit

Automatisierung ermöglicht es, Prozesse, also definierte Folgen von Arbeitsschritten, durchzuführen, ohne dass ein Eingreifen von menschlichen Akteuren nötig ist. DevOps sieht einen hohen Grad an Automatisierung vor und verspricht sich diverse Vorteile davon.
Aufgrund der immer größer werdenden Komplexität eingesetzter Technologien für den Softwareentwicklungsprozess und den Deployment-Prozess setzt DevOps auf Automatisierung, beispielsweise durch den Einsatz von Skripten. Schon an Stellen, an denen nur ein Befehl in eine Konsole eingegeben werden muss, fordert DevOps den Einsatz von Skripten. So soll die nötige Expertise zur Bedienung von Infrastruktur verringert werden. Dadurch kann das Operations-Team entlastet werden, da Entwicklern die Bedienung der Tools durch Skripte vereinfacht wird. Für Firmen ist dieser Ansatz zur Automatisierung höchst interessant, da es an Personal für die Stelle des Operators mit entsprechender Expertise mangelt (siehe hierzu Bass u. a. (2015), S. 10-11).

[13] Genaueres zum *partiellen Produktivbetrieb* im Unterabschnitt 5.7.1 Deployment-Pipeline.
[14] Genaueres zur *Deployment-Pipeline* im Unterabschnitt 5.7.1 Deployment-Pipeline.

Monitoring ist ein wichtiger Aspekt, der einen höheren Grad an Automatisierung ermöglicht. Das technisch unterstützte Überwachen von Prozessen in Software, Hardware, Infrastruktur und Tools liefert Rückmeldungen über ausgewählte Ereignisse. Monitoring ist deswegen für die Automatisierung wichtig, da auf erhaltene Werte aus überwachten Prozessen automatisiert reagiert werden kann. So wird unter anderem ermöglicht, Systeme zu entwickeln, in die geschlossene Rückkopplungsschleifen integriert sind.

Auch bringt Automatisierung den Vorteil der **Wiederholbarkeit**. Vorgänge werden immer exakt gleich durchgeführt. Ein menschlicher Akteur, der einen Prozess ausführt, hat die Möglichkeit, Zwischenschritte auszulassen, die ihm unnötig erscheinen oder Zwischenschritte zu übersehen. Entstehen hierdurch Fehler, kann ihre Behebung besondere Probleme bereiten, da der durchgeführte Prozess schwer nachvollziehbar ist. Bei eingesetzter Automatisierung hingegen kann im Fehlerfall die Verkettung von durchgeführten Aktionen problemlos nachvollzogen werden.
Um einen höheren Grad an Wiederholbarkeit zu erreichen, können alle Operationen, die ausgeführt werden und die nicht durch automatisierte Tools abgedeckt sind, in Skripte geschrieben werden. Skripte können in verschiedenen Versionen vorkommen. Diese sollten daher mit Kommentaren versehen werden, die deren Einsatzumstände genau beschreiben. Außerdem müssen die Parameter mit denen Skripte und Tools benutzt werden, genauestens für jede Ausführung festgehalten werden. So lässt sich die Wiederholbarkeit von automatisierten Prozessen gewährleisten.

Einen weiteren Vorteil bietet die Automatisierung für Situationen, in denen schnell gehandelt werden muss. Dabei können Fehler vermieden werden, die durch manuelle Ausführung unter Stress entstehen.

Die aufgeführten Gründe legen nahe bei der Anschaffung von Tools darauf zu achten, dass diese zu großen Teilen automatisierbar sind.

Alle oben genannten Punkte wirken sich vorteilhaft auf das Hauptziel von DevOps aus: die Verringerung der Time to Market. Es sollte jedoch bedacht werden, dass an Stellen, an denen zu viel automatisiert wird, die Gefahr besteht, dass Prozesse zu starr werden und nicht mehr einfach angepasst werden können. Trotz der Vorteile eines hohen Automatisierungsgrades muss das richtige Maß an Automatisierung gewählt werden.

5.5 Infrastructure-as-Code

DevOps sieht für den Umgang mit Infrastruktur "*Infrastructure-as-Code*" vor, also die Nutzung und Entwicklung von Infrastruktur(code) unter Einhaltung der gleichen Qualitätsstandards,

die auch für den zu entwickelnden Softwarecode gelten. So unterliegen alle Skripte und Konfigurationsdateien den gleichen Code-Konventionen wie auch Softwarecode. Änderungen an der Infrastruktur werden wie auch bei Software-Code mit Hilfe von Versionsverwaltungssystemen registriert. Hierdurch werden die Änderungen nachvollziehbar, was die *Wiederholbarkeit* steigert und unter anderem hilfreich sein kann für das Beheben von Fehlern.
Infrastruktur sollte genau wie Code ausreichend und wenn möglich automatisiert getestet werden.
Es kann hilfreich sein, Drittanbieter-Software in eigenen Code einzubetten, um darin Fehler abzufangen und lesbarer durch Logging dokumentieren zu können.

5.6 Continuous *Everything*

Unter dem Begriff *Continuous Everything* wird in dieser Arbeit *Continuous Integration*, *Continuous Delivery* und *Continuous Deployment* zusammengefasst. In diesem Abschnitt wird auf diese im Einzelnen eingegangen.

5.6.1 Continuous Integration

Continuous Integration (CI) ist ein Prozess, der nach dem Einchecken (*Check-In*) von neu entwickeltem Code in ein Versionsverwaltungssystem automatisiert durchlaufen wird. Er soll verhindern, dass Entwickler Gefahr laufen, durch zu seltenes Einchecken von neu entwickeltem Code oder zu seltenem Zusammenführen von Branchs Integrationsprobleme zu bekommen. Der Prozess sieht vor, so häufig wie möglich durchgeführt zu werden, immer dann, wenn es durch ein Einchecken Veränderungen an der Codebasis gibt. Für die Durchführung ist ein CI-Server zuständig. Er erstellt, getriggert durch das Einchecken, einen neuen Build aus der veränderten Codebasis. Über diese lässt er Unit- und Integrationstests laufen. Abhängig von der Konfiguration können CI-Server über Softwaremetriken zusätzlich die Qualität des Codes und auf das Einhalten von Code-Konventionen testen.
Sollten die ausgeführten Tests in Fehler laufen oder die Qualität des Codes nicht den Richtlinien entsprechen, wird der Check-In abgelehnt und der Entwickler, der diesen vorgenommen hat, wird über das Problem informiert. Der CI-Server stellt auch entsprechende Informationen bereit, wenn der Check-In erfolgreich verlaufen ist.

5.6.2 Continuous Delivery & Continuous Deployment

Continuous Delivery beinhaltet den kompletten CI-Prozess, erweitert um einen zusätzlichen Schritt. Nach den CI-Schritten wird automatisiert sichergestellt, dass die neu entstandene

Version dafür bereit ist, im Produktivbetrieb eingesetzt zu werden. Der CI-Prozess wird um den Schritt *Akzeptanztest / Staging / Performancetests* erweitert. In diesem wird die neue Version unter anderem in einer der Produktivumgebung so nahe wie möglich nachempfundenen Testumgebung (*Staging-Umgebung*) auf Herz und Nieren getestet.[15]

Beim **Continuous Deployment** wird als weiterer Schritt die neue Version, die während des Continuous Delivery-Prozesses als produktiv-tauglich befunden wurde, automatisch in den Produktivbetrieb bereitgestellt. Ein Vorteil von Continuous Deployment ist, dass sich auf die Entwicklung konzentriert werden kann und sich nicht um den Bereitstellungsvorgang gekümmert werden muss.

Es sollte hier erwähnt werden, dass die Begriffe *Continuous Delivery* und *Continuous Deployment* im Fachbereich IT unterschiedlich belegt werden. Die oben beschriebene Interpretation ist nur eine von mehreren. So gibt es auch die Interpretation, dass die Begriffe "[...] *Continuous Deployment* und *Continuous Delivery* austauschbar benutzt werden können, da sie dasselbe meinen." (Lakuzz und Mohdit (2015)). In dieser Arbeit werden die beiden Begriffe jedoch differenziert eingesetzt. Das automatisierte Bereitstellen unterscheidet die beiden Prozesse.

5.7 Deployment

Wie anfänglich schon im Kapitel 5 beschrieben ist der Deployment-Prozess Teil des Softwareentwicklungsprozesses. In diesem Abschnitt wird nun die Deployment-Pipeline im Allgemeinen beschrieben, die sich aus dem Deployment-Prozess ableitet. Danach wird gezielt auf das Blue/-Green Deployment, einer speziellen Deployment-Art, eingegangen.

5.7.1 Deployment-Pipeline

Eine *Deployment-Pipeline* beschreibt den in Schritte eingeteilten Deployment-Prozess und die dabei eingesetzte Infrastruktur. Sie leitet sich aus dem Deployment-Prozess ab.
Die Pipeline führt eine neue Version der Codebasis durch ihre verschiedene Schritte. Innerhalb dieser Schritte wird die neue Version auf verschiedene Arten getestet, es wird mindestens ein neuer Build aus ihr erstellt und dieser wird am Ende in den Produktivbetrieb übergeben. Wie der Name schon sagt, ist die Aufgabe der Pipeline das Deployment. DevOps sieht eine hoch automatisierte Deployment-Pipeline vor.

Die im Folgenden beschriebene Deployment-Pipeline ist nur ein Beispiel dafür, welche Schritte während des Deployment-Prozesses durchlaufen und wie diese ausgestaltet werden können.

[15] Näheres zum Schritt *Akzeptanztest / Staging / Performancetests* im Unterabschnitt 5.7.1 Deployment-Pipeline.

Die einzelnen Schritte sind von Projekt zu Projekt individuell gestaltet. Der Deployment-Prozess ist abhängig von den Anforderungen an die zu bauende Software und auch von den Projekt- und Firmenstrukturen, in deren Kontext die Software entwickelt wird.
Bass u. a. (2015) schreiben: "Eine sachgemäße Deployment-Pipeline zu haben ist essentiell um Systeme schnell entwickeln und bereitstellen zu können. Die Pipeline hat mindestens fünf Hauptschritte—Vor-Commit-Phase, Build und Integrationstests, Akzeptanztests / Staging / Performance-Tests, [partieller] Produktivbetrieb und Freigabe in den normalen Produktivbetrieb." (S. 98) Jeder der genannten Schritte wird in der oben aufgelisteten Reihenfolge durchlaufen und hat jeweils seine eigene Umgebung und Konfiguration, mit der die Software getestet wird. Jeder erfolgreich durchlaufene Schritt steigert die Wahrscheinlichkeit, dass der Code einer neuen Version fehlerfrei genug ist, um in den Produktivbetrieb überzugehen. Natürlich gibt es aber keine Garantie dafür. Neben der Reife der Anwendung für den Produktivbetrieb steigt normalerweise auch der Zeitaufwand, der anfällt, um die einzelnen Schritte zu durchlaufen. So schreibt auch Fowler (2013): "Jede [weitere] Phase [einer Deployment-Pipeline] bringt eine stärkere Zuversicht [dass die Software fehlerfrei ist], normalerweise auf Kosten von zusätzlichem Zeitaufwand."

Im Folgenden wird auf die von Bass u. a. (2015) vorgeschlagenen fünf Hauptschritte der Pipeline eingegangen und beschrieben, wie diese im Idealfall durchlaufen werden.

1. Vor-Commit-Phase

In dieser Phase wird noch in der Entwicklungsumgebung operiert. Entwickler arbeiten an einer neuen Version einer Anwendung. Mit jedem Commit - dem Beitragen von Code in die Codebasis - werden idealerweise automatisch alle Unit-Tests durchgeführt oder um Zeit zu sparen nur eine Untermenge. Außerdem werden einige wenige Smoke-Tests durchgeführt und die Einhaltung von Code-Konventionen überprüft.[16] Sind die Tests erfolgreich durchlaufen worden, wird der Commit angenommen. Spätestens wenn die Entwickler eine zusammenhängende Einheit Code entwickelt haben (z.B. ein Feature oder ein Bugfix) und die Codebasis in einen neuen lauffähigen Zustand gebracht wurde, wird diese an die automatisierte Überprüfung und Bereitstellung übergeben. Dies kann beispielsweise passieren, indem ein Feature-Branch in den Entwicklungs-Branch eingeführt wird (*Merging*), wodurch automatisch der nächste Schritt der Deployment-Pipeline angestoßen wird (siehe dazu *2. Build und Integrationstest*).

[16] weiteres zu *Smoke-Tests* siehe im Glossar

Zur besseren Absicherung können die Entwickler vor dem Übergeben von Veränderung manuell alle Unit-Tests ausführen. Während der Entwicklung werden die Unit-Test um Tests erweitert, die die neu entwickelte Funktionalität abdecken. Die neuen Unit-Tests werden mit ausgeführt.

2. Build und Integrationstest

Zuständig für diesen Schritt ist ein Integrationsserver (*CI-Server*). Von ihm werden die Ressourcen zusammengestellt, von der die Anwendung abhängig ist: Die internen Ressourcen werden organisiert, wie zum Beispiel benötigte APIs und externen Ressourcen werden konfiguriert beispielsweise eine Anbindung an eine Datenbank. Die Codebasis wird zusammen mit den Ressourcen in eine ausführbare Anwendung gebaut und diese mit der entsprechenden Konfiguration versehen. Bei diesem Vorgang entsteht das Artefakt *Build*. Der Build sollte ab diesem Schritt nicht mehr verändert werden, außer durch den Einfluss von Konfigurationsparametern. Der CI-Server führt an dem Build Integrationstests durch, die das fehlerlose Zusammenspiel aller Komponenten der Anwendung testen. Es ist sinnvoll, schon hier in einer Umgebung zu testen, die der Produktivumgebung so stark wie möglich gleicht. Vom Build genutzte externe Services sollten in diesem Schritt als Mock aufgesetzt oder wie z.B. im Fall einer Datenbank möglich als Testversion der echten Services realisiert sein. Tests sollten unter keinen Umständen Ereignisse im Produktivbetrieb auslösen oder Produktivdaten beeinflussen. Statt Mocks oder Testversionen kann es auch produktiv laufende externe Services geben, die mit Testnachrichten umgehen können, mit denen interagiert wird. Diese Nachrichten werden gesondert behandelt. Entsprechende Services nennen Bass u. a. (2015) "*test-aware*" (dt.: sich den stattfindenden Tests bewusst sein). Wird mit test-aware Services gearbeitet, müssen entsprechende Nachrichten als Testnachrichten markiert werden.
In diesem Schritt ist es außerdem sinnvoll, alle Unit-Tests durchlaufen zu lassen.
Die Anzahl der Tests, die ausgeführt werden, sollte gut abgewogen sein. Zu viele Tests führen zu größerem zeitlichen Aufwand, zu wenig Tests erhöhen die Chance, dass Fehler nicht gefunden werden. Tests sollten mit Bedacht geschrieben werden und nicht mit dem Ziel, alle Fehler in der Software zu finden, da dies unmöglich ist. Es wird immer Fehler geben, die sich erst bei längerem Produktivbetrieb auftun.

3. Akzeptanztests / Staging / Performance-Tests

Beim *Staging* wird versucht eine (*Staging*-)Umgebung zu schaffen, die der Produktivumgebung so nahe wie möglich nachempfunden ist. In dieser Umgebung wird die Software testweise betrieben, ohne jedoch mit produktiv betriebenen Services zu interagieren (mit Ausnahmen von test-aware Services). Die Anwendung sollte in dieser Phase dieselbe Konfiguration haben, wie sie sie auch im (vorläufigen) Produktivbetrieb haben wird.
Es werden Akzeptanztests mit einer Auswahl von Nutzern durchgeführt, die die Anwendung wie vorgesehen (z.B. über die GUI) nutzen und Feedback über eventuelle Probleme geben. Das Testen kann entweder durch vorgeschriebene Testfälle gesteuert werden, die den Anwendern zum Durcharbeiten gegeben werden. Oder aber die Anwender testen die Anwendung eigenständig. Es können auch automatisierte Akzeptanztests eingesetzt werden, die das Testen durch menschliche Anwender so gut wie möglich nachempfinden. Durch den Einsatz von automatisierten Akzeptanztests, wird auch, wie bei manuell ausgeführten vorgeschriebenen Akzeptanztest, ein hoher Grad an Wiederholbarkeit erreicht.
In diesem Schritt werden auch nicht-funktionale Tests durchgeführt, die Performance, Sicherheit etc. testen.

4. Partieller Produktivbetrieb

Es gibt verschiedenen Arten des partiellen Produktivbetriebs. In diesem Schritt befindet sich die Anwendung bereits in der Umgebung, in der später auch der normale Produktivbetrieb abläuft. Im Folgenden werden verschiedene Möglichkeiten beschrieben, partiell mit neuen Versionen produktiv zu gehen.
Beta-Tests sind wohl die gängigsten und bekanntesten Tests im Produktivbetrieb, bei denen die Anwendung mit einem kleinen Benutzerkreis getestet wird. Da mehr Nutzer die Anwendung testen als in den Schritten zuvor, ist die Wahrscheinlichkeit groß, dass diese Fehler finden. Den Anwendern wird ermöglicht, auf einfache Art Fehler oder Verbesserungsvorschläge zurückzumelden. Durch langsames Vergrößern ders Nutzerkreises kann die Belastbarkeit der Anwendung getestet und überwacht werden.
Canary-Tests sind eine weitere Möglichkeit, die Anwendung vorab zu testen. Auch beim Canary-Testen wird einem eingeschränkten Nutzerkreis die Funktionalität der neuen Version zur Verfügung gestellt. Diese Art Tests kann beispielsweise in einer

Mikroservice-Architektur eingesetzt werden: Bei einem Service, von dem mehrere Kopien im Einsatz sind, werden einige der Kopien durch eine neue Version des Services ersetzt. Die Registries oder Load-Balancer leiten nun Anfragen teilweise auf die neuen Versionen um und so können die neuen Services im Produktivbetrieb getestet werden, während sie unter Beobachtung stehen. Bringt die neue Version die gewünschten Resultate und läuft nicht in Fehler hinein, können auch die restlichen alten Version des Services abgelöst werden. Treten andererseits Probleme auf, kann die neue Version ganz einfach wieder aus dem Betrieb gezogen werden und durch alte Versionen ersetzt werden.
A/B-Tests funktionieren ähnlich wie Canary-Tests. Auch hier werden zwei Versionen *A* und *B* beispielsweise eines Services produktiv gestellt. Ziel bei A/B-Tests ist es herauszufinden, welche Version des Services eine oder mehrere gewünschte Eigenschaften besser erfüllt als die andere. Registries oder Load-Balancer verteilen Anfragen von verschiedenen Anwendern auf die beiden Versionen, wobei sie Anfragen von demselben Anwender immer auf die gleiche Version leiten.

Die Nutzerkreise der oben genannten Testverfahren können sowohl zufällig als auch bewusst ausgewählt sein.
Es ist auch möglich, während des Produktivbetriebs sogenanntes ***Live Testing*** durchzuführen. Hierbei werden bewusst nicht-funktionale Fehler im Produktivsystem ausgelöst. Beispielsweise unterbricht man dafür die Kommunikationsverbindung zwischen zwei Services. Das System wird also während es produktiv läuft herausgefordert, mit Fehlern, wie unter anderem Ausfällen von in Abhängigkeit stehenden Services, zurechtzukommen. Live-Tests führen letztendlich zu mehr Robustheit, sollten allerdings nur sehr bewusst eingesetzt werden. Der Produktivbetrieb darf niemals in die Gefahr kommen, in zu großen Teilen oder komplett auszufallen oder in einen Datenverlust hineinzulaufen.

5. Normaler Produktivbetrieb

Während des normalen Produktivbetriebs läuft die Anwendung konstant in einer Version oder einer Komposition aus Versionen, wenn diese aus mehreren Komponenten (z.B. Microservices) besteht. Die Anwendung wird weiterhin überwacht, denn auch noch in diesem Schritt der Deployment-Pipeline können Fehler auftreten.
Auch während des normalen Produktivbetriebs kann das im Schritt *4. Partieller Produktivbetrieb* beschriebene *Live Testing* durchgeführt werden.

Die einzelnen Schritte der Deployment-Pipeline sind zeitaufwändig. Um den gesamten Prozess bei auftretenden Fehlern nicht unnötig zu verlängern, ist es sinnvoll, die einzelnen Schritte der Deployment-Pipeline so zu entwickeln, dass ein Wiedereinsteigen an dem Schritt möglich ist, in dem der Fehler aufgetreten ist. So wird der Zeitaufwand der schon erfolgreich durchlaufenen Schritte eingespart. Gleiches gilt für einzelne Schritte, die zeitlich sehr lange laufen. Es ist sinnvoll, solche Schritte in logisch zusammenhängende Einheiten zu unterteilen, sodass beim Auftreten von Fehlern nicht der gesamte Schritt erneut durchlaufen werden muss.

Zwischen den einzelnen Schritten ist es die Entscheidung des *Gatekeepers*, ob einer neue Version der Übergang in den nächsten Schritt gewährt wird. Die Rolle des Gatekeepers kann sowohl automatisiert als auch von einem Teammitglied besetzt sein. Beispielsweise kann es sinnvoll sein, die Entscheidung, ob eine neue Version in den Produktivbetrieb gehen darf, von einem menschlichen Gatekeeper treffen zu lassen.

5.7.2 Blue/Green Deployment

Es gibt diverse Deployment-Arten die es ermöglichen, einen Deployment-Prozess effektiv zu gestalten. Dieser muss natürlich für den Projektkontext und die zu entwickelnde Anwendung passend zusammengestellt werden. Zu den im DevOps-Kontext eingesetzten Deployment-Arten gehören unter anderem *Rolling Upgrade* oder *Blue/Green Deployment*. Auf das Blue/Green Deployment wird in diesem Unterabschnitt kurz eingegangen.[17]
Blue/Green Deployment ermöglicht es eine neue Softwareversion bereitzustellen, ohne dabei den Produktivbetrieb unterbrechen zu müssen.
Dazu existieren im Betrieb zwei Softwareversionen parallel. Eine von beiden, hier *Blue*, ist die aktuelle Softwareversion und bedient den Produktivbetrieb. Die andere, *Green*, ist die Vorgängerversion von Blue. Wird eine neue Version bereitgestellt, ersetzt diese die Vorgängerversion Green. Respektive wird die neue Softwareversion dadurch zur *Green* Version und übernimmt zugleich die Abarbeitung des Produktivbetriebs von der Blue Version. Die Blue Version bedient den Produktivbetrieb nicht weiter. Dieser Vorgang wiederholt sich mit jedem Deployment.
Das Umschalten des Produktivverkehrs übernimmt ein vorgeschalteter Load-Balancer.
Wie bereits beschrieben entsteht so die Möglichkeit, während des laufenden Produktivbetriebs eine neue Softwareversion bereitzustellen. Ein weiterer Vorteil den diese Deployment-Art bietet ist, dass sie ein einfaches Rollback ermöglicht: Sollte eine fehlerhafte Version bereit-

[17] Aufgrund des begrenzten Rahmens dieser Arbeit, kann nicht auf das Konzept hinter *Rolling Upgrade* eingegangen werden. Der interessierte Leser kann dazu allerdings in der gängigen Fachliteratur Informationen finden, z.B. im Buch *DevOps: A Software Architect's Perspective*, siehe Bass u. a. (2015).

gestellt worden sein, kann diese wieder aus dem Verkehr gezogen werden. Dazu leitet der Load-Balancer den Produktivbetrieb einfach wieder auf die entsprechende Vorgängerversion, die noch immer parallel bereitliegt.

5.8 Cloud

Der Begriff *Cloud* bezeichnet bereitgestellte virtuelle Computerressourcen. Nutzer einer Cloud brauchen keine eigenen leistungsfähigen Computer zu besitzen, sondern mieten bei Bedarf die nötige Hardware, in Form von bereitgestellten virtuellen Ressourcen.
Anbieter von Clouds sind beispielsweise *Microsoft*, *Amazon* oder *Google*. Sie betreiben die echte physikalische Hardware, die als Host für die Cloud dient. Bei der physikalischen Hardware einer Cloud handelt es sich oft um Rechenzentren, die an geografisch unterschiedlichen Standorten platziert sind.
Die Betreiber von Clouds stellen den Nutzern nur die tatsächlich genutzte Leistung in Rechnung. Dieses Konzept macht die Nutzung von Clouds für ihre Anwender attraktiv. Sie müssen die Hardware, die die benötigten Ressourcen betreibt, nicht selbst anschaffen, verwalten und warten. Lediglich für den Zugriff auf die Cloud muss ein Computer zur Verfügung stehen. Hier reicht aber schon ein einfacher Computer mit Internetanbindung.
Bei manchen Cloud-Konzepten muss der Nutzer die virtuelle Hardware nicht einmal selbst konfigurieren, denn dies wird als Service durch den Cloud Anbieter erledigt. Dem Nutzer der Cloud ist transparent von welcher physikalischen Maschine er gerade die Ressourcen zur Verfügung gestellt bekommt. Eine Ausnahme macht es, wenn der Nutzer die Transparenz explizit nicht möchte, da er zum Beispiel geringe Latenz benötigt und deshalb bestimmen möchte, von welchem geografischen Standort die physikalischen Hardware einer Cloud eingesetzt werden soll.
Gerät der Nutzer an die Auslastungsgrenze der angemieteten virtuellen Ressourcen, kann er weitere Ressourcen hinzu mieten. Entsprechend kann er Ressourcen wieder abgeben, sollte nur noch ein Teil der ursprünglich genutzten Ressourcen benötigt werden. Clouds bieten diverse Vorteile, einige davon sind *Skalierung (Elastizität)*, *Verfügbarkeit* und *Reduzierung von Latenz*. Die Cloud kann wirtschaftlich lukrativ sein. Bass u. a. (2015) schreiben hierzu: "Durch die Elastizität von Clouds, dem pay-as-you-go Modell und der einfachen Provisionierung von virtueller Hardware, ist die Einplanung von Kapazitäten eher Laufzeitüberwachung und automatisierte Skalierung, als die Planung des Erwerbs von [neuer] Hardware." (S.51)

5.8.1 Fallbeispiel: Kosten von Cloud vs. virtuellem Server

Ein einfaches Fallbeispiel soll eine Idee dafür geben, wie sich die Kosten für die Nutzung einer Cloud zu den Kosten für einen gemieteten virtuellen Server (VPS) verhalten. Im Verlauf dieser Arbeit wird darauf Bezug genommen. Das Fallbeispiel ist dabei stark vereinfacht und nicht verallgemeinerbar. Es hat nicht den Anspruch wissenschaftlich fundiert zu sein. Es werden die Preise von einem Produkt aus der Amazon Cloud *EC2* verglichen mit dem einem Angebot eines virtual Servers von der Firma *Host Europe GmbH*. Die Amazon Cloud und Host Europe gehören zu den teureren Anbietern in ihren Bereichen. Auf den zwei ausgewählten Produkten kann das im Analyseteil dieser Arbeit beschriebene System von delegs betrieben werden, ohne dass dabei die Leistung der gemieteten Ressourcen übermäßig hoch ausgelegt sind. Um den Vergleich noch mehr zu vereinfachen, wird nur auf die CPUs, den Arbeitsspeicher und den zur Verfügung gestellten nichtflüchtigen Speicher eingegangen.
Im Folgenden werden die zwei Produkte beschrieben:

- **Amazon Web Services, Inc., EC2: *m4.large***
  vCPU: 2x 2,3-GHz-Prozessoren Intel Xeon®
  RAM: 8 GB
  Nichtflüchtiger Speicher:
  Variable Nutzung des Services *Elastic Block Store (EBS)*, für dieses Beispiel wird das Produkt *Amazon EBS General Purpose SSD (gp2) volumes* genutzt
  Kosten für ein Jahr: 256 - 267 € Gesamtkosten[18]

- **Host Europe GmbH: *Starter Plus***
  vCPU: 2x (keine näheren Angaben)
  RAM: 4 GB (zugesichert), 8 GB (dynamisch)
  Nichtflüchtiger Speicher: 150 GB
  Kosten für ein Jahr: 180 €[19]

Alle Preisangaben sind gerundet. Für die Amazon-Produkte sind die Preise von Dollar nach Euro umgerechnet, mit einem Kurs von 1 Dollar = 0,9323 Euro.

[18] Zusammensetzung der Gesamtkosten bei Amazon:
- m4.large: 56 - 67 €, je nach Zahlungsart
- EBS: 200 € bei 150 GB genutztem Speicher (0,1109 € pro GB/Monat)

Die Preise stammen von Amazon Inc. (2017a) und Amazon Inc. (2017b).
[19] Der Preis für den Host Europe VSP stammen von Host Europe (2017).

Das Verhältnis gilt nur für eine spezielle Produktart aus der Amazon Cloud und von Host Europe und ist nicht repräsentativ für alle Angebote. In speziell diesem Fall sind die Kosten für den Betrieb in der Cloud allerdings mindestens 40% höher, als für den Betrieb auf einem gemieteten VPS.
In einem Artikel des Wirtschaftsmagazins *Forbes* von 2015 schreibt *Tom Gillis*:
"All die gegebenen Variablen mit einbezogen denke ich, dass die echten Kosten, die anfallen um Berechnungen in der Cloud vorzunehmen, ungefähr zweimal so hoch sind, wie ein vollständig ausgelastetes, selbstverwaltetes Rechenzentrum." (Gillis (2015))
Allgemeingültige Aussagen über die Kosten für die Nutzung der Cloud zu treffen ist schwierig. Es können sicherlich Anwendungsfälle gefunden werden, in denen ihre Nutzung günstiger ist. Gerade in Fällen, wo Serverkapazitäten nur sporadisch und für einen begrenzten Zeitraum gebraucht werden. Würde man dafür eine eigene Infrastruktur stellen oder ein Root-Server oder VPS mieten, währen die Kosten mit sehr großer Wahrscheinlichkeit höher, als wenn man dies in der Cloud berechnen würde.

5.9 Integration von DevOps in den laufenden Projektbetrieb

In diesem Abschnitt werden einige Anregungen gegeben, wie ein laufendes Softwareprojekt, das nicht DevOps konform durchgeführt wird, in ein DevOps getriebenes Projekt umgewandelt werden kann.
Die folgenden Änderungen am Projekt sollten im laufenden Projektbetrieb umsetzbar sein:

- **Anpassung der Teamstrukturen:**
  Sollte ein externes Team die Überführung nach DevOps vornehmen (hier: *Einführungsteam*), schlagen Bass u. a. (2015) vor, zwei Teams aufzubauen. Eines zuständig dafür, die in Entwicklung befindlichen Anwendungen so anzupassen, dass diese durch die neue Pipeline (siehe unten) geleitet werden können. Die Entwickler der Anwendungen sind Teil dieses ersten Teams und arbeiten mit an der Umstrukturierung.
  Das zweite Team ist zuständig für Entwicklung und das Management der Pipeline und die in der Pipeline eingesetzten Tools.
  Die zwei Teams können in kleineren Projekten oder Organisationen auch von den gleichen Mitarbeitern besetzt sein und so direkt als ein Team zusammengefasst werden.

  Unabhängig davon, ob ein Einführungsteam eingesetzt wird oder nicht, gelten hier die in Abschnitt 5.3 Das DevOps-Team aufgeführten Prinzipien und es werden die dort vorgestellten Rollen verteilt.

- **Aufnehmen von Anforderungen:**
  Es sollten nachträglich Anforderungen an den Deployment-Prozess und die Produktivphase aufgenommen werden. Die Anforderungen können dann ausgewertet und dort, wo es möglich ist umgesetzt werden.

- **Umstrukturierung des Deployment-Prozesses und der Pipeline:**
  Hierzu sollte zuerst der Deployment-Prozess des Ist-Zustands als Pipeline visualisiert werden. Dies hilft bei der Analyse der vorgenommenen Schritte. Abgeleitet von der Analyse kann dann eine neue effizientere Deployment-Pipeline entworfen werden.

  Bei Projekten mit mehreren Anwendungen ist es sinnvoll, nicht direkt alle Anwendungen über die neue Pipeline bereitzustellen. Es sollte zuerst mit ausgewählten Applikationen die neue Pipeline getestet und verfeinert werden.

- **Änderungen an der Systemarchitektur:**
  Änderungen an der Systemarchitektur müssen nicht immer notwendig sein, eventuell kann auf eine Container-Technologie aufgesetzt werden.

Im Allgemeinen sollten Umsetzungen immer schrittweise durchgeführt werden, sodass der Produktivbetrieb davon nicht negativ beeinflusst wird.

5.10 Abwägung von Kosten und Nutzen

Es muss sorgfältig untersucht werden, ob sich eine Einführung von DevOps in ein zu planendes oder laufendes Projekt oder eine Firma lohnt. Entstehender Nutzen und eingesparte Kosten müssen den durch die Einführung von DevOps anfallenden Aufwänden gegenüber gestellt werden. Eine gut durchgeführte Einführung von DevOps kann im Projektverlauf viel Geld einsparen, da qualitativer und effizienter gearbeitet werden kann. Im Folgenden wird auf einige Punkte eingegangen, die in Bezug auf die Kosten und den Nutzen zu bedenken sind.

Der Deployment-Prozess ist ein kritischer Teil der Softwareentwicklung. Es ist wichtig, dass dieser ohne Verzögerungen ablaufen kann und somit nicht an Kosten gespart wird. Bei einem fehlerhaften Deployment besteht die Gefahr, dass sehr hohe Kosten entstehen.[20] So können Tools, die sorgfältig ausgewählt und gekauft oder eigens für den Deployment-Prozess gebaut werden, eine lohnende Investition sein. Sie unterstützen und beschleunigen den Prozess und können in dessen Iterationen wiederholt eingesetzt werden. Aufwände entstehen für solche Tools, durch mögliche Lizenzkosten oder dadurch dass Experten sie installieren, konfigurieren

[20] Siehe dazu Bass u. a. (2015) auf den Seiten 8-9.

und warten müssen. Daher sollten diese möglichst automatisierbar sein, sodass dadurch Kosten für deren Anwendung eingespart werden können.

Auch das Umstellen eines Projektbetriebs verursacht Kosten. Hier muss überprüft werden, ob die Umstrukturierung einer Firma zu radikal ist und vielleicht vorerst nur ein zukünftiges Projekt als DevOps Projekt geplant wird. Ist erst einmal ein DevOps-Projekt erfolgreich durchgeführt, kann die gewonnen Expertise und eingesetzte Infrastruktur auch für andere Projekte eingesetzt werden und verursacht dort weniger Aufwand.

Interessant ist auch ein Blick auf die Personalkosten. Bass u. a. (2015) weisen darauf hin, dass Softwareingenieure im Schnitt 50% mehr verdienen als Operations-Personal. Es steigen die Kosten für Gehälter, da Aufgaben von Operators mit der Umstellung nach DevOps nun auch von Softwareentwicklern durchgeführt. Dies zeigt, wie wichtig Automatisierung ist, da diese unter anderem den Zeitaufwand minimiert, den Entwickler mit ursprünglichen Operations-Aufgaben verbringen.

Im Rahmen dieser Arbeit kann nur beispielhaft skizzieren werden, worauf bei der Planung eines DevOps getriebenen Projekts oder der Umwandlung eines bestehenden Projekts im Hinblick auf Kosten und Nutzen geachtet werden muss.

Teil III

Analyse & Lösungsstrategie

6 Ist-Zustand

In diesem Kapitel wird der Ist-Zustand des delegs-Projekts mit Fokus auf die Entwicklung des delegs-Editors beschrieben und analysiert. Auf ihm basierend wird eine Lösungsstrategie erarbeitet, mit der DevOps in das delegs-Projekt integriert werden kann.

Im ersten Abschnitt wird der fachliche Rahmen beschrieben, in dem das delegs-Projekt durchgeführt wird. Danach wird die technische Seite der delegs-Editor-Anwendung beleuchtet. Es folgen Abschnitte über den Kontext des delegs-Editors, über die Umgebungen, die für seine Entwicklung und den Betrieb genutzt werden und über die Hardware, die für seinen Betrieb eingesetzt wird. Die darauf folgenden Abschnitte beschreiben das zentrale Team und ihre Arbeit, die Arbeitsabläufe und den Deployment-Prozess durch den die delegs-Editor-Anwendung bereitgestellt wird. Bevor schließlich zu den beschriebenen Bereichen Auswertungen gemacht und Empfehlungen zu Veränderungen ausgesprochen werden, wird noch eine Zusammenfassung über die Interviews gegeben, die der Autor dieser Arbeit mit diversen Mitarbeitern des Projekts und deren Umfeld gemacht hat. Aus diesen werden mögliche Anforderungen an das Projekt extrahiert.

In diesem Kapitel wird nur auf die für den Bereich DevOps interessanten Aspekte des Projekts eingegangen. Zwar wird auf Abhängigkeiten von Drittanbieter-Software und eingesetzten Tools eingegangen, jedoch erscheint es nicht als zielführend, eine komplette Liste aller eingesetzten Technologien und Abhängigkeiten des delegs-Editors aufzuführen. Eine solche Auflistung ist im Anhang unter b. Technische Abhängigkeiten & eingesetzte Technologien im Ist-Zustand zu finden. Auch wird auf ein Beispiel verzichtet, das aufzeigt, wie der delegs-Editor fachlich eingesetzt wird, da diese Informationen für die Einführung von DevOps im Projekt uninteressant sind. Der interessierte Leser möge die Anwendung unter der Adresse ausprobieren:
http://www.delegs.de/delegseditor/

6.1 Fachliche Beschreibung des delegs-Editors

Im Folgenden wird kurz die Anwendung des delegs-Projekts, der *delegs-Editor*, mit seinen Funktionen und seinem fachlichen Hintergrund beschrieben. Hierdurch soll ein Verständnis für die Anwendung, ihren fachlichen Kontext und ihren Einsatz geschaffen werden.

Der delegs-Editor ist eine Webanwendung zum Erstellen von kontrastiven Lehrmaterialien. Die Lehrmaterialien können erstellt werden, um Schriftformen verschiedener Gebärdensprachen (z.B. deutsche *Gebärdenschrift*) und die Schriftformen natürlicher Sprachen (z.B. Deutsch) zu vermitteln. Weltweit existieren mehr als 100 Gebärdensprachen.[21] Der delegs-Editor unterstützt davon einige wenige, darunter die deutsche Gebärdensprachen am vollständigsten. Innerhalb der einzelnen Gebärdensprachen gibt es oft verschiedene Dialekte, wie es auch aus den natürlichen Lautsprachen bekannt ist. Diese werden in Teilen vom Editor unterstützt, der auch das nachträgliche Einpflegen von Gebärden durch Anwender erlaubt.

Der delegs-Editor besteht aus drei fachlichen Hauptkomponenten, nämlich dem *Dokumenteneditor*, dem *Gebärdeneditor* und der *Dokumentenverwaltung*, die im Folgenden beschrieben werden:

- Im **Dokumenteditor** können Dokumente erstellt und bearbeitet werden. Er ermöglicht das gängige Arbeiten an Layouts von Dokumenten, wie es auch in Editoren wie Microsofts *Word* oder dem LibreOffice *Writer* bekannt ist. So können zum Beispiel Schriftart, -größe, -farbe, Einrückungen usw. bearbeitet werden. Der delegs-Editor bietet dabei eine auf seinen fachlichen Kontext angepasste Besonderheit: Er ermöglicht es, Wörter aus einer Lautsprache zu schreiben, welche direkt in die Gebärdenschrift der entsprechenden Gebärdensprache übersetzt werden. Damit die semantisch richtige Übersetzung genutzt werden kann, bietet der Editor die verschiedenen Belegungen eines Worts, sowie eventuelle dialektale Abwandlungen zur Auswahl an.
  Eine weitere Funktion ist das Verlinken von im Internet bereitgestellten Videos und Bildern, um diese im Dokument anzuzeigen.

- Die fachliche Komponente **Gebärdeneditor** ermöglicht es, Gebärden zu bearbeiten, die in Schriftform hinterlegt sind und neue Gebärden in Schriftform anzulegen. Auch dort können Attribute der Schrift bearbeitet werden, wie Farbe, Größe, usw.

- Die fachliche Komponente **Dokument-Manager** bietet die Möglichkeit, Dokumente zu speichern, zu verwalten und zu teilen. Dazu können Benutzer, Räume und Ordner

[21] Als Einstieg in das Thema *Gebärdensprachen* (und ihre Anzahl und vielfältigen Ausprägungen) siehe Wikipedia (2017b)

angelegt werden. Räume sind beschränkbar auf den Zugriff durch bestimmte Benutzer und auch auf die Art des Zugriffs, lesend oder schreibend, kann gesetzt werden. Ordner bieten die Möglichkeit, innerhalb von Räumen eine Ordnung anzulegen. Sie können sowohl in Räumen als auch in anderen Ordnern angelegt werden. In diesen Ordnern können Dokumente abgelegt werden, die mit dem delegs-Editor erstellt wurden.

Der delegs-Editor unterstützt offiziell die Browser *Mozilla Firefox* und *Google Chrome* auf den Betriebssystemen Microsoft *Windows* und Apples *Mac OS*. Die meisten Anwender des delegs-Editors nutzen eine der daraus resultierenden vier Kombinationen. Da Muttersprachler der Gebärdensprachen eine Minderheit in der Gesellschaft sind, ist der Nutzerkreis des Editors relativ klein. Näheres zur Nutzung des delegs-Editors im Abschnitt (6.3 Kontext: Anwender, Stakeholder & Konkurrenz).

6.2 Technische Beschreibung des Systems

Nun zur technischen Beschreibung der delegs-Editor-Anwendung.
Die delegs-Editor-Anwendung ist ein verteiltes System, das sich in einen Server und multiple Web-Clients unterteilt. Eine Übersicht über das verteilte System ist in der Abbildung 6.1 Ist-Zustand des verteilten Systems: Client, Server und Nachbarsystem modelliert und wird im Folgenden textuell beschrieben.

Serverseitig wird die delegs-Editor-Anwendung durch einen *Apache Tomcat 8* Web-Server bereitgestellt. Dieser läuft auf einem, bei der Firma *ServerAnbieterA* gemieteten, virtualisierten Hardware-Server mit *Debian 8.7* Betriebssystem.[22,23] Der Hardware-Server stellt sowohl die Webseite von delegs und die delegs-Editor-Anwendung als auch eine Testversion des delegs-Editors bereit. Alle drei arbeiten jeweils auf einer eigenen MySQL-Datenbank, die auf dem Hardware-Server durch denselben MySQL-Server zur Verfügung gestellt werden.

Die Produktivversion des delegs-Edtiors wird dem Tomcat-Server als *delegseditor.war* Artefakt übergeben.[24] Er stellt sie unter der folgenden URL bereit:
http://delegs.de/delegseditor/
(Auf diese URL wird von nun an unter der Bezeichnung *Produktiv-URL* Bezug genommen.)

Der Hardware-Server agiert auch als Staging-Umgebung. Auf demselben Web-Server, der die Produktivversion von delegs bereitstellt, wird bei Bedarf eine Testversion des delegs-Editors

[22] Der Name des Anbieters wurde in dessen Interesse anonymisiert.
[23] Genaueres zum Hardware-Server im Abschnitt 6.5 Der gestellte Hardware-Server.
[24] **.war Artefakte* entstehen im delegs-Projekt als Endprodukt des Build-Prozesses. Genaueres hierzu im Unterabschnitt 6.2.3 Google Web Toolkit (GWT) und im Abschnitt 6.7 Deployment-Prozess (Ist-Zustand).

bereitgestellt.[25] Sie wird dem Web-Server als *delegseditortest.war* Artefakt übergeben und ist dann unter der folgenden URL erreichbar:
http://delegs.de/delegseditortest/
(Auf diese URL wird von nun an unter der Bezeichnung *Test-URL* Bezug genommen.)

Der Hardware-Server stellt außerdem noch die Webseite des delegs-Projekts als *WordPress 4.7.2* Anwendung bereit, erreichbar unter:
http://delegs.de/
Über die Webseite kann auch der delegs-Editor aufgerufen werden.
Die *delegs.de* Domain wird von einer Firma gestellt, die unabhängig von ServerAnbieterA ist.

Um als Anwender den delegs-Editor zu nutzen, reicht es, die oben angegebene URL mit einem Browser zu öffnen. Bei Aufruf wird die Client-Software in Form von JavaScript-Code vom Server an den Browser übertragen. Der Browser fungiert ab dann als Laufzeitumgebung für den delegs-Editor-Client und führt dessen JavaScript-Code aus. Die Kommunikation zwischen Server und Client findet größtenteils über GWT-RPC und zu einem kleinen Anteil über eine vom Server angebotene REST-API statt.[26] Jeder Aufruf einer delegs-Editor-URL startet eine neue Instanz des Editors und legt für diese auf dem Server eine neue Session an. Dies gilt auch für den Fall, dass die URL aus demselben Browser heraus mehrmals aufgerufen wird. So zum Beispiel über verschiedenen Tabs oder durch Neuladen der Webseite (*Refresh* der vom Browser geladenen Inhalte).

Es gibt nur ein Nachbarsystem, von dem der delegs-Editor abhängig ist, es ist die Website *signbank.org*. Sie stellt eine Datenbank von Gebärden verschiedener Sprachen in Schriftform zur Verfügung. Außerdem stellt sie auch die einzelnen Symbole der Gebärdenschriften. Über eine REST-Schnittstelle importiert die delegs-Editor-Anwendung alle Gebärden der vom Editor unterstützten Sprachen.
Es sind keine weiteren Details über das Nachbarsystem bekannt.

Der delegs-Editor selbst bietet auch eine REST-Schnittstelle an, über die Gebärden in Schriftform geladen werden können. Nachbarsysteme, die diese verwenden, werden in dieser Arbeit nicht weiter betrachtet. Die Schnittstelle wird von Veränderungen am System nicht beeinträchtigt.

[25] Wenn eine neue Version des delegs-Editors bereitgestellt werden soll, muss diese zuvor in der Staging-Umgebung getestet werden. Genaueres zur Testversion im Abschnitt 6.7 Deployment-Prozess (Ist-Zustand).
[26] Genaueres zur Kommunikation zwischen Server und Client im Unterabschnitt 6.2.3 Google Web Toolkit (GWT).

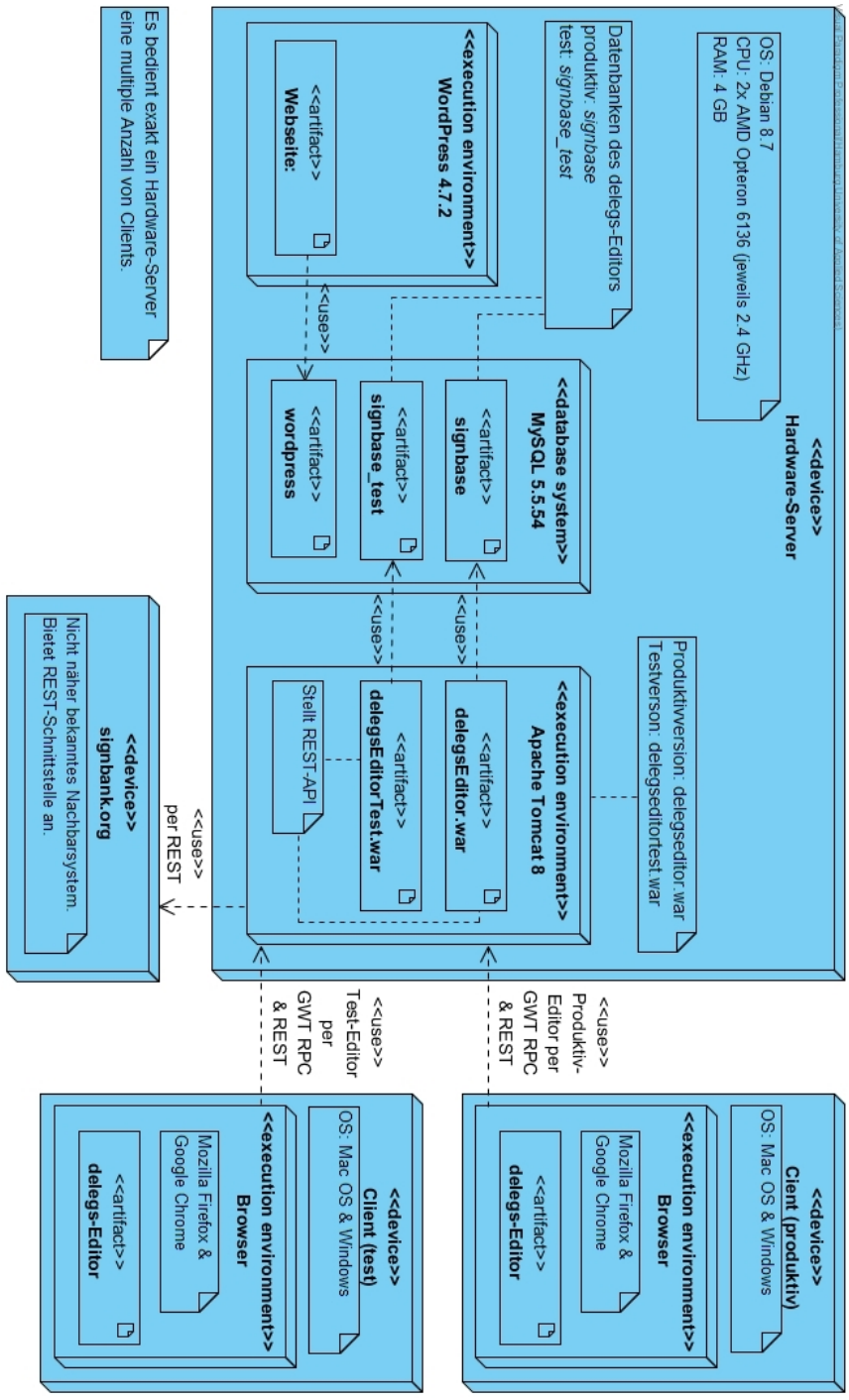

Abbildung 6.1: Ist-Zustand des verteilten Systems: Client, Server und Nachbarsystem

6.2.1 Die Codebasis des delegs-Editors

Die Codebasis des delegs-Editors wird im Projekt-Repository verwaltet. Es liegen dort Client- und Servercode zusammen. Ursprünglich ein Studentenprojekt, das später in das Forschungsprojekt *delegs* integriert wurde, ist die Codebasis kontinuierlich und teilweise chaotisch gewachsen. Sie ist unter anderem dadurch relativ komplex und nicht leicht zu durchdringen bzw. zu überblicken. Wird an ihr etwas verändert, ist es oft für die Entwickler nicht ersichtlich, ob ihre Änderungen nicht auch andere Stellen im Code beeinflussen. Letzteres liegt auch an der personellen Fluktuation[27]: Es gibt fast keine Entwickler, die lange genug im Projekt sind, um die Codebasis in ihrer Gesamtheit zu durchdringen und zu überblicken. So kommt es auch vor, dass die Umsetzung eines Features oder eines Bugfixes bei der Planung als relativ einfach eingestuft wird, der Aufwand aber bei der Umsetzung sehr stark anwächst.
Diese Tatsache muss hier erwähnt werden, da sie Auswirkung auf Entscheidungen hat, die im weiteren Verlauf dieser Arbeit getroffen werden. An den entsprechende Stellen, wird im Text darauf unter Nennung der *gewachsenen Codebasis* hingewiesen.

6.2.2 Architektur des delegs-Editors

Die Server- und die Client-Komponente des Systems sind nach dem *Werkzeug und Materialansatz (WAM-Ansatz)* und unter Einsatz des *Google Web Toolkits (GWT)* in *Java 1.7* gebaut.[28] Bei der Client-Anwendung handelt es sich um einen Rich Client, der bei Aufruf der URL des delegs-Editors als JavaScript-Anwendung in den Browser geladen wird. Im folgenden Unterabschnitt 6.2.3 Google Web Toolkit (GWT), wird auf das *Google Web Toolkit (GWT)* eingegangen.

Eine tiefergehende Beschreibung der Architektur des delegs-Editors erscheint während der Analyse für diese Arbeit nicht zielführend. Die Informationen aus diesem Abschnitt der Analyse sind für die Entwicklung einer Lösungsstrategie ausreichend. Aus diesem Grund wird an dieser Stelle auf eine umfassende Analyse der Architektur der verteilten Anwendung verzichtet.

[27] Auf die personelle Fluktuationen im Tech-Team wird im Abschnitt 6.6 Arbeit des zentralen delegs-Teams näher eingegangen.

[28] Der WAM-Ansatz beschreibt eine Referenzarchitektur mithilfe einer eigenen Mustersprache. Er wurde von *Reinhard Budde* und *Heinz Züllighoven* entwickelt. H. Züllighoven ist Geschäftsführer und Gründer der WPS. Für vertiefende Literatur zum WAM-Ansatz siehe "Object-Oriented Construction Handbook" Züllighoven (2005).

6.2.3 Google Web Toolkit (GWT)

Der delegs-Editor wird unter Einsatz vom GWT entwickelt. Da das GWT in Anwendungen, in denen es eingesetzt wird, bestimmte Vorgaben macht (u.a. architektonischer Art), ist es wichtig, es hier zu beschreiben und einen entsprechenden Überblick zu schaffen.

GWT beschreibt sich auf seiner Website als einen "Entwicklungs-Werkzeugsatz zum Bauen und Optimieren von komplexen [web-]browserbasierten Anwendungen." (GWT (2017))
GWT-Projekte werden in Java entwickelt. Wird ein GWT-Projekt gebaut, compiliert der GWT-Compiler den Java-Client-Code nach JavaScript, der Server-Code bleibt in Java. Das beim Bauen entstehende Artefakt ist ein Projekt, das nach der Java-Servlet-Specification aufgebaut ist. Es kann als **.war* Datei gepackt werden, um diese dann ohne weitere Änderungen auf einem Webserver im Internet bereitzustellen.[29] GWT ermöglicht es Rich Client Webanwendungen zu erstellen, deren Client-Code direkt beim ersten Aufrufen der URL der Webanwendung vom Server geladen wird. Auch ist es möglich Anwendungen so zu konstruieren, dass der Client-Code nachträglich nachgeladen werden kann. Um GWT-Anwendungen nutzen zu können, ist keine Installation von zusätzlicher Software nötig. Da alle gängigen Browser mit JavaScript-Interpretern ausgestattet sind, funktioniert die Ausführung von GWT-Anwendungen in den Browsern, ohne dass ein Plugin installiert werden muss. Die Browser fungieren dabei als Laufzeitumgebung für die Anwendung. GWT deckt fast alle Fälle ab, um GWT-Anwendungen mit den Eigenheiten aller gängiger Browser kompatibel zu halten.[30] Ein GWT-SDK liefert den nötigen Compiler und die nötigen APIs. Für die Entwicklung von GWT-Anwendungen steht ein Plugin für Eclipse zur Verfügung.

Die Struktur von GWT-Projekten stellt den Entwicklern drei Packages zur Verfügung: *Client*, *Server* und *Shared*. Wie zu erwarten wird Client-Code im *Client*-Package abgelegt und Server-Code im Package *Server*. In das Package *Shared* kommen alle gemeinsam genutzten Klassen hinein, also solche, deren Instanzen während der Kommunikation zwischen Client und Server ausgetauscht werden.
Die Kommunikation ist über *GWT Remote Procedure Calls* (*GWT-RPC*) und über HTTP-Anfragen, die JSON-Daten transportieren, realisiert. Mit GWT erstellte Anwendungen sind *AJAX*-Anwendungen, deren Kommunikation asynchron stattfindet. AJAX-Anwendungen erlauben es, dass bei Veränderungen am Inhalt einer Webseite, diese nicht komplett neu geladen werden muss, sondern ihr Inhalt On the Fly verändert werden kann. Arbeitet man mit der GWT-RPC Variante, macht GWT die Kommunikation zwischen Client und Server transparent und es

[29] *war* steht für *Web Archive*. Es handelt sich hierbei um ein Dateiformat, mit dem in Java implementierte Web-Anwendungen bereitgestellt werden können.
[30] Von GWT unterstützte Browser sind *Firefox*, *Internet Explorer* (8 bis 11), *Edge*, *Safari*, *Chromium*, *Google Chrome* und *Opera*.

somit den Entwicklern leicht. Diese müssen sich (fast) nicht mit der Kommunikation zwischen Server und Client auseinander setzen. GWT übernimmt den größten Teil und kümmert sich um das Senden, Empfangen und auch um die Serialisierung von Objekten, die von Client an Server oder andersherum übertragen werden sollen. Klassen, deren Objekte ausgetauscht werden sollen, müssen lediglich in den *Shared* Ordner abgelegt werden, Javas natives Interface zur Serialisierung *Serializable* implementieren und einen Standardkonstruktor aufweisen.
Nachdem ein GWT-Projekt gebaut und auf einem Server unter einer URL bereitgestellt wurde, kann ein aufrufender Browser den Client-Code laden.

Für das manuelle Testen und Debugging werden von GWT zwei verschiedene Modi angeboten, der *Classic Dev Mode (CDM)* und der *Super Dev Mode (SDM)*. Beide Modi starten lokal einen Server, auf dem die Anwendung über einen Browser erreichbar ist. Der CDM ermöglicht es, sich in einer veralteten Browser Version in den Programmfluss einzuhängen und in Eclipse die einzelnen Programmschritte im Java-Code per Debugging nachzuvollziehen. Für den CDM hatte Google ursprünglich Plugins für die gängigen Browser zur Verfügung gestellt. Die Browser unterstützen aber schon seit Längerem die für die Plugins benötigten Features nicht mehr, daher wurde bei GWT der CDM durch den SDM abgelöst. Ein Nachteil, den der neuere SDM gegenüber dem CDM hat, ist, dass mit diesem kein Debugging mehr direkt im Java-Code möglich ist. Will man unter Nutzung des SDM Debugging betreiben, ist dies nur noch mit den browser-integrierten Entwicklertools und auch nur noch auf dem automatisch von GWT generierten JavaScript-Code möglich. Dies macht das Debugging umständlicher, was der Grund dafür ist, dass während der Entwicklung des delegs-Editors beide Modi eingesetzt werden. Um den CDM einsetzen und lokal testen zu können, nutzt das Tech-Team die letzte, stark veraltete Version des Firefox (Version 24), die noch das GWT-Plugin unterstützt.[31]

6.3 Kontext: Anwender, Stakeholder & Konkurrenz

In diesem Abschnitt wird der Kontext des delegs-Editors in Bezug auf dessen Stakeholder und seine Konkurrenzsituation aufgeführt. Stakeholder inklusive der Anwender können beeinflussend auf die Entwicklung der Anwendung einwirken und könnten so auch die Entwicklung eines neuen Deployment-Konzepts beeinflussen.

Stakeholder, wie die vier Projektpartner und deren Mitarbeiter und das BMAS als Förderer des Projekts, wurden bereits im Kapitel 4 Das delegs-Forschungsprojekt beschrieben. Nun soll auch noch auf die Anwender des delegs-Editors eingegangen werden.

[31] Das letzte Firefox 24 Update wurde im September 2014 ausgeliefert, ab Oktober 2014 wird in GWT standardmäßig der Super Dev Mode eingesetzt.

Mit **Anwendern** der Software von außerhalb des zentralen Teams gibt es keinen direkten Kontakt.[32] Ausnahme macht der eine speziellen Power User, auf den im Abschnitt 6.7 Deployment-Prozess (Ist-Zustand) näher eingegangen wird. Im folgenden Unterabschnitt Nutzerstatistik wird statistisch auf die Nutzung des delegs-Editors und damit indirekt auf die Anwender eingegangen, über die es sonst kein Wissen gibt.

Es gibt keine **Konkurrenzsituation** für den delegs-Editor. Das delegs-Projekt ist ein durch das *Bundesministerium für Arbeit und Soziales* gefördertes Forschungsprojekt, das keinen Gewinn einfahren darf. Die im Projekt (weiter-)entwickelte und gewartete Software steht nicht in Konkurrenz zu anderen Produkten auf dem Markt und so gibt es auch keinen Konkurrenzkampf um die Nutzer der Software. Auch handelt es sich nicht um eine Software, die in einem sicherheitskritischen Kontext eingesetzt wird. Daher wird aus diesem Bereich des Kontextes von delegs kein Einfluss auf dessen Entwicklung ausgeübt.

6.3.1 Nutzerstatistik

An dieser Stelle wird kurz auf die Nutzerstatistik der URL *delegs.de* eingegangen. Die angegebenen Werte stammen von *Google Analytics* und beziehen sich auf den Zeitraum von einem Jahr, zwischen dem 01.04.2016 und dem 01.04.2017.[33]
Im diesem Zeitraum gab es 4875 Sitzungen auf *delegs.de* mit einer durchschnittlichen Sitzungsdauer von 02:43 Minuten und einer Absprungrate von 72,06%.[34]
Prozentual stammen die Sitzungen aus folgenden Ländern:

- 76,49% aus Deutschland
- 8,04% aus Brasilien
- 3,92% aus dem Vereinigten Königreich Großbritannien und Nordirland
- 2,73% aus Russland
- 2,17% aus den USA
- 6,65% restliche Welt

[32] Manche Personen aus dem Kreis der Stakeholder wenden den delegs-Editor selbst auch an, dies gilt z.B. für die Gruppe der Dozenten der FAW. Wenn diese Feedback geben, geschieht dies immer über das Lehrteam des zentralen Teams. Ausnahme macht hierbei der spezielle Power User.
[33] Quelle der Nutzerstatisitik ist Google Inc. (2017), alle Werte sind auf zwei Stellen hinter dem Komma gerundet.
[34] Absprungrate ist der prozentuale Anteil an Nutzern, die die Website besuchen ohne eine Interaktion vorzunehmen. Sitzungsdauer ist dort bei 0 Sekunden.

Von den 4875 Sitzungen sind 3950 auf *delegs.de/**delegseditor**/* (dem delegs-Editor) gestartet. Das entspricht 81,03% aller Sitzungen.
Bei den relativ wenigen Zugriffen auf den delegs-Editor sollte bedacht werden, dass auch die Entwickler durch die kontinuierliche Entwicklung des Editors Zugriffe verursachen. Diese wurden nicht aus den oben angegebenen Daten herausgefiltert.

6.4 Eingesetzte Umgebungen

Für die Entwicklung und Bereitstellung des delegs-Editors werden unterschiedliche Umgebungen eingesetzt. Diese sind neben der *Entwicklungsumgebung* die *Testumgebungen* und die *Produktivumgebung*. Sie werden in den folgenden Unterabschnitten beschrieben.

6.4.1 Entwicklungsumgebung

Die Entwicklungsumgebung wird für die Entwicklung am delegs-Editor eingesetzt. Sie setzt sich zusammen aus den Entwicklerlaptops unterschiedlicher Modelle und der darauf jeweils eingerichteten Software, inklusive des Betriebssystems. Es gibt zwar teilweise Richtlinien, welche Entwicklungswerkzeuge (Tools) für das Projekt auf den Entwicklungslaptops installiert sein sollen. Sie sind in einem Dokument festgehalten, das durch das Aufsetzen eines Entwicklerarbeitsplatzes führt. Ihr Einsatz ist allerdings nicht zwingend vorgegeben. Das Dokument deckt nicht vollständig alle eingesetzten Tools ab und gibt auch nicht vor, welches Betriebssystem eingesetzt werden soll. Daher gibt es keine, für alle Entwickler homogene Entwicklungsumgebung.
Den größten Unterschied macht beim Stand dieser Arbeit das Betriebssystem. In der gegenwärtigen Besetzung des Tech-Teams wird von allen Entwicklern *Microsoft Windows* eingesetzt. Es handelt sich allerdings um unterschiedliche Versionen von Windows (*8*, *8.1*, *10*).
Einheitlich geregelt sind nur die IDE und ihre Version (Eclipse), Versionsnummer des GWT-Plugins sowie die Hauptversionsnummer des eingesetzten Java-Compilers.

6.4.2 Testumgebungen

Für das Testen werden verschiedene Umgebungen eingesetzt, die folgend näher beschrieben werden:

- Mit der **Entwicklungsumgebung** werden automatisierte und manuelle Tests an der Anwendung durchgeführt. Das automatisierte Testen wird über Unit-Tests und eine von GWT gestellte Test-API abgehandelt. Für das lokale manuelle Testen nutzt das Tech-Team

zum größten Teil den *Classic Dev Mode (CDM)* und in seltenen Fällen den *Super Dev Mode (SDM)* von GWT.[35] Da sich die Modi deutlich voneinander unterscheiden, kann man hier von zwei verschiedenen Testumgebungen sprechen.
Manuelle und automatisierte Tests arbeiten beide auf einer lokal abgelegten Kopie der Produktivdatenbank. Ihr Stand spiegelt nicht immer den aktuellen Stand der Daten auf der Produktivdatenbank, jedoch zumeist den aktuellen Stand des produktiv eingesetzten Datenbankschemas wieder. Wenn Änderungen am delegs-Editor auch Änderungen am Schema nach sich ziehen und darüber noch nicht alle Entwickler informiert wurden bzw. diese noch nicht ihre lokale Datenbank entsprechend migriert haben, kann es passieren, dass das Schema nicht auf allen Entwicklerrechner aktuell ist.

- Die **Staging-Umgebung** wird eingesetzt, um Testversionen bereitzustellen und auf diesen sehr produktionsnahe Performance- und Akzeptanztests durchzuführen. Auf ihr werden zu testende Versionen als *.war Artefakt bereitgestellt. Diese können dort direkt über die Test-URL getestet werden, unter Einsatz der gängigen Client-Betriebsystem- und Browserkombinationen.
  Die Staging-Umgebung unterscheidet sich serverseitig von der Produktivumgebung nur durch die genutzte Testdatenbank und die von der Produktiv-URL abweichende Test-URL. Ansonsten wird dasselbe Setup eingesetzt wie auch für die Produktivversion: Hardware-Server, Betriebssystem und Tomcat- und MySQL-Server.[36]
  Als Clients dienen die Entwicklerlaptops des Tech-Teams und die dafür abgestellten Apple Computer mit entsprechendem *Mac-OS* Betriebssystem.

6.4.3 Produktivumgebung

Die Produktivumgebung setzt sich auf Serverseite aus dem Hardware-Server und dem darauf laufenden Tomcat-Server sowie dem MySQL-Server zusammen. Letzterer stellt eine Produktivdatenbank bereit.
Auf Clientseite besteht die Produktivumgebung offiziell aus allen Computern, auf denen die unterstützen Client-Betriebsystem- und Browserkombinationen eingesetzt werden. Inoffiziell besteht sie aus allen digitalen Geräten, auf denen ein Browser läuft, der über einen JavaScript-Interpreter verfügt, um die Client-Anwendung darstellen zu können. Der inoffizielle Teil geht nicht in diese Arbeit mit ein.[36]

[35] Details zu den Dev Modes im Unterabschnitt 6.2.3 Google Web Toolkit (GWT).
[36] Eine genaue technische Beschreibung der eingesetzten Soft- und Hardware ist im Abschnitt 6.2 Technische Beschreibung des Systems zu finden.

6.5 Der gestellte Hardware-Server

Die "*Hardware*" ,auf dem die delegs-Editor-Anwendung läuft, ist ein virtueller Server, der bei der Firma *ServerAnbieterA* als *managed Server* angemietet ist.[37] Er verfügt über zwei *AMD Opteron 6136 Prozessoren* mit jeweils einem 2.4 GHz Kern, 4 GB Arbeitsspeicher und 4 GB Swap Speicher. Auf dem Hardware-Server ist als Betriebssystem *Debian 8.7* installiert. *Managed Server* bedeutet, dass der Anbieter nicht nur den Hardware-Server stellt, sondern einen Teil der Konfiguration und Wartung des Betriebssystems und der Software-Server (z.B. Apache, Tomcat, WordPress) übernimmt. Im Fall delegs gehören dazu Updates, das Management von Nutzerrechten, Umgebungsvariablen und Weiteres. Die Firma ServerAnbieterA ist zuständig für das gesamte Setup des Servers. Ihre Aufgabe ist es auch, regelmäßig Backups des Servers anzulegen. Zusätzlich ist die Firma zuständig für das Monitoring der Verfügbarkeit und Sicherheit des Servers und sie verwaltet die für den Server nötigen Cron-Jobs.

Für das delegs-Team gibt es allerdings Schwierigkeiten mit dem Anbieter. Aufträge an die Firma werden zu langsam ausgeführt. Trotz der tendenziell langsamen Entwicklungsgeschwindigkeit am delegs-Editor geht die Abarbeitung von Aufträgen bei ServerAnbieterA immer noch zu langsam vonstatten.[38] Vertraglich sind keine Reaktionszeiten mit der Firma festgelegt.
Ein weiteres Problem für das Tech-Team liegt darin, dass ServerAnbieterA bei der Ausführung von Aufträgen teilweise unzuverlässig ist und diese fehlerbehaftet durchführt. Beispielsweise wurde ein "lange angekündigtes Update der OS-Version [...] bis heute [...] nur teilweise umgesetzt. Der Server braucht noch einen Neustart." Weitere Probleme sind bekannt, unter anderem dass einige Konfigurationen des Betriebssystems "[...] nicht logisch durchgeführt [worden sind]. Z.B. ist apt-get im Pfad eingetragen und sollte somit verwendbar sein, die Dependencies für apt-get sind aber nicht eingetragen."[39,40]

Das Tech-Team wird in administrativen Bereichen, die mit dem Managen des gemieteten Servers zu tun haben, gelegentlich durch das Administrationsteam (*Admins-Team*) der Firma WPS unterstützt. Tech-Team und Admins-Team haben auf dem Server nahezu vollständigen Administrationszugriff (sudo-Rechte) und verwalten sowohl Hard- als auch installierte Software-Server.
Aufgrund der Problematik mit ServerAnbieterA bearbeiten das Tech-Team und das Admins-

[37] Noch einmal der Hinweis: Wenn in dieser Arbeit von *Hardware* gesprochen wird, ist keineswegs nur *physikalische Hardware* gemeint. Genauso kann es sich um *virtualisierte Hardware* handeln. Sollte explizit nur eines von beiden gemeint sein, ist der Text entsprechend verfasst.
[38] Die tendenziell langsame Entwicklungsgeschwindigkeit ist bedingt durch die personelle Fluktuation im Tech-Team. Näheres dazu im Abschnitt 6.6 Arbeit des zentralen delegs-Teams
[39] Zitat *Adrian Metzner*, Entwickler des Tech-Teams, aus einer Gruppenkonversation der *SIG DEVOPS* der WPS zum Thema *ServerAnbieterA*, 04.03.2017.
[40] Beim Stand dieser Arbeit waren die genannten Probleme immer noch aktuell.

Team vieles, was eigentlich vom Anbieter erledigt werden sollte und beheben so unter anderem auch selbstständig Fehler, die durch das Handeln von ServerAnbieterA entstanden sind.
Allerdings gibt es auch Arbeiten, die nicht selbstständig von den Teams durchführbar sind. Zum Beispiel ist es nötig, dass Neustarts durch ServerAnbieterA durchgeführt werden, da sich andere Akteure im Fehlerfall selbst aus dem Server ausschließen könnten. Entsprechend können Aktionen, die einen Neustart benötigen, nicht vom Tech-Team (oder Admins-Team) getätigt werden. Auch können die für den Server notwendigen Cron-Jobs nicht selbst verwaltet werden.
Im Fall der Cron-Jobs besteht zusätzlich noch das Problem, dass deren Anzahl vertraglich limitiert sind, aber mehr Jobs benötigt werden.

6.6 Arbeit des zentralen delegs-Teams

Das zentrale Team setzt sich zusammen aus dem Lehrteam und dem Tech-Team. In diesem Abschnitt wird auf die Strukturen, die Rollenverteilung und die Arbeit der beiden Teams eingegangen. Zusätzlich werden die Arbeitsabläufe des Tech-Teams unter Einsatz des Versionsverwaltungssystems Git betrachtet.

Beide Teams arbeiten zusammen im selben Büro, wodurch ihnen eine enge Zusammenarbeit möglich ist und es sehr kurze Feedback-Wege gibt. Das Lehrteam ist konstant durch eine Vollzeit und eine Teilzeitkraft besetzt, letztere arbeitet zwanzig Stunden in der Woche. Der Anteil der Arbeit des Lehrteams am delegs-Editor ist für den Fokus dieser Arbeit nur bedingt interessant. Das Team ist nur wenig am Entwicklungsprozess beteiligt und meist nur in Bereichen, die nicht direkt von *DevOps* betroffen sind. Daher wird es hier nicht genauer beschrieben, sondern nur an den für die Arbeit interessanten Stellen.

Nun zum Tech-Team und dessen Arbeitsweise. Im Tech-Team gibt es starke *personelle Fluktuationen*, auf die in der weiteren Arbeit unter dieser Bezeichnung Bezug genommen wird. Einen großen Anteil an den Fluktuationen machen die studentischen Hilfskräfte aus, die dem Projekt als Entwickler von der WPS zur Verfügung gestellt werden. Sie arbeiten vertraglich bedingt in der Vorlesungszeit pro Kopf maximal 80 Stunden pro Monat im Projekt. Bei einer durchschnittlichen Wochenzahl von ~4,3 pro Monat, errechnet sich im Schnitt pro studentischer Entwicklerkraft eine maximale Arbeitszeit von ca. 18,61 Stunden in der Woche. Zu Klausurzeiten und besonderen Ereignissen (wie z.B. Abschlussarbeiten) arbeiten die studentischen Kräfte teilweise weniger Stunden oder auch gar nicht im Projekt. In der vorlesungsfreien Zeit erhöht

sich ihre Arbeitszeit, allerdings nicht auf Basis einer festen Regel.[41] Es gibt eine Tendenz bei den studentischen Kräften, feste Arbeitstage festzulegen, aber auch diese können sich bedingt durch Veranstaltungen im Studienbetrieb verschieben.
Der andere große Anteil an personeller Fluktuation entsteht dadurch, dass seitens der WPS jederzeit die Möglichkeit besteht, Kräfte, die dem delegs-Projekt zur Verfügung gestellt wurden, kurzfristig abzuziehen. Die Gründe dafür liegen bei der WPS. Betroffen davon sind alle im Tech-Team arbeitenden Kräfte, auch ein Entwickler, der als Dreiviertelkraft bei der WPS fest angestellt ist. Abgezogene Kräfte fallen für unterschiedlich lange Zeiträume für das Projekt aus, teilweise kommen sie aber auch gar nicht mehr zurück.

Der Entwicklungsprozess im Tech-Team ist agil und orientiert sich an Vorgehensmodellen wie *Scrum* und *Kanban*. Allerdings ist kein konkretes Vorgehensmodell oder Rahmenwerk eingesetzt, sondern eine Mischform.

Innerhalb des Teams sind die folgenden Rollen verteilt:
Der Projektleiter ist Stakeholder für den delegs-Editor und stellt teilweise Anforderungen an diesen.
Der technische Projektleiter steuert das Tech-Team und trifft Entscheidungen darüber, ob Anforderungen umgesetzt werden und wie diese priorisiert werden sollen. Gemeinsam mit dem Lehrteam, das selbst auch Stakeholder für den delegs-Editor ist und fachliche Anforderungen an diesen stellt, priorisiert er die fachlichen Anforderungen. Dabei hält er die Balance zwischen technischen und fachlichen Anforderungen und steuert so den Entwicklungsprozess. In die Steuerung des Entwicklungsprozesses wird nur sehr selten durch den Projektleiter, Herrn Dr. Gryczan, eingegriffen.
Bei technischen Fragen des Tech-Teams steht der technische Projektleiter diesem zur Seite und trifft Entscheidungen hinsichtlich der technischen Umsetzung. Für die studentischen Kräfte hat er auch eine ausbildende Funktion. Er steht dem Projekt einen Tag pro Woche zur Verfügung.

Im Tech-Team gibt es keine klar verteilten Rollen. Teammitglieder übernehmen jede Art von Aufgaben, egal ob es sich um solche aus dem Bereich *Entwicklung* oder aus dem Bereich *Operations* handelt. Das Team entwickelt, behebt Fehler, wartet die Anwendung und stellt sie bereit und ist zugleich zuständig für den Anwender-Support. Auch kümmert es sich um die eingesetzte Hard- und Software, sowohl für die Entwicklung und das Deployment als auch für den Test- und den Produktivbetrieb. Es gibt einige wenige administrative Aufgaben, die

[41] Zwei Semester haben zusammen eine zeitliche Länge von einem Jahr. Davon sind ca. 8,8 Monate Vorlesungszeit und entsprechend ca. 3,2 Monate vorlesungsfreie Zeit.

an die Administration der WPS oder den Anbieter des Hardware-Servers des delegs-Editors weitergereicht werden.[42]

Durch die personelle Fluktuation im Team gibt es beim Vorgehen innerhalb einer Arbeitswoche kaum einen festen Rhythmus. Nur montags finden als feste Termine ein einstündiges Meeting des zentralen Teams und ein zeitlich nicht begrenztes Meeting des Tech-Teams statt. Das Meeting des zentralen Teams dient dazu, alle Teammitglieder betreffend der allgemeinen projektpolitischen Lage, der Entwicklung des delegs-Editors, der Entwicklung von Lehrmaterialien und dem Verlauf von angebotenen Sprachkursen auf den gleichen Stand zu bringen. Das Meeting des Tech-Teams dient dazu, technische Fragen zur Weiterentwicklung und Wartung des delegs-Editors zu klären. Für den Rest des Montags steht es dem Tech-Team offen, zusammen im Mob am delegs-Editor zu arbeiten oder mit der normalen Entwicklung fortzufahren.[43]

Die Entwicklung an einer neuen Version des Editors nennt sich im Tech-Team *Iteration*. Iterationen tragen als Bezeichner immer die Nummer der sich in Entwicklung befindlichen Softwareversion. Anders als bei agilen Vorgehensmodellen üblich haben sie keine feste Dauer. Stattdessen wird in jedem Tech-Team-Meeting beschlossen, ob eine neue Produktiv- oder Testversion des delegs-Editors bereitgestellt werden soll.[44] Hauptgründe dafür sind die oben erwähnte personelle Fluktuation und die gewachsene Codebasis, die es schwierig machen, Schätzungen darüber abzugeben, wie lange die Umsetzung einzelner Anforderungen braucht. Trotzdem wird versucht, so häufig wie möglich eine neue Version bereitzustellen. Umzusetzende Anforderungen werden in der relativ einfachen Projektplanungssoftware (*XPlanner*) und parallel auf einer Art Kanban-Board, auf Karteikarten, mit Verweis auf den jeweiligen XPlanner-Eintrag festgehalten und organisiert. Es gibt keine klare Richtlinie, wie die Anforderungen formuliert werden. So gibt es auch keine Richtlinie, diese fachlich, wie zum Beispiel als User-Stories, zu formulieren.

Im XPlanner gibt es eine Sammlung von Anforderungen, die die Bezeichnung *Backlog* hat, die allerdings nicht wie ein Backlog à la Scrum eingesetzt wird. Die Sammlung beherbergt alle Anforderungen, die weder einer Iteration noch dem *Real Backlog* (siehe weiter unten) zugewiesen sind. Entsprechend enthält das Backlog sehr viele Anforderungen. Diese sind zwar teilweise priorisiert, aber ihre Priorisierung ist nicht up to date. Zusätzlich gibt es eine weitere Sammlung von Anforderungen, die die Bezeichnung *Real Backlog* hat. Sie ist schon eher gleichzusetzen mit einem Scrum-Backlog. In ihr sind die Anforderungen priorisiert und

[42] Näheres zum eingesetzten Hardware-Server ist unter 6.5 Der gestellte Hardware-Server zu finden.

[43] *Mob-Programmierung* ist eine Softwareentwicklungsmethodik ähnlich zu *Pair Programming*[45], bei der Software von einem Teil oder dem gesamten Team zusammen, im *Mob*, an einem Arbeitsplatz, sprich einem Computer, entwickelt wird.

[44] Näheres zur Bereitstellung der Anwendung und dem dazugehörigen Deployment-Prozess ist im Abschnitt 6.7 Deployment-Prozess (Ist-Zustand) zu finden.

werden bei der Iterationsplanung in die neue Iteration hineingezogen. Teilweise werden auch von anderer Stelle Anforderungen in die Iteration aufgenommen. So zum Beispiel aus den Meetings des zentralen Teams oder sie entstehen während des Entwicklungsprozesses.
Im XPlanner sind Iterationen unter ihrer Bezeichnung angelegt. Eingesetzt wird er fast ausschließlich vom Tech-Team, nur das macht darin die Einträge.

Im Tech-Team wird niemals alleine entwickelt. Entwicklungsarbeit findet immer im Pair statt. Sollte die Anzahl der Entwickler vor Ort ungerade sein, wird auch zu dritt an einem Arbeitsplatz entwickelt.[45]

Alle Mitglieder des Tech-Teams haben Zugriff auf die eingesetzten Systeme und so auch auf das Produktivsystem. Dies gilt für alle Bereiche, auch für die Zugangsdaten der Produktivdatenbank. Alle Zugangsdaten werden in einer verschlüsselten Datei, die in einem Websystem liegt, abrufbereit gehalten.

6.6.1 Arbeitsabläufe unter Einsatz von Git

An dieser Stelle wird kurz vorgestellt, wie der Workflow unter Einsatz des Versionsverwaltungssystems *Git* vonstatten geht. Der Workflow funktioniert sehr ähnlich dem von *Vincent Driessen* vorgestellten *GitFlow*.[46]

Zur Versionsverwaltung der Codebasis wird im delegs-Projekt *Git* eingesetzt. Der in Git verwaltete Master-Branch fungiert als *Development-Branch*. Er wird auf einem *sauberen Stand* gehalten, heißt, er hält immer eine Version der Codebasis bereit, die bei Ausführung der automatisierten Tests keine Fehler wirft. Alle Umsetzungen von Anforderungen oder behobenen Bugs, im restlichen Abschnitt als *Feature* bezeichnet, werden auf ihm festgehalten. Wenn Entwickler beginnen, an einem Feature zu arbeiten, erzeugen sie in Git immer einen neuen Branch (*Feature-Branch*), auf dem die Änderungen an der Codebasis durch die Entwickler festgehalten werden. Ein Feature-Branch bekommt als Bezeichner immer die Id des Features, die dieses im XPlanner zugeordnet bekommen hat.
Wird an einem neuen Tag die Arbeit an einem Feature wieder aufgenommen, wird zuerst ein *Merging* des Master-Branchs in den Feature-Branch vorgenommen. Das heißt, die zwei Branchs werden so zusammengeführt, dass davon nur der Feature-Branch beeinflusst wird. Aus einem Merging entsteht ein Merge-Commit, der in diesem Fall auf dem Feature-Branch liegt.

[45] *Pair-Programmierung* stammt aus der Softwareentwicklungsmethodik *Extreme Programming (XP)* und ist selbst eine Methodik, bei der Software von zwei Entwicklern zusammen, im *Pair*, an einem Arbeitsplatz, sprich einem Computer, entwickelt wird. Im Umfang dieser Arbeit kann nicht weiter auf XP eingegangen werden. Bei Interesse möge sich der Leser der weiterführenden Fachliteratur bedienen.
[46] Näheres zu *GitFlow* (auch: *Git-Flow* oder *Git Flow*) siehe auf der Webseite von Driessen: Driessen (2010)

Auch nachdem ein Feature vollständig umgesetzt wurde, wird ein Merging des Master-Branchs in den Feature-Branch durchgeführt. Die daraus entstehende Codebasis, auf dem Stand des Merge-Commits, wird mindestens durch manuelles Anstoßen der automatisierten Tests auf Fehlerfreiheit überprüft. Ist die Codebasis fehlerfrei, wird ein Merging des Feature-Branchs in den Master-Branch vorgenommen.
Soll eine neue Version des delegs-Editors bereitgestellt werden, wird diese aus der Codebasis erzeugt, die auf dem Master-Branch liegt. Bereitgestellte Versionen des delegs-Editors werden auf dem Master-Branch durch Tags markiert.

Mit dem Workflow kann es zu einem Sonderfall kommen, wenn eine Testversion bereitgestellt und in dieser ein Fehler gefunden wurde und zugleich schon an der Folgeversion der Anwendung weiterentwickelt wurde. Der Sonderfall macht den Umgang mit dem Workflow kompliziert. Ihn an dieser Stelle genauer zu beschreiben, ist allerdings nicht zielführend für die Analyse des Projekts. Eine Beschreibung ist im Anhang unter c. Beschreibung des Sonderfalls im Workflow des Ist-Zustands zu finden.

Es gibt Fälle, in denen Feature-Branchs über mehrere Wochen oder auch über einen Monat hinaus existieren. Dies ist der personellen Fluktuation und der gewachsenen Codebasis geschuldet. Gelegentlich läuft eine Umsetzung eines Features in komplexe Refactorings hinein, wodurch der entsprechende Branch länger existiert. Es gibt auch Fälle, in denen Features einen höheren Fokus als andere bekommen und bevorzugt bearbeitet werden. Dies kommt allerdings selten vor, denn dass Tech-Team versucht, *keine Baustellen offen liegen zu lassen.*

6.7 Deployment-Prozess (Ist-Zustand)

In diesem Abschnitt wird der Ist-Zustand des Deployment-Prozesses im delegs-Projekt beschrieben. Es wird versucht, den Prozess nicht zu detailliert zu beschreiben. So werden unnötige Details, wie beispielsweise genaue Muster für Dateibenennungen, nicht aufgeführt. Dieser Abschnitt hat den Anspruch den Prozessablauf zu erfassen, nicht aber eine vollständige Anleitung für den Deployment-Prozess zu sein.

Der Deployment-Prozess im delegs-Projekt, der durchlaufen wird, wenn eine neue Version des delegs-Editors bereitgestellt werden soll, ist in zwei Subprozesse unterteilt: das *Test Deployment* und das *Produktiv Deployment*. Ein Produktiv Deployment wird immer nur nach einem erfolgreich durchgeführten Test Deployment ausgeführt. Sollte eine per Test Deployment bereitgestellte Version nicht den Qualitätsansprüchen des Projekts entsprechen, kann kein Produktiv Deployment vorgenommen werden. Zwischen den beiden Subprozessen liegt eine Zeitraum von mindestens einer Woche. Dadurch kommt es des Öfteren vor, dass die beiden

Subprozesse eines Deployments von unterschiedlichen Mitgliedern des Tech-Teams und so in unterschiedlichen Entwicklungsumgebungen, den *Entwicklerlaptops*, durchgeführt werden.

Die Entscheidung darüber, ob ein Deployment vorgenommen werden soll, wird einmal wöchentlich im Tech-Team-Meeting von allen anwesenden Entwicklern und dem technischen Projektleiter gemeinsam getroffen.
Die manuelle Abarbeitung der Subprozesse findet immer in Pairs statt. Für beide Subprozesse ist das Vorgehen jeweils in einer Textdatei beschrieben, die über das Projekt-Repository zugänglich ist. In diesem wird auch die Codebasis des Editors verwaltet.[47] Bei Durchführung eines Subprozesses wird eine Kopie der entsprechenden Textdatei angelegt. Die darin beschriebenen einzelnen Schritte werden nach deren Durchführung mit einem "(x)" markiert. Die Kopie der Datei wird nach erfolgreicher Durchführung des Subprozesses als Protokoll des Vorgangs im Projekt-Repository abgelegt. Der Dateiname des Protokolls wird mit der Bezeichnung des Subprozesses und der Versionsnummer der bereitgestellten Version versehen. Im Protokoll selbst sind das Datum, nochmals die Versionsnummer sowie die Bezeichnung des Subprozesses festgehalten. Außerdem sind dort noch die Kürzel der Entwickler eingetragen, die den Subprozess durchgeführt haben.

Für die verschiedenen Deployment-Umgebungen und die Entwicklungsumgebung stehen vorgefertigte Konfigurationen im Repository bereit. Es finden sich dort Konfigurationen für den Produktiv- und Testeinsatz der Anwendung auf dem Hardware-Server sowie für das lokale Arbeiten auf den Entwicklerlaptops.

In den drei folgenden Unterabschnitten, wird nun zuerst das *Test Deployment* und danach das *Produktiv Deployment* beschrieben, um anschließend die beiden Prozesse als Pipeline darzustellen.

6.7.1 Test Deployment

Vorbereitung der neuen Testversion:

Zu Beginn des Test Deployments wird sichergestellt, dass lokal auf dem Master-Branch gearbeitet wird und dass dieser auf dem aktuellen Stand des remote Projekt-Repository ist.[48] Außerdem wird sichergestellt, dass das Schema der lokalen Datenbank auf dem aktuellen

[47] Der Begriff *Repository* meint im Kontext dieser Arbeit immer ein Verzeichnis, das von einem Versionsverwaltungssystem (VCS) verwaltet wird. Bei delegs wird als VCS *Git* eingesetzt. In diesem Abschnitt beziehen sich alle Pfadangaben auf das Projekt-Repository. Auf Pfade, die nicht im Projekt-Repository liegen, wird explizit hingewiesen. Kopien des Projekt-Repositories liegen sowohl auf jedem Entwicklerlaptop, als auch zu Synchronisationszwecken auf einem separaten Server der WPS.

[48] In diesem Abschnitt bezieht sich *lokal* immer auf eine Entwicklungsumgebung eines Entwicklers. *Remote* hingegen bezieht sich immer auf entfernt liegende, über das Internet erreichbare Ressourcen.

Stand ist. Dies kann anhand der Versionsnummer der Datenbank überprüft werden, die in einer Tabelle hinterlegt ist. Ist die Datenbank nicht auf dem aktuellen Stand, werden die nötigen Migrationsskripte auf der Datenbank ausgeführt, um ihre Version anzuheben. Die Migrationsskripte werden im Projekt-Repository hinterlegt. Ist für die aktuelle Version schon ein Migrationsskript vorhanden, heißt das, dass während des aktuellen Entwicklungszyklus Anpassungen an der Datenbank vorgenommen wurden. Ist noch kein Migrationsskript vorhanden, wird es angelegt. Danach werden in der Entwicklungsumgebung *Eclipse* alle Unit- und Integrationstests ausgeführt. Waren diese erfolgreich, führen die Entwickler Akzeptanztests auf einer auf dem Entwicklerlaptop gestarteten Instanz der Anwendung durch. Sie sind in einer Textdatei beschrieben, die im Projekt-Repository hinterlegt ist. Nachdem die Akzeptanztests durchgeführt wurden, wird die neue Versionsnummer im delegs-Editor in einer seiner Konfigurationsdateien gesetzt. Die Änderungen an der Konfigurationsdatei und das eventuell veränderte (bzw. neu angelegte) Migrationsskript für die Datenbank werden auf den Master-Branch hinzugefügt. Für den daraus neu entstanden Commit wird ein Versions-Tag erstellt, das als Bezeichnung die neue Versionsnummer des delegs-Editors trägt. Der neue Commit und der Versions-Tag werden auf das remote liegende Projekt-Repository übertragen. Als nächstes werden die für Testversionen vorgefertigten Dateien zur Konfiguration der Anwendung in das dafür vorgesehene *config* Verzeichnis kopiert. Sie liegen im Projekt-Repository in Unterordnern, die jeweils für die unterschiedlichen Konfigurationen angelegt wurden.

Erstellung des Builds:

Als nächstes wird die Anwendung unter Einsatz des *Google GWT-Compilers* gebaut. Hierfür wird der Compiler konfiguriert. Die Konfiguration ist abhängig von der Hardware des Entwicklercomputers. So kann dem Compiler die Anzahl von Prozessoren oder die Menge an Speicher angegeben werden, die während des Kompiliervorgangs genutzt werden dürfen. Diese Einstellungen bestimmen die Dauer, die der Compiler zum Bauen des Projekts benötigt. Der leistungsfähigste Computer, der dem Entwickler-Team zur Verfügung steht, benötigt bei Stand dieser Arbeit ca. zehn Minuten, um den Vorgang durchzuführen. Ist der Kompiliervorgang abgeschlossen, steht das kompilierte Projekt (das *Build-Artefakt*) im *war* Verzeichnis bereit. Es liegt eine gepackte *TutorialVideos.zip* Datei im Verzeichnis *setup*, deren Inhalt nun dem Build-Artefakt hinzugefügt und dafür in den *war/vid* Ordner entpackt werden muss. Dieser Vorgang ist nur bei einem Test Deployment nötig, wenn ein Subprozess des Deployments das erste Mal auf einer Entwicklungsumgebung durchgeführt wird. Bei erneutem Bauen des Projekts auf der selben Umgebung, wird das war/vid Verzeichnis nicht verändert und enthält

dementsprechend noch die entpackten Videos. Ist dies geschehen, wird als letzter Schritt das Build-Artefakt als *delegseditortest.war_* Datei im zip-Format gepackt.

Bereitstellung der Anwendung auf dem Hardware-Server:

Bereitgestellte **.war* Dateien liegen auf dem Hardware-Server im Verzeichnis */var/lib/tomcat8/webapps*. Ist noch eine alte Testversion des delegs-Editors bereitgestellt, wird diese zuerst als inaktiv gesetzt. Dies geschieht, indem die *delegseditortest.war* Datei aus dem Verzeichnis *webapps* gelöscht wird. Der Tomcat-Server stoppt und entfernt die bereitgestellte aktive Version daraufhin automatisch. Nach dem Löschen der Datei wird sichergestellt, dass die Testversion nicht mehr erreichbar und somit nicht mehr aktiv ist. Dies geschieht durch Aufrufen der URL der Testversion im Browser.

Der Zugriff auf den Hardware-Server erfolgt per *SSH* unter Einsatz des Programms *PuTTY* oder per *SFTP* unter Nutzung des Programms *Cyberduck*. Letzteres wird eingesetzt, um Dateien auf den Server hochzuladen. Es kann nicht auf alle Verzeichnisse auf dem Server zugreifen, da für den Zugriff teilweise *root*-Rechte benötigt werden. Diese können sich mit Cyberduck nicht verschafft werden. Sollen unter dessen Einsatz Dateien in ein Verzeichnis kopiert werden, das dem Nutzer *root* gehört, ist ein mehrschrittiges Kopieren der Dateien unter zusätzlichem Einsatz von PuTTY nötig.

Als nächstes wird die Testdatenbank des Servers angepasst:
Ein Dump aller Daten der Produktivdatenbank (die den Namen *signbase* trägt) wird als *self-contained*-Datei erstellt. In der erstellten Datei wird der Name der Datenbank von *signbase*, auf den Namen der Testdatenbank *signbase_test* geändert. Danach wird der Dump in die Testdatenbank des auf dem Hardware-Server laufenden MySQL-Servers eingespielt. Als letzter Schritt der Datenbankanpassung wird auf dieser nun noch das Migrationsskript für die neue Version auf die remote Testdatenbank angewendet.

Als letzten Schritt des Bereitstellens der Testversion wird die zuvor gebaute *delegseditortest.war_* Datei in das Verzeichnis *webapps* kopiert und danach in *delegseditortest.war* umbenannt. Das Kopieren und das Umbenennen müssen nacheinander durchgeführt werden. Damit wird vermieden, dass der Tomcat-Server versucht, die *.war Datei bereitzustellen, bevor diese vollständig in das Verzeichnis kopiert wurde. Daher die Benennung der Dateiendung (**.war_*). Der Tomcat-Server stellt automatisch alle im *webapps* Verzeichnis abgelegten **.war* Dateien als eigene Webseite bereit, unter der URL
http://www.delegs.de/[Name der war Datei ohne Dateiendung]/

Durchführen diverser Tests:

Unter anderem wird die Performance der Anwendung gemessen. Hierzu prüft man manuell, wie schnell der delegs-Editor drei vorgegebene Dokumente öffnet, wenn die Anwendung über den *Google Chrome*-Browser genutzt wird. Die gemessenen Zeiten werden im Deployment-Protokoll festgehalten.

Erneut werden von den Entwicklern alle manuellen GUI-Tests durchgeführt, diesmal natürlich auf der Version, die auf dem Server bereitgestellt wurde. Der dazu genutzte Browser wird aus den unterstützten Browsern beliebig gewählt und die Wahl im Protokoll festgehalten. Danach wird das Deployment-Protokoll im Projekt-Repository abgelegt.

Informieren über neue Testversion:

Ein spezieller Power User und die restlichen Stakeholder werden per E-Mail über die bereitgestellte Testversion und die darin vorgenommen Änderungen und Erweiterungen informiert. Sie haben eine Woche Zeit, die Testversion zu nutzen und auf Fehler zu prüfen. In diesem Zeitraum, der in der weiteren Arbeit von nun an als *Testwoche* bezeichnet wird, testet das Lehrteam das Nachbauen verschiedener Beispieldokumente, um sicherzustellen, dass die Anwendung korrekt funktioniert (Akzeptanztests). Auch der spezieller Power User testet die Anwendung. Er ist Lehrer und setzt die Anwendung viel im Unterricht ein.

Im Fehlerfall:

Fallen während der Tests durch die Entwickler oder während der Testwoche Fehler auf, wird entschieden, ob diese schwer genug sind, um ein Produktiv Deployment zu verhindern. Die Einschätzung der Schwere eines Fehlers wird auf technischer Ebene vom Tech-Team und auf fachlicher Ebene vom Lehrteam vorgenommen. Kann die Anwendung wegen vorhandener Fehler nicht in den Produktivbetrieb gehen, werden die Fehler mit höchster Priorität behoben und danach ein neues Test Deployment angestoßen. Wurden keine Fehler in der Anwendung gefunden, wird als nächster Schritt das Produktiv Deployment vorgenommen.

6.7.2 Produktiv Deployment

Viele Schritte des Produktiv Deployments werden analog zum Test Deployment ausgeführt. Da diese bereits oben für das Test Deployment beschrieben wurden, wird auf sie für das Produktiv Deployment nur noch kurz eingegangen.

Die produktiv bereitzustellende Version hat den selben Stand der Codebasis, der auch für das Test Deployment genutzt wurde - es wird dasselbe Versions-Tag genutzt.

Vorbereitungen:

Zwei Tage bevor die neue Version bereitgestellt werden soll, wird mit einem speziellen Power User ein Zeitfenster für das Produktiv Deployment abgeklärt. Hintergrund hierfür ist, dass der Anwender ein Lehrer ist, der den delegs-Editor viel im Unterricht einsetzt. Es soll vermieden werden, dass ihm durch das Bereitstellen einer neuen Version Probleme entstehen, bedingt durch den Ausfall der Anwendung.
Mindestens drei Stunden vor dem Produktiv Deployment wird eine E-Mail an den speziellen Power User und alle restlichen Stakeholder verschickt, in der diese über das Deployment und dessen Zeitpunkt informiert werden. Die Mail wird fünfzehn Minuten vor dem Deployment erneut verschickt. In beiden Fällen werden die Empfänger auf den Zeitpunkt hingewiesen, zu dem die Anwendung ausfallen wird und dass es dringend erforderlich ist, die Nutzung der Anwendung rechtzeitig einzustellen und zuvor den gegenwärtige Arbeitsstand zu speichern.

Erstellung des Builds:

Die für Produktivversionen vorgefertigte Konfiguration wird in das dafür vorgesehene Verzeichnis *config* kopiert. Danach wird die Konfigurationsdatei, die die Versionsnummer des delegs-Editors enthält, darauf überprüft, dass der entsprechende Eintrag auf die neue Version gesetzt ist. Dies sollte so während des Test Deployments geschehen sein. In einer entsprechenden xml Konfigurationsdatei werden Debug-Einstellungen auskommentiert.
Nun wird das Projekt mit dem GWT-Compiler gebaut. Sofern noch nicht geschehen, wird danach die *TutorialVideos.zip* Datei aus dem *setup* Verzeichnis in den *war/vid* Ordner entpackt. Zum Schluss wird der Inhalt des *war* Verzeichnisses in die Datei *delegseditor.war_* gepackt.

Bereitstellung der Anwendung:

Die Vorgängerversion des delegs-Editors wird inaktiv gesetzt, indem die *delegseditor.war* Datei aus dem Verzeichnis *webapps* in ein Verzeichnis auf dem Hardware-Server verschoben wird, das als Archiv für alte Versionen dient. Dies geschieht, um darauf im Fehlerfall zurückgreifen zu können und ein Rollback zu ermöglichen. Danach wird sichergestellt, dass die alte Produktivversion nicht mehr erreichbar und somit nicht mehr aktiv ist. Für den Fehlerfall wird auch ein Backup der Produktivdatenbank erstellt und lokal auf dem Entwicklerrechner gespeichert. Danach wird das aktuelle Migrationsskript auf der Produktivdatenbank ausgeführt. Die *delegseditor.war_* Datei wird in das Verzeichnis *webapps* verschoben und in *delegseditor.war* umbenannt, wodurch der Tomcat-Server die neue Version bereitstellt.

Durchführen von Tests:

Auf der neuen Version werden die manuellen GUI-Tests durch die Entwickler durchgeführt und es wird wieder festgehalten, welcher Browser für die Tests verwendet wurde.

Abschlussarbeiten:

Das während der Erstellung des Builds vorgenommene Auskommentieren der Debug-Einstellungen wird wieder rückgängig gemacht. Außerdem wird das Deployment-Protokoll im Projekt-Repository abgelegt.

Informieren über neue Produktivversion:

Auch über das Deployment der neuen Produktivversion und die darin vorgenommenen Änderungen und Erweiterungen werden Power User und die restlichen Stakeholder per E-Mail informiert. Außerdem wird auf die neue Version auf der Homepage und auf der Facebook-Seite von delegs hingewiesen.

Rollback im Fehlerfall:

Sollte während des Deployment-Prozesses ein Fehler auftreten sein, so wird der Versuch unternommen, diesen zu beheben. Sollte der Fehler gravierend sein oder dessen Behebung zu viel Zeit beanspruchen, wird das Produktiv Deployment abgebrochen. Eine maximale Dauer, die nicht überschritten werden darf, ist nicht gesetzt. Allerdings wird ein Produktiv Deployment tendenziell abgebrochen, wenn dieses einen Ausfall der Anwendung von mehr als einem halben Tag bedeuten würde. Ist ein Fehler aufgetreten nachdem die alte Produktivversion inaktiv gesetzt wurde, wird diese aus dem Archiv geholt und wieder in den aktiven Produktivbetrieb gestellt. Entsprechend wird auch mit dem erstellten Backup der Datenbank verfahren.

6.7.3 Der Deployment-Prozess als Pipeline

An dieser Stelle scheint es sinnvoll, den gesamten Deployment-Prozess in einer Deployment-Pipeline abzubilden. Ziel ist es, dadurch ein besseres Verständnis für den Prozess zu bekommen und später, während der Entwicklung einer Lösungsstrategie, darauf zurückgreifen zu können. Fowler (2013) empfiehlt dies in einem Beitrag auf seiner Website:
"Eine gute Art Continuous Delivery einzuführen ist, den gegenwärtigen Auslieferungsprozess als eine Deployment-Pipeline zu modellieren und danach [diese] auf Engpässe, Möglichkeiten

zur Automatisierung und auf Stellen an denen zusammengearbeitet werden kann zu untersuchen."[49]
Hierzu wurden zwei Abbildungen erstellt. Eine für das Test Deployment, Abbildung 6.2 und eine für das Produktiv Deployment, Abbildung 6.3.

[49] Die hier erwähnte *Continuous Delivery*, meint im Kontext dieser Arbeit *Continuous Deployment*. Siehe dazu 5.6.2 Continuous Delivery & Continuous Deployment.

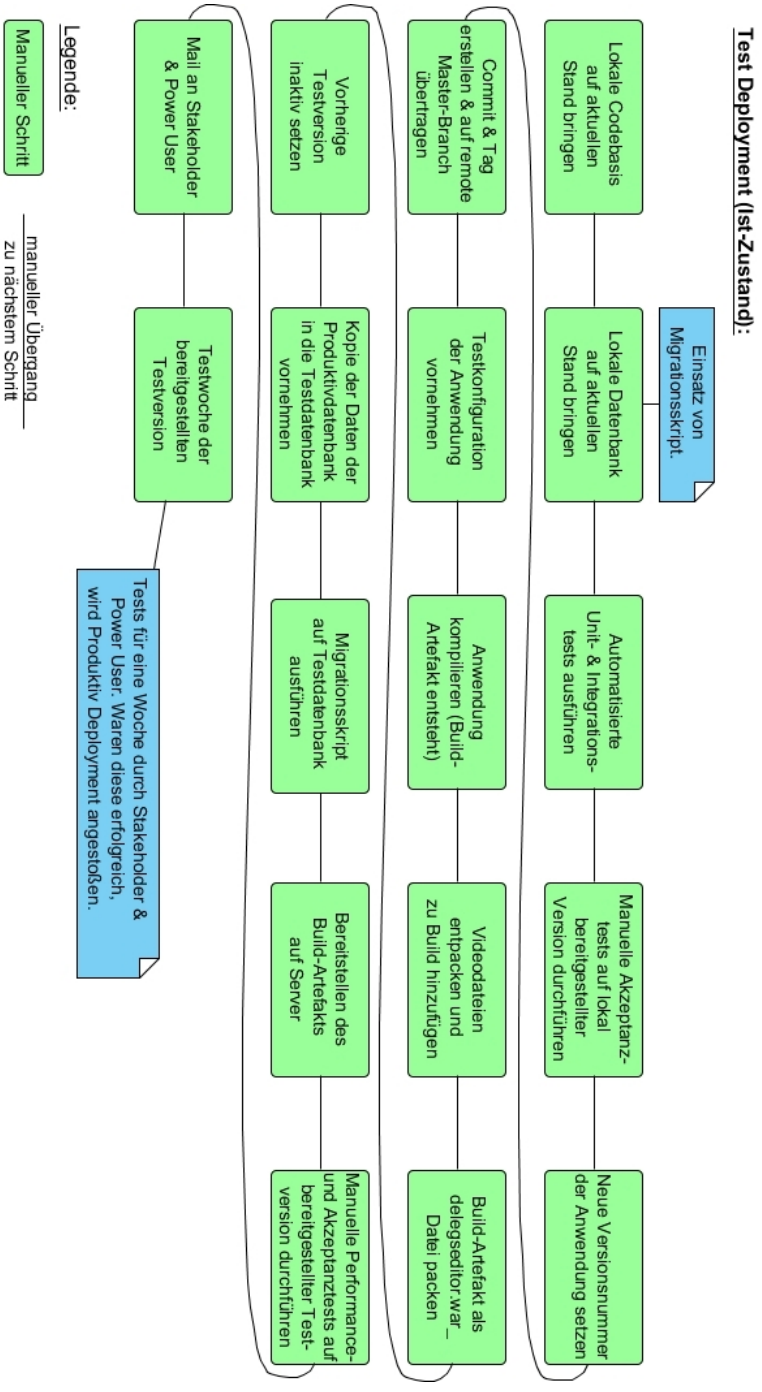

Abbildung 6.2: Deployment-Pipeline für Test Deployment (Ist-Zustand)

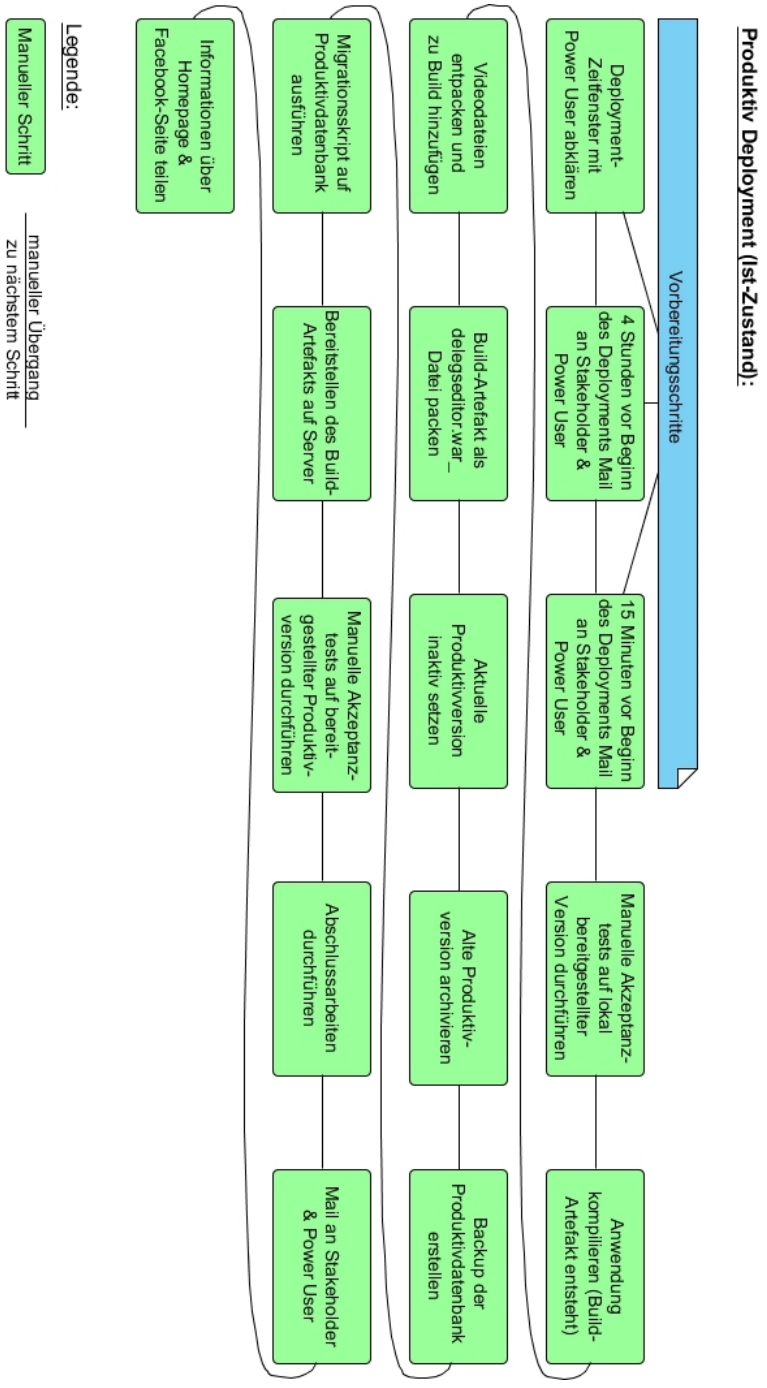

Abbildung 6.3: Deployment-Pipeline für Produktiv Deployment (Ist-Zustand)

6.8 Interviews im delegs-Projekt

Zur Analyse des Projekts wurden Interviews, speziell mit dem Fokus auf das Thema DevOps, mit allen Mitarbeitern des zentralen Teams geführt. Dabei wurden zwei verschiedene Interview-Gruppen zusammengefasst. Die eine Gruppe bildet das Tech-Team zusammen mit dem (technischen) Projektleiter, Herrn Koch und die andere Gruppe bildet das Lehrteam. Für jede der beiden Gruppen wurden speziell zugeschnittene Fragen vorbereitet. Dem Tech-Team wurden Fragen über den Entwicklungs- und Deployment-Prozess gestellt und außerdem solche, die sich mit der Zusammenarbeit und der Rollen- und Aufgabenverteilung im Team beschäftigen. Die Fragen wurden so zusammengestellt, dass sie auch für die Analyse anderer Projekte angewendet werden können und so einen Mehrwert haben. Die Fragen, die dem Lehrteam gestellt wurden, sind nur auf die Zusammenarbeit im zentralen Team, auf das Rückkoppeln von Fehlern und das Einbringen von Feature-Requests bezogen. Das hat den Hintergrund, dass das Lehrteam kaum an der Entwicklung und dem Deployment beteiligt ist.
Nachfolgend werden die während der Interviews gestellten Fragen aufgeführt und die Ergebnisse zusammengefasst und ausgewertet. Die kompletten Interviews sind im Anhang dieser Arbeit zu finden.[50]

6.8.1 Interview mit Tech-Team & technischen Projektleiter

Das Interview mit dem Tech-Team und dem technischen Projektleiter hatte das Ziel, Informationen direkt vom an der Entwicklung und am Deployment-Prozess beteiligten Personal zu sammeln. Es sollte herausgefunden werden, wie das Projekt in den vielseitigen Aspekten von DevOps aufgestellt ist. Diese Informationen sollten aus der individuellen Sicht der Teammitglieder gewonnen werden, unabhängig davon, ob sie mit DevOps vertraut waren. Zusätzlich sollten mögliche Probleme im Projekt sowie Wünsche und Ideen zur Verbesserung gesammelt werden, anhand derer Anforderungen für die Einführung von DevOps abgeleitet werden sollen. Im Folgenden nun die Interview-Fragen.

Interview-Fragen

Da der Begriff "Deployment-Prozess" im IT-Bereich mit unterschiedlicher Bedeutung belegt ist, wurde dem Befragten zu Beginn des Interviews eine Definition zu dem Begriff gegeben. So wurde zwischen Interviewer und Befragten eine gemeinsame Basis geschaffen, auf der das Interview stattfinden konnten. Der Begriff "Deployment-Prozess" wurde wie folgt beschrieben:

[50] Komplette Interviews im Anhang, siehe e. Interviews mit dem zentralen Team

Der Deployment-Prozess ist Teil des Softwareentwicklungsprozesses und setzt genau an der Stelle ein, an der Code durch Softwareentwickler, wie z.B. in Form eines neuen Features, fertiggestellt wurde. Das Deployment ist das Bereitstellen dieses Codes in Form einer neuen Version in Test- oder Produktivumgebungen, z.B. auf einem virtuellen oder physischen Hardware-Server. Unter den Begriff *Deployment-Prozess* kann auch schon das Integrieren von Code in die Codebasis fallen und auch Produktivbetrieb von Software und deren Wartung.

Folgende Fragen wurden während des Interviews gestellt:

1.) Was ist Ihre Aufgabe im Projekt?
2.) Gibt es mehrere Teams, die an der Software arbeiten?
    2.1) Wenn ja: Wie viele Entwickler haben die jeweiligen Teams?
3.) Wie viele Entwickler arbeiten in Ihrem Projektteam?
4.) Administratorische Aufgaben im Projekt:
    4.1) Gibt es Administratoren / Operators, die für Ihr Projekt zuständig sind oder die Teil Ihres Projektteams sind und im Team mitarbeiten?
        4.1.1) Wenn ja, wie viele und welche Aufgaben haben diese?
    4.2) Bei mehreren Teams:
    Gibt es Administratoren / Operators, die für andere Teams ihres Projekts zuständig sind oder die als Teammitglied in den anderen Teams mitarbeiten?
        4.2.1) Wenn ja, wie viele und welche Aufgaben haben diese?
    4.3) Werden administratorische Aufgaben / Operations-Aufgaben von Entwicklern übernommen?
        4.3.1) Wenn ja, welche?
5.) Wie oft, also in welchem Rhythmus oder nach welchen Zeiträumen, ist in Ihrem Projekt ein Produktiv-Deployment von neuen Features, Versionen oder Ähnlichem vorgesehen?
    5.1) Wenn Rhythmus vorgesehen: Wird der vorgesehene Rhythmus eingehalten?
        5.1.1) Wenn nein: Was sind die Gründe, die ein Deployment verzögern?
    5.2) Wenn nein: Sollte es Ihrer Meinung nach einen festen Deployment-Rhythmus geben?
        5.2.1) Wenn ja, warum?
6.) Der Deployment-Prozess (bei mehreren Prozessen die folgenden Fragen je Prozess durchgehen):
    6.1) Gibt es einen oder mehrere Deployment-Prozesse (z.B. Test- & Produktiv Deployment)?

6.2) Sind Sie am Deployment-Prozess beteiligt?
Wenn nein: Weiter bei 6.10)
6.3) Beschreiben Sie den Deployment-Prozess.
6.4) Welche Infrastruktur wird genutzt (Tools, Server, etc.)?
6.5) Gibt es klar verteilte Aufgaben und Verantwortlichkeiten für das Deployment?
6.5.1) Wer ist zuständig für den Deployment-Prozess? Beschreiben Sie die Verteilung der Aufgaben und die Aufgaben selbst.
6.5.2) Wer ist zuständig für die während des Deployment-Prozesses genutzte Infrastruktur?
6.6) Fehlerfälle während des Deployment-Prozesses:
6.6.1) Gibt es in einem Fehlerfall während des Deployment-Prozesses ein festgelegtes Vorgehen?
6.6.1.1) Wenn ja: Beschreiben Sie dieses!
6.6.2) Wie wird, unabhängig von möglichen Vorgehensvorschriften, tatsächlich reagiert, welche Maßnahmen werden ergriffen?
6.7) Empfinden Sie den Deployment-Prozess als leichtgewichtig oder schwergewichtig?
6.7.1) Ist er leicht zu verstehen?
6.7.2) Ist er leicht umzusetzen?
6.8) Ist der Deployment-Prozess, bzw. das Vorgehen bei diesem, ausreichend und verständlich dokumentiert und ist die Dokumentation leicht zugänglich?
6.9) Gibt es während des Deployment-Prozesses Feedback (z.B. durch Monitoring), das klar verständlich ist, sodass Sie sich sicher genug fühlen in der Durchführung des Deployments?
6.9.1) Gibt es genug Feedback, wenn Probleme auftreten?
6.9.2) Gibt es genug Feedback, wenn keine Probleme auftreten?
6.10) Gibt es generell Probleme, die Sie im Deployment-Prozess wahrnehmen / erfahren?
6.10.1) Wenn ja, beschreiben Sie diese.
6.11) Sehen Sie Behinderungen, die den Fluss des Deployment-Prozesses aufhalten?
6.11.1) Wenn ja, welche?
7.) Gibt es Wünsche oder Verbesserungsvorschläge für Ihr Projekt,
7.1) in Bezug auf den Deployment-Prozess?
7.2) in Bezug auf den Informationsfluss, hinsichtlich der Rückmeldungen durch den

Deployment-Prozess (sowohl an die direkt beteiligten Kräfte, als auch an die nicht direkt beteiligten Kräfte)?

7.3) in Bezug auf das Testen?

7.4) in Bezug auf die Verteilung von Aufgaben und Verantwortlichkeiten?

8.) Haben Sie noch weitere Anmerkungen?

6.8.2 Interview mit dem Lehrteam

Es erschien sinnvoll, auch Interviews mit den Mitgliedern des Lehrteams zu führen. Grund dafür ist, dass das Lehrteam maßgeblich Anforderungen an die Entwicklung des Editors stellt und Teil des Personals ist, das die Akzeptanztests an der Software durchführt. Ziel war es hierbei, Informationen darüber zu erhalten, wie die Kommunikation und Interaktion innerhalb des zentralen Teams funktioniert, speziell zwischen Lehr- und Tech-Team. Die Auswertung der Informationen versprach, eventuell weitere Möglichkeiten zur Verbesserung des Arbeitsprozesses zu finden.

Interview-Fragen

1.) Wie findet die Kommunikation mit dem Tech-Team statt?

1.1) Sehen Sie hier Verbesserungsmöglichkeiten?

2.) Wie gehen Sie vor, wenn Ihnen Fehler oder Probleme in einer Testversion auffallen?

2.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

3.) Wie gehen Sie vor, wenn Ihnen Fehler oder Probleme in einer Produktivversion auffallen?

3.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

4.) Wie gehen Sie vor, wenn Ihnen Verbesserungsvorschläge für die Anwendung einfallen?

4.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

5.) Haben Sie den Eindruck, dass Ihr Feedback aufgenommen und entsprechende Maßnahmen umgesetzt werden?

6.) Haben Sie noch weitere Anmerkungen?

6.8.3 Übergabe durch das alte DevOps-Einführungsteam

Bei delegs hat es in der Vergangenheit bereits einen Versuch gegeben, DevOps in das Projekt zu integrieren. Es gab ein dreiköpfiges DevOps-Einführungsteam (folgend als *DevOps-Team*

bezeichnet), bestehend aus einem Entwickler des delegs-Teams und zwei projektexternen Mitarbeitern. Die Mitglieder des DevOps-Teams sind Teil der *Special Interest Group: DevOps (SIG DEVOPS)* der Firma WPS, die sich auf das Einführen von DevOps in Projekten und Firmen spezialisiert hat. Nachdem das DevOps-Team seine Arbeit am delegs-Projekt begonnen hatte, wurden die zwei projektexternen Mitarbeiter aus dem Team herausgenommen, da sie in anderen Projekten gebraucht wurden. Bei delegs wurde daraufhin überprüft, ob es sinnvoll wäre, die vom DevOps-Team angefangene Arbeit fortzuführen. Die Entscheidung fiel gegen eine Fortführung aus, da der Aufwand als zu hoch eingeschätzt wurde. Das Tech-Team hätte sich nur noch auf die Einführung von DevOps in das Projekt konzentrieren können und wäre nicht mehr in der Lage gewesen, sich mit der Weiterentwicklung der Anwendung zu beschäftigen. Die Pläne des DevOps-Teams wurden nur anfänglich umgesetzt und der umgesetzte Teil kam im Projekt nie zum Einsatz. Ein Gespräch mit dem DevOps-Team hat deren Ideen, Pläne und Umsetzungen offengelegt und diese werden im Folgenden beschrieben.

Die Art, wie das Bereitstellen einer neuen Version des delegs-Editors durchgeführt wird, war die Hauptmotivation des DevOps-Teams, um DevOps in das delegs-Projekt zu bringen. Es sollte sich der entstehenden Ausfallzeit der Anwendung und dem umständliche Senden einer Nachricht an alle Stakeholder, inklusive Power User entledigt werden.

An der Art der Bereitstellung hat sich seitdem nichts geändert.[51] Vom DevOps-Team wurde damals festgestellt, dass dem Projekt *Continuous Integration (CI)* und *Continuous Delivery (CD)* fehlte. Die Einführung von CI erschien wichtig, um die Qualität der Anwendungsentwicklung zu steigern und die bestehende Qualität der Anwendung zu sichern. Außerdem sollte den Entwicklern die Arbeit vereinfacht werden. Mit der Einführung von CD sollte der Deployment-Prozess beschleunigt und wiederholbar werden. Der Plan des Teams war es, zuerst CI zu realisieren. Hierfür sollte das Provisionierungstool *Vagrant* eingesetzt werden. Mit dessen Einsatz sollten Entwicklungs-, Test- und Produktivumgebungen provisioniert werden und zwar unabhängig von den verschiedenen Hard- und Softwarekonfiguration der Entwickler-PCs und der Test- und Anwendungsserver. Hierdurch sollte die Möglichkeit geschaffen werden, einfach, schnell und weitestgehend einheitliche Entwicklungs-, Test- und Produktivumgebungen aufzusetzen. Außerdem sollte die Provisionierung ermöglichen, alle erstellten Umgebungen auch auf Entwicklerrechnern hochfahren zu können, um mit ihnen lokal zu arbeiten und beispielsweise auf ihnen zu testen. In einer ersten Vagrant-Datei wurde die Erzeugung eines *Jenkins*-Servers[52] beschrieben. Das DevOps-Team hat sich bewusst gegen den Einsatz von *Docker* entschieden, da die Komplexitätsstufe, die Docker mitgebracht hätte, für das delegs-Projekt, als zu hoch

[51] Näheres zum Deployment-Prozess (des IST-Zustands) ist unter 6.7 Deployment-Prozess (Ist-Zustand) zu finden.
[52] *Jenkins* ist ein webbasiertes CI-System.

eingestuft wurde.
Von einer Veränderung der Aufgabenverteilung im Tech-Team, wie im Abschnitt 5.3 Das DevOps-Team beschrieben, wurde abgesehen, da im Tech-Team bereits alle Aufgaben von allen Teammitgliedern gemeinsam übernommen wurden und es keine klassische Aufteilung in Entwicklungs- und Operationsteam gab.

6.8.4 Auswertung der Interviews

An dieser Stelle sollen nun die Erkenntnisse aus den Interviews festgehalten werden. In die Auswertung gehen die Ergebnisse aus den Interviews beider Gruppen *Tech-Team & Projektleiter* und *Lehrteam* ein. Sie lassen sich in den folgenden vier Kategorien zusammenfassen: *Teamstruktur*, *Entwicklungsprozess*, *Deployment-Prozess* und *Veränderungswünsche*. Zuerst werden die Ergebnisse der einzelnen Kategorien aufgezeigt und anschließend aus ihnen mögliche Anforderungen extrahiert.

Teamstruktur

Aus den Interviews geht hervor, dass es im Tech-Team keine klassische Aufteilung in Entwicklungsteam und Operations-Team gibt. Alle Teammitglieder können alle anfallenden Aufgaben im Projekt übernehmen. Es gibt allerdings einen Teilbereich der Hardware-Server-Konfiguration und -Verwaltung, in dem das Tech-Team nicht unabhängig ist, da dort der Anbieter des genutzten managed Hardware-Servers Aufgaben übernimmt und das Projekt in Abhängigkeit zu ihm steht. Zu den vom Anbieter übernommenen Aufgaben gehören z.B. Updates des Betriebssystems oder des Webservers. Auch gibt es projektexterne Administratoren der Firma WPS, die Aufgaben in diesem Bereich übernehmen. Im Team scheint es außerdem die Tendenz einer Wissensinsel zu geben: Ein Teammitglied hat mehr Wissen über das Projekt als die anderen.

Entwicklungsprozess

Für den Entwicklungsprozess ergibt sich aus den Interviews, dass Probleme meist direkt angesprochen werden können, da das Tech-Team und das Lehrteam gemeinsam in einem Büro sitzen. Es gibt dadurch eine relativ enge Rückkopplung zwischen den Entwicklern und Teilen der Stakeholder. Probleme können spätestens bei den montags stattfindenden Meetings angesprochen werden. Allerdings wurde als ein Hindernis hierbei die lange Liste an Anforderungen und Aufgaben des Tech-Teams genannt. Diese führe dazu, dass aus Rücksicht auf das Tech-Team, Fehler in der Anwendung nicht immer angesprochen werden.

Die besprochenen Inhalte der Meetings des zentralen Teams scheinen teilweise zu technisch für das Lehrteam zu sein. Auch scheint dem Lehrteam teilweise die Übersicht darüber zu fehlen, woran das Tech-Team gegenwärtig konkret arbeitet. Die Kommunikation darüber, welche Anforderungen in der Zukunft umgesetzt werden sollen, scheint auch nicht klar genug mit dem Lehrteam kommuniziert zu werden. Während der Interviews wurde der Wunsch geäußert, mehr Absprachen zwischen Lehr- und Tech-Team über die Priorisierung der als nächstes umzusetzenden Anforderungen zu treffen.

Als negativ wurde der Zeitaufwand für das Ausführen der automatisierten Tests genannt.

Deployment-Prozess

Der Deployment-Prozess wird vom Tech-Team im Allgemeinen als schwergewichtig, zeitaufwändig und auch im Hinblick auf die manuell auszuführenden Aufgaben aufwändig empfunden. Speziell das Kompilieren der Anwendung, das Ausführen der automatisierten Tests und das manuelle Testen würden zu viel Zeit in Anspruch nehmen. Der Prozess als solches wurde aber als leicht verständlich und ausreichend dokumentiert bewertet. Zu der Frage, ob der Prozess genug Rückmeldung für die Tech-Team-Mitglieder gibt, gab es keine einheitliche Meinung. Hier sollten vor allem die Stimmen gehört werden, für die der Prozess zu wenig Rückmeldung gibt. Denn ein Deployment kann nur effektiv sein, wenn es vom gesamten dafür zuständigen Personal einfach und ohne Probleme durchgeführt werden kann. Von einem Tech-Team-Mitglied wurde das Problem genannt, dass bei Fehlern Rückmeldungen aus dem Prozess teilweise zu kryptisch und dadurch schwer zu verstehen seien. Als Behinderung der Arbeit während des Deployments wurden zu den oben bereits erwähnten Punkten noch die unterschiedlich konfigurierten Umgebungen genannt, die eingesetzt werden.

Die Mitglieder des Tech-Teams sprachen außerdem davon, dass die Bereitstellung der Anwendung nur unregelmäßig stattfindet. Als Grund hierfür wurde die Schwergewichtigkeit des Deployment-Prozesses genannt. Es lohne sich wegen des Aufwands nicht, kleine oder nur wenige fertiggestellte Features bereitzustellen. Ein weiterer genannter Grund war die personelle Fluktuation, wodurch die Entwicklung an Features verzögert würde.

Veränderungswünsche

Als Veränderungswünsche wurde folgende genannt:

- regelmäßig stattfindendes Deployment
- höherer Grad an Automatisierung:
  Hier wurden Continuous Integration und Build-Server genannt sowie automatisierte

(GUI-)Tests. Außerdem gab es den Wunsch, nach jedem Produktiv Deployment automatisiert eine Mail zu erstellen, die Informationen über die neu eingeführten Features (Namen der Anforderungen und deren Ids) enthält. Diese sollte nicht zu technisch formuliert sein und vor dem Verschicken manuell überprüft werden.
Allerdings gab es zur Automatisierung vom technischen Projektleiter auch eine im Widerspruch stehende Anmerkung: Das delegs-Projekt ist ein Ausbildungsprojekt und daher sei es nicht gut, wenn das Deployment automatisiert werde, da die Teammitglieder damit die Möglichkeit verlören, die "[...] einzelnen Schritte [des Deployment-Prozesses] sehen und verstehen zu können [...]".[53]

- Deployment wird ohne Ankündigung bei Stakeholdern und speziellem Power User durchgeführt:
  Zu diesem Wunsch wurde ein weiterer ähnlicher geäußert, der hier auch aufgeführt werden soll: Durch das Produktiv Deployment soll kein Ausfall der laufenden Anwendung stattfinden und dadurch keine Einschränkung für die Anwender bei ihrer Arbeit entstehen.

- Unabhängigkeit von (Entwicklungs-)Umgebungen:
  Einheitliche Konfiguration der eingesetzten Umgebungen, um Fehler zu vermeiden, die durch unterschiedlich Konfigurationen entstehen.

- Übersicht der Testabdeckung

- mehr Austausch zwischen Lehr- und Tech-Team

Anforderungen aus Interviews

Nun werden mögliche Anforderungen aufgeführt, die sich aus den Interviews ergeben und die in den Bereich von DevOps fallen. Es handelt sich hierbei um *vorläufige Anforderungen*, die bei der Anforderungserhebung für diese Arbeit untersucht und aufgenommen bzw. verworfen werden (siehe hierzu Kapitel 7 Anforderungen). Zur Abgrenzung von den tatsächlich aufgenommenen Anforderungen erhalten die vorläufigen Anforderungen einen temporären Identifikator (*Id*): *TA[Natürliche Zahl]*. Die vorläufigen Anforderungen sind in der folgenden Tabelle 6.1 zu finden.

[53] Zitiert aus dem Interview mit Herrn J. Koch (technischer Projektleiter), siehe Anhang e. Interviews mit dem zentralen Team

Tabelle 6.1: Vorläufige Anforderungen aus Interviews

Id	**Name, Zusammenfassung**	**Beschreibung**
TA01	Infrastruktur wird selbst verwaltet	Die unterliegende Infrastruktur soll vom Tech-Team selbstständig verwaltet werden. Dabei geht es nicht darum, die Hardware selbst zu stellen, sondern die darauf laufende Software inkl. Betriebssystem und Software-Server selbst zu verwalten und up to date zu halten.
TA02	Vermeidung von Wissensinseln	Im Tech-Team soll so gearbeitet werden, dass keine neuen Wissensinseln entstehen und bestehende abgebaut werden. *Wissensinseln* sind einzelne Teammitglieder, die Wissen, welches kein anderes Teammitglied besitzt, in ihrer Person bündeln.
TA03	Rückkopplung durch Deployment-Pipeline	Die Deployment-Pipeline soll für die Mitglieder des Tech-Teams, für Menschen lesbare Rückmeldungen über Zustand und Verlauf der einzelnen Schritte der Pipeline geben. Sie soll zu jederzeit Informationen darüber liefern, in welchem Schritt sich der Deployment-Prozess befindet, unabhängig davon, ob der Prozess erfolgreich verläuft oder nicht.
TA04	Entwicklungsumgebung gleicht Produktivumgebung	Die Entwicklungsumgebung ist der Produktivumgebung so nah wie möglich nachempfunden. Dies bezieht sich nur auf das Testen der Anwendung. Wird die Anwendung auf der Entwicklungsumgebung testweise bereitgestellt, unterscheidet sich ihr Verhalten von einer produktiv laufenden Anwendung nur so minimal, dass es vernachlässigt werden kann.
TA05	Testumgebung gleicht Produktivumgebung	Die Testumgebung ist der Produktivumgebung so nah wie möglich nachempfunden. Wird die Anwendung auf der Testumgebung bereitgestellt, unterscheidet sich ihr Verhalten von einer produktiv laufenden Anwendung nur so minimal, dass es vernachlässigt werden kann.

Tabelle 6.1: Fortsetzung: Vorläufige Anforderungen aus Interviews

Id	Name, Zusammenfassung	Beschreibung
TA06	Ein Deployment ist jederzeit möglich	Der Aufwand, der durch das Durchführen des Deployment-Prozesses entsteht, ist unabhängig von Größe und Anzahl bereitzustellender Features immer insignifikant genug, um vernachlässigbar zu sein. Der aus dem Deployment entstehende Aufwand beeinflusst nicht die Entscheidung, ob ein solches durchgeführt werden soll. So ist ein Deployment jederzeit durchführbar.
TA07	Deployments finden regelmäßig statt	Deployments werden regelmäßig in einem festen Entwicklungszyklus vorgenommen.
TA08	Einsatz von Continuous Integration	Continuous Integration wird im Projekt eingesetzt.
TA09	Produktiv Deployment löst Mail-Generierung aus	Nach einem erfolgreichen Produktiv Deployment wird automatisch eine E-Mail erzeugt, in der Namen und Ids der umgesetzten Anforderungen enthalten sind. Die Mail wird erst an den speziellen Power User und die Stakeholder verschickt, wenn sie manuell überarbeitet wurde.
TA10	Keine Automatisierung des Deployment-Prozesses	Da es sich bei dem delegs-Projekt um ein Ausbildungsprojekt handelt, soll der Deployment-Prozess nicht automatisiert werden. Damit haben die Teammitglieder die Möglichkeit, die einzelnen Schritte des Prozesses durch manuelles Ausführen zu erfahren und zu verstehen.
TA11	Deployment wird bruchlos durchgeführt	Das Deployment findet bruchlos statt. Das Bereitstellen einer neuen Produktivversion kann bei laufendem Betrieb der aktuellen Version durchgeführt werden, ohne dass dem Anwender dadurch Daten verloren gehen oder die Anwendung ausfällt.

Tabelle 6.1: Fortsetzung: Vorläufige Anforderungen aus Interviews

Id	Name, Zusammenfassung	Beschreibung
TA12	Deployment durchführbar ohne Ankündigung	Es ist nicht mehr nötig, Anwender über den bevorstehenden Prozess des Produktiv Deployments zu informieren. Durch ein Produktiv Deployment läuft kein Anwender Gefahr, Daten zu verlieren.
TA13	Geringer Zeitaufwand für automatisierte Tests	Der Zeitaufwand, der durch das Durchführen automatisierter Tests entsteht, ist immer insignifikant genug, um vernachlässigbar zu sein. Entwickler, die automatisierte Tests anstoßen, werden dadurch nicht in ihrer Arbeit aufgehalten oder ausgebremst, es sei denn, sie sind abhängig von deren Ergebnis.
TA14	Klarheit über Vorgehen	Allen Teammitgliedern ist jederzeit einsehbar, welches weitere Vorgehen für die Entwicklung am delegs-Editor geplant ist.

6.9 Auswertung des Ist-Zustands

In diesem Abschnitt wird der beschriebene Ist-Zustand des delegs-Projekts aus der Sicht von DevOps bewertet. Es wird untersucht, an welchen Stellen im Projekt schon Strategien angewendet werden, die zu DevOps konform sind. Außerdem werden Empfehlungen für Änderungen im Projekt und an der Anwendung ausgesprochen, um dieses in ein DevOps getriebenes Projekt umzuwandeln.
Zu Beginn werden Aufstellung und Rollenverteilung des zentralen Teams untersucht, danach wird auf deren Arbeit und Arbeitsabläufe eingegangen. Es folgen Bewertung und Empfehlungen für die technische Infrastruktur und die eingesetzten Umgebungen. Zum Schluss wird noch der Deployment-Prozess näher untersucht. Alle Bewertungen und Empfehlungen beziehen sich auf die zusammengetragenen Informationen des restlichen Kapitels 6 Ist-Zustand.

Es gibt einige Dinge, die im Projekt schon DevOps konform sind. So werden Skripte und Code, die im Umgang mit der Infrastruktur eingesetzt werden, nach dem *Infrastructure-as-Code*-Prinzip gehandhabt und im Projekt-Repository abgelegt. Dazu gehören die Konfigurationsdateien für die Anwendung, die diese für die entsprechenden Umgebungen anpassen.

Auch die Protokolle, die festhalten wie der Deployment-Prozess durchgeführt wurde, liegen im Projekt-Repository. Außerdem werden die Migrationsskripte für die Datenbanken im Projekt-Repository abgelegt.

6.9.1 Auswertung: Struktur und Aufgabenverteilung des zentralen Teams

DevOps sieht ein Team vor, in dem Entwicklung und Operations nicht streng getrennt sind, sondern die früheren Grenzen zwischen diesen aufgelöst werden. Im Tech-Team existieren solche Grenzen nicht. Stattdessen gibt es eine gleichmäßige Verteilung aller Verantwortlichkeiten und anstehenden Aufgaben. Dies gilt für die Bereiche Entwicklung, Deployment, Wartung und für den Support. Die Aufgaben im Projekt sind also schon im Sinne der DevOps-Philosophie auf die Teammitglieder verteilt. Eine Ausnahme macht hier ein Teilbereich der Wartung des gemieteten Hardware-Servers. Genaueres zu dieser Problematik in Abschnitt 6.5 Der gestellte Hardware-Server.

Die Einführung des DevOps-Rollenmodells ist aufgrund des Kontextes, in dem das delegs-Projekt stattfindet und der daraus resultierenden spärlichen und fluktuierenden Besetzung des Tech-Teams, nicht empfohlen. Es ist nötig, dass jedes Teammitglied mit den Aufgaben der Rollen *Reliability Engineer*, *Gatekeeper* und *DevOps Engineer* vertraut ist und in der Lage ist, diese auszuführen und entsprechend Verantwortung zu übernehmen. So entsteht kein Problem für das Projekt, wenn spontan Entwickler und mit ihnen ihr Wissen, aus diesem ausscheiden. Es ist dringend notwendig, dass keine Wissensinseln entstehen und daher ist es auch nicht sinnvoll, Rollen zu vergeben.
An dieser Stelle sollen trotzdem noch ein paar Anmerkungen zu einigen der DevOps-Rollen gemacht werden. Die Rolle des *Gatekeepers* wird im Ist-Zustand des Projekts von den Entwicklern besetzt, da es keine Automatisierung des Deployment-Prozesses gibt. Zu den beiden Rollen *Teamleiter* und *Service-Owner* gibt es folgende Anmerkung: Beide Rollen sind im delegs-Team durch dieselbe Person besetzt, dem technischen Projektleiter. Dies widerspricht der agilen Idee, aus dem die zwei Rollen des DevOps-Teams entstanden sind. So ist vorgesehen, dass der Teamleiter das Team und dessen Arbeitsprozess vor störenden äußerlichen Einflüssen schützt und für seinen Zusammenhalt sorgt. Er wirkt kaum Autorität auf das Team aus. Der Service-Owner hingegen ist die Schnittstelle nach außen, im Fall von delegs zu den Stakeholdern. Er nimmt Anforderungen entgegen und priorisiert diese. Diese zwei Rollen in einer Person zu vereinigen, könnte zum Beispiel dazu führen, dass der Service-Owner während der laufenden Iteration dem Team neue Anforderungen hereinreichen möchte oder den Druck, den er von den Kunden und Stakeholdern bekommt, an das Team weiterreicht. Dies steht im Widerspruch zu den Aufgaben des Teamleiters, soll er doch das Team und den Iterationsprozess genau vor

solchen Einflüsse schützen. Bei delegs ist der technische Projektleiter, der diese zwei Rollen in seiner Person vereint, nur einen Tag pro Woche anwesend. Durch diese Einschränkung kann er das Team nicht kontinuierlich unterstützen. Aufgrund der personellen Fluktuation kann aber an dieser Stelle keine konkrete Empfehlung ausgesprochen werden. Die Rolle des Teamleiters müsste an den Entwickler vergeben werden, der durchschnittlich am meisten Zeit im Projekt anwesend ist, besonders auch zu den Projekt-Meetings, in denen das Projekt gesteuert wird.
Trotz der Empfehlung, die DevOps-Rollen nicht im Tech-Team zu verteilen, wird es als sinnvoll erachtet, die Rollen und ihre Aufgabenbereiche dem Team vorzustellen. Dies soll dazu beitragen, das Verständnis für DevOps, speziell den Deployment-Prozess und die darin zu erfüllenden Aufgaben, zu schärfen.

Alle Tech-Team-Mitglieder haben Zugriff auf die Zugangsdaten der Produktivdatenbank. Grund hierfür ist die gleiche Verteilung von Aufgaben und Verantworlichkeiten. Daneben ist es auch technisch begründet, denn Produktiv- und Testdatenbank werden auf demselben MySQL-Server bereitgestellt. Der allgemeine Zugriff auf die Zugangsdaten für den Produktivbetrieb ist ungünstig, er birgt Gefahren. Beispielsweise hatte Anfang April 2017 bei DigitalOcean Inc. ein "[...] Entwickler aus Versehen die produktive Datenbank gelöscht [...]. Zwar sei der Fehler nur wenige Minuten nach dem Auftreten durch Kundenberichte bemerkt worden, das Kopieren und Wiedereinspielen der Daten aus einer replizierten Datenbank habe allerdings mehrere Stunden Zeit in Anspruch genommen." (Grüner (2017))
Es scheint so, dass es aufgrund des Teamkontextes nicht möglich ist, den Zugriff für den Produktivbetrieb auf wenige Personen zu beschränken. Zumindest aber sollten Test- und Produktivbetrieb jeweils eigene Zugangsdaten bekommen.

Zu der bestehenden Wissensinsel und der Gefahr, dass sich weitere bilden, gibt es nur die Empfehlung, im Tech-Team weiterhin in Pairs zu arbeiten und diese immer wieder sinnvoll zu durchmischen. Dies ist die beste Möglichkeit, mit Wissensinseln umzugehen und deren Bildung zu vermeiden.

Die Auswertung der Struktur und Rollenverteilung im zentralen Team ist an dieser Stelle noch nicht abgeschlossen. In den weiteren Abschnitten der Auswertung des Ist-Zustands, speziell von den dort ausgesprochenen Empfehlungen, wird immer wieder auch die Arbeit des zentralen Teams betroffen sein.

6.9.2 Auswertung: Arbeit im Team

Die Arbeitsabläufe im zentralen Team sind generell gut aufgestellt. Es gibt eine enge Rückkopplung zwischen Entwurf, technischer Umsetzung, Bereitstellung, Wartung und zwischen

Tech-Team und Lehrteam. Beispielsweise kann das Tech-Team dadurch, dass dessen Mitglieder sowohl die Rolle der Entwickler als auch die Rolle der Operators einnehmen, auf kurzem Weg Feedback über entsprechende Arbeitsabläufe geben und diese optimieren. Auch das Lehrteam kann direkt Feedback zu Fehlern in der Anwendung an das Tech-Team geben. Entsprechend hat das Tech-Team die Möglichkeit, direkt Fragen zur Fachlichkeit und Benutzerfreundlichkeit (*Usability*) der Anwendung beim Lehrteam zu stellen.
Die einzige Schwierigkeit in der Zusammenarbeit ist die fehlende Übersicht des Lehrteams darüber, an welchen Aufgaben das Tech-Team gerade arbeitet bzw. welche Aufgaben für deren Iteration eingeplant sind. Anteil hat daran auch der Einsatz des Tools XPlanner. Es ist in seiner Funktionalität sehr technisch und wenig benutzerfreundlich, was dazu führt, dass das Lehrteam - als nicht technisches Personal - dort keine Einträge vornimmt. Dies führt außerdem dazu, dass vom Tech-Team sehr viel Fleißarbeit nötig ist, um vorzunehmende Einträge übersichtlich zu gestalten. Für die Umsetzung von *Continuous Everything* wird daher empfohlen, Tools auszuwählen, in denen eine Pipeline realisiert werden kann und die zugleich über eine übersichtlichere Planungssoftware verfügen. Eine Planungssoftware könnte auch als Erweiterung oder als Einbindung einer externen Anwendung in ein solches Tool integrieren werden. So wird eine engere Kopplung zwischen Planung und Umsetzung (Entwicklung) erzeugt und der Arbeitsprozess kann übersichtlicher visualisiert werden. Anforderung an entsprechende Tools ist natürlich, dass alle Mitglieder des zentralen Teams mit dem Tool arbeiten können. Dies unterstützt eine engere Zusammenarbeit. Dabei reicht ein neues Tool allein nicht aus. Auch die Art und Weise, in der es eingesetzt wird, ist ausschlaggebend. Im Ist-Zustand gibt es keine klare Linie, wie Anforderungen festgehalten werden, weder für die Struktur noch für die Art. Es sollte darauf geachtet werden, Stories fachlich und in einer vorgegebenen Struktur festzuhalten - beispielsweise in User-Stories. Auch das unerstützt die Zusammenarbeit zwischen technischen und fachlichen Team. Letzteres hat ohnehin nur die Möglichkeit, fachlich zu formulieren. Analog dazu wird empfohlen, nur noch *ein* Backlog zu verwalten und damit so umzugehen, wie Scrum es vorsieht. Anforderungen, die nicht im Backlog oder in einer Iteration liegen, können in einer Sammlung abgelegt werden, man könnte sie z.B. *Ideensammlung* nennen. So wird noch einmal ein Stück mehr Klarheit in den Entwicklungsprozess gebracht.

Bei der Arbeit des Tech-Teams gibt es allerdings Möglichkeiten, Abläufe zu optimieren. Dazu gehören unter anderem das Testen als Teil des Deployment-Prozesses und der Umgang mit dem Versionsverwaltungssystem Git.
Der Einsatz von Git ist insofern noch optimierbar, als dass *GitFlow* direkt angewendet werden sollte und nicht eine Abwandlung davon. GitFlow bietet verschiedene Vorteile, unter anderem, dass es ein im IT-Bereich bekanntes Vorgehensmodell ist. So wird es Entwicklern, die neu

ins Projekt kommen und denen GitFlow schon bekannt ist, leichter fallen, in das Projekt einzusteigen. Entwickler, denen GitFlow noch nicht bekannt ist, lernen, ganz im Sinne des *Ausbildungsprojekts* delegs, ein Vorgehensmodell kennen, das sie in andere Projekte mitnehmen können. Hinzu kommt, dass GitFlow sehr übersichtlich strukturiert ist, wodurch Fehler vermieden werden können und der Prozess einen höheren Grad an Wiederholbarkeit erreicht. Echtes GitFlow nimmt auch die Komplexität aus dem gegenwärtigen Vorgehen heraus. So kann mit GitFlow nicht in den Sonderfall, der im Abschnitt 6.6.1 Arbeitsabläufe unter Einsatz von Git beschriebenen ist, hineingelaufen werden. Daneben lässt sich GitFlow gut mit automatisiertem Deployment kombinieren.
Mangels Zeit und Fokus wird GitFlow an dieser Stelle nicht tiefergehend beschrieben. Der Leser möge sich bei Interesse der weiterführenden Literatur widmen. Zu empfehlen ist hier *Vincent Driessens* Webseite zum Thema (siehe: Driessen (2010)), der dort GitFlow als erster beschrieben hat.

Auch der Umgang mit Tests und deren Einsatz sollte dringend umstrukturiert werden. Tests werden unnötig oft wiederholt und teilweise zu ungünstigen Zeitpunkten ausgeführt. Hinzu kommt, dass die Laufzeit der *Testwoche* nicht im Verhältnis zu den in dieser Zeit ausgeführten Tests steht. Die Tests wären problemlos an einem Arbeitstag durchführbar. Die manuell durchgeführten Performance-Tests sind zwar nicht wirklich aufwendig, sollten aber trotzdem automatisiert werden. Auf lange Sicht werden damit Zeit und Arbeitsaufwand eingespart. Außerdem werden verlässlichere Ergebnisse und ein höherer Grad an Wiederholbarkeit erreicht.[54] Bei den Tests gibt es viel Potenzial, die Time to Market zu verkürzen, den Entwicklern merklich Arbeit zu ersparen und den gesamten Deployment-Prozess positiv zu beeinflussen.

Die Existenz von Feature-Branchs, die Wochen überdauern, ist fragwürdig. Nach agilem Vorgehen sollten abzuarbeitende Aufgaben in kleine, klar definierte Pakete geschnitten werden, die in maximal zwei Tagen abgearbeitet werden können. Da die lange Lebensdauer der meisten Branchs wieder den Faktoren der personellen Fluktuation und gewachsenen Codebasis geschuldet ist, könnte dies eventuell vermieden werden, wenn deutlich mehr Zeit in das *Schnüren* von Paketen und das Schätzen von Aufwänden gesteckt wird. Es könnten Code-Reviews stattfinden, um Refactoring-Aufwände besser abschätzen zu können. Auch Refactoring-Iterationen wären eine Möglichkeit, um die Qualität der Codebasis zu verbessern, von denen in der Vergangenheit schon einige im Projekt vorgenommen wurden. Allerdings leiden all diese Maßnahmen weiterhin unter der personellen Fluktuation.

[54] Auf die Automatisierung der Performance-Tests kann aufgrund des begrenzten Rahmens dieser Arbeit nicht weiter eingegangen werden.

6.9.3 Auswertung: Technische Infrastruktur

DevOps sieht beim Thema eingesetzte "Hardware" vor, dass diese so stark wie möglich vom Projektteam selbst verwaltet wird. Es soll die daraus resultierenden Freiheiten nutzen und so ein Produkt effizienter entwickeln können. Hier stellt sich die Frage, ob es sinnvoll ist, wirklich 100% der Infrastruktur selbst zu verwalten. Im Extremfall bedeutet dies, dass entsprechend geschultes Personal zur Verfügung stehen muss, um sich um die physikalischen Maschinen und ihre Konfiguration und Wartung zu kümmern. Zum Zeitpunkt, zu dem diese Arbeit geschrieben wird, scheint das Besitzen von physikalischen Servern und der damit verbundene Aufwand rückständig - unabhängig vom delegs-Projekt. Ausgenommen von dieser Einschätzung sind große Firmen, die viele verschiedene Produkte anbieten und viele Kunden betreuen, daneben Anbieter von Cloud-Services und kleine selbstverwaltete physikalische Server, die für firmeninterne Zwecke eingesetzt werden. Es gibt sicherlich noch ein paar weitere Fälle, in denen es sinnvoll ist, eigene physikalische Server zu haben, aber für eine Anwendung wie den delegs-Editor und die delegs-Webseite lohnt sich dies nicht. Für das delegs-Projekt ist es angemessener, zum Bereitstellen einer Anwendung Server- oder Cloud-Kapazitäten zu mieten. Es macht auf jeden Fall Sinn, zum einen unabhängiger vom Anbieter zu werden und zum anderen nicht den Aufwand zu haben, wirklich alles selbst zu managen. Daher wird empfohlen, auf einer Container-Plattform, wie zum Beispiel *Docker*, aufzusetzen. Damit wären die Anforderungen an den Anbieter von Hardware und die Abhängigkeit von diesem dahingehend minimiert, eine funktionierende Plattform zu stellen, die die entsprechende Laufzeitumgebung für Container bietet. Natürlich müsste der Anbieter auch Konfigurationsaufwand, wie Updates, Sicherheit, etc. übernehmen oder zumindest Zugriff auf die entsprechenden Konfigurationsmechanismen gewähren. Das Tech-Team würde die einzelnen Container selbst zusammenstellen und managen. Der Einsatz von Container-Technologie würde den Übergang in die Cloud vereinfachen. Auch für den Deployment-Prozess ergeben sich Vorteile durch den Einsatz von Containern.
Weitere Empfehlungen, den Einsatz von Container-Technologie betreffend, sind in den Abschnitten 6.9.4 Auswertung: Eingesetzte Umgebungen und 6.9.5 Auswertung: Deployment-Prozess zu finden.

In jedem Fall sollte mit ServerAnbieterA, dem Anbieter des gegenwärtig eingesetzten managed Servers, ein neuer Vertrag abgeschlossen werden, in dem genau auf dessen Aufgaben eingegangen wird und darauf, welche Reaktionszeiten von ihm einzuhalten sind.

Unabhängig davon, ob Container eingesetzt werden, sollte die Abhängigkeit von der Administration der WPS so weit wie möglich abgebaut werden. So bleibt das Tech-Team in seiner Arbeit flexibler und hat eine Instanz weniger, mit der interagiert werden muss, was den Kommunikationsaufwand abbaut, ganz im Sinne von DevOps.

Die Empfehlung, die Infrastruktur zu einem großen Teil selbst zu verwalten, birgt allerdings auch die Gefahr, dass Wissen über die eingesetzten Technologien verloren geht, siehe *personelle Fluktuation*. Es muss also trotz Selbstverwaltung und möglichst hohem Grad an Automatisierung darauf geachtet werden, dass ausscheidende Entwickler das Team ausreichend schulen, sodass diese den Projektbetrieb selbstständig aufrechterhalten können. Dies bei der Einplanung der menschlichen Ressourcen zu beachten, ist Aufgabe der Projektleitung.

Ein Thema, das sich zwischen Infrastruktur und Umgebungen bewegt, ist der Einsatz von Continuous Integration (Technologie). Bei delegs gibt es im Ist-Zustand keine CI-Unterstützung. Es wird dringend empfohlen, CI im Projekt zu realisieren. Aus DevOps-Sicht ist dies, als Teil von Continuous Deployment, unumgänglich.[55] Bereits schon der Fakt, dass dies bei der Herstellung einer fehlerfreieren Codebasis helfen kann, rechtfertigt ihren Einsatz.

6.9.4 Auswertung: Eingesetzte Umgebungen

Nun werden die in Abschnitt 6.4 beschriebenen eingesetzten Umgebungen bewertet und gegebenenfalls Empfehlungen für Veränderungen ausgesprochen. Da sich die verschiedenen Umgebungen thematisch überschneiden, werden zunächst die einzelnen Umgebungen bewertet, um danach gesammelt Empfehlungen auszusprechen.

Produktivumgebung

Die Produktivumgebung des Ist-Zustands ist, genau wie die Staging-Umgebung, sehr abhängig von der Infrastruktur auf der sie aufsetzt und die Teil von ihr ist. Dies ist eine negative Eigenschaft, die verändert werden sollte.

Staging-Umgebung

Die Staging-Umgebung ist an sich für die Entwicklung im Ist-Zustand gut aufgestellt. Sie ist sehr nah an der Produktivumgebung gehalten, was wichtig ist, um so realitätsnah wie möglich testen zu können. Es gibt allerdings zwei Dinge, die optimiert werden sollten: Die Abhängigkeit von der unterliegenden Infrastruktur und dass sie auf demselben Server-Setup aufgesetzt ist, wie auch die Produktivumgebung. Die beiden Umgebungen sollten getrennt werden, sodass das Staging nicht den Produktivbetrieb stören kann.

[55] Ausnahmen würden Projekte machen, in deren Kontext kein Continouos Deployment sinnvoll implementiert werden kann.

Entwicklungsumgebung

Es ist sinnvoll, die IDE, das GWT Plugin und die Hauptversionsnummer des eingesetzten Java-Compilers auf allen Rechnern einheitlich zu halten. Für das lokale Testen ist die Entwicklungsumgebung allerdings wenig repräsentativ. Sie ist der Produktivumgebung sehr fern, sprich, sie gleicht ihr zu wenig. Der GWT Classic Dev Mode kann nur zusammen mit einer stark veralteten Firefox-Version eingesetzt werden und auch der Super Dev Mode (SDM) gleicht der Produktivumgebung immer noch wenig. Beide Modi erzeugen nur eine lokale Instanz des Servers. Der SDM ermöglicht immerhin das Testen auf aktuellen Browsern, er wird aber kaum eingesetzt.

Allgemein

Es gibt keinen separaten Platz, an dem Backups abgelegt werden können. Im Ist-Zustand werden diese entweder auf der Produktiv- oder der Entwicklungsumgebung abgelegt. Beide Umgebungen erfüllen nicht die nötigen Anforderungen, um dort Sicherungen abzulegen.

Empfehlungen

Nun zu den Empfehlungen für die verschiedenen Umgebungen:
Es wird empfohlen, das manuelle Testen, das den Anspruch hat repräsentativ für den Produktivbetrieb zu sein, auf die Staging-Umgebung zu beschränken. Natürlich sind die Dev Modi von GWT während der Entwicklung hilfreich und sollen auch weiterhin für Debugging oder kurze Überprüfungen der Funktionalität eingesetzt werden können. Verlässliche manuelle Tests sollten aber nur über die Staging-Umgebung abgehandelt werden.

Generell ist es im Ist-Zustand schwierig, die Produktivumgebung befriedigend ähnlich genug auf den Entwicklungsumgebungen nachzustellen. Hier wird die Container-Technologie interessant. Der Einsatz einer solchen ist nicht nur aus diesem Grund sinnvoll. Sie bietet generell Vorteile für das delegs-Projekt und wird daher empfohlen. Unter anderem ermöglicht sie es, unabhängig vom Anbieter der Hardware zu werden, auf der die Anwendung aufsetzt. Für das delegs-Projekt hieße das, dass sowohl die Produktivumgebung als auch die Staging-Umgebung auf fast jeder Hardware aufgesetzt werden könnte. Einzige Voraussetzung ist, dass diese die entsprechende Container-Plattform unterstützt und sie installiert hat. Die Technologie ermöglicht auch das einfache Aufsetzen und dadurch das einfache Umziehen von Produktiv- und Staging-Umgebungen auf verschiedene Systeme, sodass diese sich nicht mehr gegenseitig stören können. Die Container würden die Komponenten wie zum Beispiel Tomcat- oder Datenbankserver enthalten. Zugleich wären sie selbst Teil der jeweiligen Umgebung, böten aber einen

Kapselungsmechanismus. Container könnten als Images hinterlegt werden, die problemlos mehrfach instanziiert werden können. Im Fall delegs könnte eine Container-Technologie dann für verschiedene Szenarien eingesetzt werden, beispielsweise:

- **Im Produktivbetrieb:**
  Ein Container mit Tomcat-Server und entsprechendem *.war Artefakt, zusammen mit einem weiteren Container, der einen MySQL-Server enthält, auf dem die Produktivdatenbank läuft.
- **Im Testbetrieb (Staging):**
  Ein Container mit Tomcat-Server und entsprechendem *.war Artefakt, zusammen mit einem weiteren Container, der einen MySQL-Server enthält, auf dem die Testdatenbank läuft.
- **Auf Entwicklungsumgebung:**
  - *Auf lokal laufendem Server für manuelle Tests, die sehr produktionsnah sind.*
    Die Entwicklungsumgebung muss nur die entsprechende Container-Plattform installiert haben und die eingesetzte delegs-Editor-Version entsprechend konfiguriert sein. Zusätzlich wird ein Container mit passend aufgesetzter Datenbank hinzugefügt.
    Für den Fall *Debugging*, ist die Entscheidung etwas schwieriger. Soll es auf Clientseite im Java-Code betrieben werden, ist der Einsatz des Classic Dev Mode (CDM) nötig, der nicht produktionsnah ist. Eventuell müssen sich die Entwickler darauf einlassen, das Debugging mit dem Super Dev Mode (SDM) zu betreiben. Dazu wäre allerdings eine Schulung in JavaScript nötig, da nicht alle Entwickler sicher genug darin sind. Zum Debugging mit dem SDM kommt aber auch noch der Arbeitsaufwand hinzu, die Transition vom generierten Javascript-Code zum entwickelten Java-Code nachzuvollziehen.
    Da Produktionsnähe aber Priorität hat, wird empfohlen, die Entwickler in JavaScript zu schulen und darauf zu achten, dass diese hauptsächlich den SDM einsetzen. So werden die Entwickler immer mehr mit dessen Einsatz und mit der Sprache vertraut und können schnell wieder eine gute Entwicklungsproduktivität erreichen. Für schwierige Probleme im Code könnte im Notfall dann immer noch auf den CDM zurückgegriffen werden.
  - *Für automatisiertes Testen:*
    Instanziieren von Containern aus Images erlaubt es, automatisierte Test durchzuführen, die manipulierend auf Datenbanken arbeiten, ohne dass die Manipulationen

zurückgesetzt werden müssen. Für jeden Test wird eine eigenen Instanz einer Testdatenbank erzeugt und diese nach dem Durchführen der Tests wieder gelöscht. Außerdem wird so ermöglicht, Tests parallel auszuführen, da diese untereinander nicht mehr über die Datenbank in Abhängigkeit stehen.

Empfohlen wird an dieser Stelle auch, einen entfernt gelegenen Ort einzurichten, an dem Backups und alte Versionen (in Form von *.war Dateien, Container-Images, etc.) abgelegt werden können. Dieser sollte physikalisch getrennt sein von den Produktiv- und Testumgebungen. Auch die Entwicklerlaptops stellen keinen validen Ort dafür dar.

6.9.5 Auswertung: Deployment-Prozess

Im Idealfall dauert ein kompletter fehlerfreier Durchlauf des Deployment-Prozesses, den eine neue Version des delegs-Editors durchläuft, zehn Arbeitstage. Das gilt insgesamt für das Durchlaufen der beiden Subprozesse Test Deployment und Produktiv Deployment und natürlich nur, wenn diese nahtlos nacheinander durchgeführt werden. In der Tabelle 6.2 Auflistung Zeitaufwand Deployment-Prozess (Ist-Zustand), werden die Verursacher des Zeitaufwands in Arbeitstagen eines kompletten Deployment-Prozessdurchlaufs aufgeführt. Es handelt sich bei den Angaben um den fehlerfreien Idealfall, also den geringstmöglichen Zeitaufwand, der anfallen kann.

Tabelle 6.2: Auflistung Zeitaufwand Deployment-Prozess (Ist-Zustand)

Zeitaufwand[56]	**Tätigkeit**
ca. 0,5 Tage*	Durchführung Test Deployment
5 Tage	Testwoche: Tests an bereitgestellter Testversion
2 Tage	Wartezeit für passenden Termin für Produktiv Deployment (wegen speziellem Power User)
ca. 0,5 Tage*	Durchführung Produktiv Deployment
ca. 8 Tage	**Gesamtaufwand Deployment-Prozess**
ca. 10 Tage	**Gesamtaufwand Deployment-Prozess, inklusive zwei Tagen Wochenende**

Da innerhalb der acht Arbeitstage immer noch ein Wochenende liegt, an dem nicht gearbeitet wird, kommen noch zwei weitere Tage hinzu. Insgesamt liegt die Time to Market dann bei mindestens zehn Tagen, was viel zu lang ist. Es wird empfohlen, den Deployment-Prozess so umzubauen, dass dieser schneller und auch zu jeder Zeit durchgeführt werden kann, ohne dabei Rücksicht auf Anwender nehmen zu müssen. Hierzu gibt es mehrere Empfehlungen für Änderungen, um den Deployment-Prozess zu optimieren:

- **Tests:**
  - *Tests aufräumen:*
    Die manuell und automatisiert durchgeführten Tests müssen effizienter durchgeführt werden. Für die verschiedenen Einsatzbereiche, in denen getestet werden muss (z.B. nach einem Commit oder nach einem Test Deployment), sollten sinnvoll zusammengestellte Test-Suites angelegt werden. Die Auswahl der in den jeweiligen Suites enthaltenen Test sollte entsprechend der Einsatzbereiche der Suites angepasst sein.
  - *Zeitraum der Testwoche verkürzen:*
    Der Zeitraum der Testwoche sollte nur so lang sein, wie es die in diesem Zeitraum durchzuführenden Tests erfordern. Er sollte präziser mit allen beteiligten Testern vereinbart werden.
- **Auf Container-Technologie aufsetzen:**
  Das Aufsetzen auf einer Container-Technologie kann diverse Vorteile bieten, einige der Vorteile werden hier kurz beschrieben:
  - *Einfacheres, schnelleres Deployment:*
    Container können einfach bereitgestellt werden und es gibt fast keinen Zeitaufwand, Container hoch- oder herunterzufahren.
  - *Paralleles Testen, ohne aufzuräumen:*
    Tests, die auf Testdatenbanken arbeiten, können auf Instanzen von Container-Images arbeiten, wodurch die Tests parallel ausgeführt werden können, ohne sich gegenseitig zu beeinflussen. Außerdem müssen die Daten der Datenbankinstanzen nach durchgeführten Tests nicht bereinigt werden. Sie werden einfach gelöscht.

[56] Reine Arbeitstage, Wochenende nicht mit eingerechnet, mit Ausnahme beim Gesamtaufwand - dort aber explizit angegeben. Ein Arbeitstag entspricht 8 Stunden. Um Personentage zu errechnen, müssen angegebene Werte mindestens mit zwei multipliziert werden, da die Arbeit bei delegs immer mindestens in Pairs durchgeführt wird.
Bei angegebenen Zeitaufwänden, die mit * versehen sind, handelt es sich um vom Autor geschätzte Werte.

- **Produktiv Deployment führt nicht zu Ausfall der Anwendung:**
  Wenn das Produktiv Deployment nicht zu einem Ausfall der Anwendung führt, wird dadurch zwar nicht unbedingt die Time to Market verkürzt. Die Anwendung bleibt aber generell auf dem Markt und entsprechend wird durch die geringeren Ausfallzeiten weniger Verlust eingefahren (auch wenn bei delegs insofern kein Verlust eingefahren werden kann).

- **Umgebungen anpassen:**
  Wie schon im Kapitel 6.9.4 Auswertung: Eingesetzte Umgebungen beschrieben.

- **Generelles Überarbeiten des Deployment-Prozesses:**
  Momentan werden alle Arbeiten des Deployment-Prozesses manuell durchgeführt. Dies kostet nicht nur viel Zeit, es leidet auch die Wiederholbarkeit darunter. Denn der Prozess wird zwar durch die Entwickler anhand eines Protokolls durchgeführt, das sie durch den Prozess leitet. Es besteht dabei aber immer die Gefahr, dass die Entwickler Schritte des Prozesses nicht wie im Protokoll beschrieben ausführen, zusätzliche Schritte durchführen und diese nicht dokumentieren oder die Durchführung fehlerbehaftet ist.
  Es kann am Deployment-Prozess vieles automatisiert werden. Dies würde den oben genannten Problemen entgegenwirken und die Durchführung des Prozesses beschleunigen. Daher sollte er entsprechend neu überdacht und ausgearbeitet werden.

Es stellt sich die Frage, ob *echtes* Continuous Deployment für das delegs-Projekt zu empfehlen ist. Die Antwort darauf lautet klar *nein*, denn der durchgehenden Automatisierung des gesamten Deployments stehen wichtige Gründe entgegen. Einer ist die gewachsene Codebasis, in Verbindung mit der personellen Fluktuation: Es gibt fast keine Entwickler, die lange genug im Projekt sind, um die Codebasis in ihrer Gesamtheit zu durchdringen und zu überblicken. Deshalb ist es nötig, nach dem Einbau von Features oder Bugfixes immer einen Großteil der Funktionalität des delegs-Editors *manuell* zu testen. Und so ist ein Deployment zu Testzwecken nötig, das den automatisierten Fluss der Deployment-Pipeline unterbricht. Um entsprechende aufwändige manuelle Tests effektiv durchzuführen, macht es natürlich Sinn, diese jeweils nur einmal, nämlich bei der Vorbereitung einer neuen Produktivversion für eine Menge von Features und Bugfixes durchzuführen - nach dem Test Deployment.
Ein anderer Grund, der gegen echtes Continuous Deployment spricht, ist, dass es von außen keine Not gibt, so schnell wie möglich neue Features bereitzustellen. *Außen* meint hier die Stakeholder und generell den Kontext, in dem der delegs-Editor eingesetzt wird. Es ist viel wichtiger, eine neue Version so fehlerfrei wie möglich bereitzustellen. Nun könnte so argumentiert werden, dass ein Fehler, der in eine neue Version eingebaut wurde, schnell mit der

Bereitstellung einer neuen Version behoben werden könnte. Denn dank echtem Continuous Deployment würde ein neues Deployment innerhalb sehr kurzer Zeit durchgeführt werden. Aber auch das greift bei delegs wieder nicht, aufgrund der gewachsenen Codebasis, die einfach zu sehr dazu verleitet, neue Fehler einzubauen. Außerdem ist nicht garantiert, dass bei der spärlichen personellen Besetzung im Projekt ein Entwickler vor Ort ist, um den Fehler in kurzer Zeit zu beheben.
Es wird also empfohlen, auf *echtes* Continuous Deployment zu verzichten.

Einwand des technischen Projektleiters zur Automatisierung

Nun noch zum Einwand des technischen Projektleiters, delegs sei ein Ausbildungsprojekt und eine Automatisierung würde bewirken, dass die technischen Mitarbeiter des Projekts weniger über den Deployment-Prozess lernen. Es wäre an dieser Stelle interessant, nach Quellen zu suchen, die sich wissenschaftlich mit dem Thema auseinandersetzen: Erfahren Auszubildende eine bessere Lehre, wenn sie in ihrer Ausbildungszeit Arbeiten, die heutzutage mit Hilfe von Maschinen (teil-)automatisiert ausgeführt werden, *per Hand*, also manuell, ausführen müssen? Ein Beispiel aus dem Handwerk wäre die Lehre bei einem Schlosser, während der dem Auszubildenden die Aufgabe gestellt wird, einen Metallrohling per Hand, mit einer Metallfeile, blank zu feilen und danach per Hand, mit einem Gewindebohrer, Gewinde in den Rohling zu schneiden. Leider weicht das Thema zu sehr von der Ausrichtung dieser Arbeit ab und kann daher nicht weiter vertieft werden.
Zur Automatisierung des Deployment-Prozesses bleibt zu sagen, dass dabei zwar fast alle Arbeiten automatisch durchgeführt werden. Die Deployment-Pipeline wird aber auch ausreichend per Text und Grafik in dieser Arbeit beschrieben, sodass der Prozess anhand der Beschreibung manuell durchgeführt werden kann. Es wäre also möglich, einem Mitarbeiter die Aufgabe zu geben, den Deployment-Prozess anhand der Beschreibung manuell auszuführen. Dieser kann dann lernen, welche Arbeiten nötig sind, um ein komplettes Deployment durchzuführen. Die beschreibenden Dokumente müssten hierzu nur im Projekt-Repository abgelegt werden.

7 Anforderungen

In diesem Kapitel werden die Anforderungen aufgeführt, die an die Lösungsstrategie und damit an die Integration von DevOps in das delegs-Projekt gestellt werden. Die Tabelle 7.1 führt die Anforderungen auf und stellt ihnen dort jeweils eine Spalte für *Id*, *Name* und *Zusammenfassung*, *Beschreibung* und *Kategorie*.
In der Spalte *Kategorie*, wird die Herkunft der jeweiligen Anforderung angegeben. Es gibt dabei drei Kategorien von Quellen für Anforderungen:

- aus dem Projekt stammend (z.B. von einem Stakeholder)
- aus den Interviews mit dem zentralen Team stammend
  Hierbei handelt es sich um die vorläufige Anforderungen, die im Abschnitt 6.8.4, Unterabschnitt *Anforderungen aus Interviews* zu finden sind.
- aus DevOps stammend
  Diese ergeben sich aus der Philosophie und den Vorgehensweisen von DevOps.

Anforderungen aus verschiedenen Quellen, die sich inhaltlich überschneiden, werden zusammengefasst. Dies ist daran zu erkennen, dass im Tabellenfeld *Kategorie* mehrere Einträge zu finden sind. Ist eine Anforderung unter anderem aus einer vorläufigen Anforderung entstanden, ist die Id der entsprechenden vorläufigen Anforderung im Kategorie-Feld eingetragen. In der Anforderungstabelle gibt es keine Priorisierung, da dies nicht sinnvoll erscheint. Alle aufgeführten Anforderungen haben ihre Berechtigung für das Ziel, DevOps in das delegs-Projekt zu integrieren. Die einzige Ausnahme macht die Anforderung *A01*, die durch den Projektleiter an diese Arbeit gestellt wurde. Genaueres hierzu im Unterabschnitt *Erläuterung und Abgrenzung* weiter unten.

Tabelle 7.1: Anforderungen

Id	Name, Zusammenfassung	Beschreibung	Kategorie
A01	Deployment wird bruchlos durchgeführt	Es ist nicht mehr nötig, Anwender über den bevorstehenden Prozess des Produktiv Deployments zu informieren, da das Deployment bruchlos stattfindet. Das Bereitstellen einer neuen Produktivversion kann bei laufendem Betrieb der aktuellen Version durchgeführt werden, ohne dass dem Anwender dadurch Daten verloren gehen oder die Anwendung ausfällt.	Projektleiter Herr Gryczan & Interviews (Ids: TA11, TA12)
A02	Beschriebene, einsehbare Deployment-Pipeline	Die Deployment-Pipeline muss für die Mitglieder des Tech-Teams klar definiert und beschrieben sein. Das Beschriebene ist immer einsehbar.	DevOps
A03	Ein Deployment ist jederzeit möglich	Der Aufwand, der durch das Durchführen des Deployment-Prozesses entsteht, ist unabhängig von Größe und Anzahl bereitzustellender Features immer insignifikant genug, um vernachlässigbar zu sein. Der aus dem Deployment entstehende Aufwand beeinflusst nicht die Entscheidung, ob ein solches durchgeführt werden soll. Somit ist ein Deployment jederzeit durchführbar.	DevOps & Interviews (Id: TA06)
A04	Rückkopplung durch Deployment-Pipeline	Die Deployment-Pipeline soll für die Mitglieder des Tech-Teams Rückmeldungen über Zustand und Verlauf ihrer einzelnen Schritte ausgeben. Sie soll zu jederzeit entsprechende Informationen liefern, unabhängig davon, ob der Prozess erfolgreich verläuft oder nicht.	DevOps & Interviews (Id: TA03)
A05	Klarheit über Vorgehen	Allen Teammitgliedern ist jederzeit einsehbar, welches weitere Vorgehen für die Entwicklung am delegs-Editor geplant ist.	DevOps & Interviews (Id: TA14)

Tabelle 7.1: Fortsetzung: Anforderungen

Id	Name, Zusammenfassung	Beschreibung	Kategorie
A06	Kontrolle über Infrastruktur	Die unterliegende Infrastruktur soll vom Tech-Team selbstständig verwaltet werden. Dabei geht es nicht darum, die Hardware selbst zu stellen, sondern die darauf laufende Software, inkl. Betriebssystem und Software-Server, selbst zu verwalten und up to date zu halten.	DevOps & Interviews (Id: TA01)
A07	Infrastructure-as-Code	Infrastruktur-Code wird durch ein Versionsverwaltungssystem überwacht.	DevOps
A08	Einsatz von Skripten	Skripte werden überall dort im Deployment-Prozess eingesetzt, wo sonst manuell auf Konsolen gearbeitet oder Tools manuell bedient werden würden. So wird ein höherer Grad an Automatisierung und Wiederholbarkeit erreicht.	DevOps
A09	Strikte Wiederholbarkeit	Eine strikte Wiederholbarkeit des Deployment-Prozesses wird durch ein hohes Maß an Automatisierung erreicht. Der Einsatz von Skripten oder Tools und deren Konfiguration während der Einsätze werden aufgezeichnet.	DevOps

Tabelle 7.1: Fortsetzung: Anforderungen

Id	Name, Zusammen-fassung	Beschreibung	Kategorie
A10	Continuous Integration	Integration von Code in die Codebasis soll automatisch stattfinden, sodass Änderungen an der Codebasis nur noch dem Versionsverwaltungs-system übergeben werden müssen. Das Bauen der veränderten Codebasis und das Durchführen von Unit- und Integrationstests auf ihr passiert automa-tisch. Außerdem wird sie automatisch auf das Ein-halten von Code-Konventionen überprüft. Verläuft das Bauen oder ein Testlauf fehlerhaft, wird der Ersteller des Commits darüber informiert und sein Commit abgelehnt. Die fehlerhafte Codebasis wird auf einen dafür neu angelegten Branch übertragen. Es gibt einen Haupt-Branch auf dem die Codebasis immer fehlerfrei ist.	DevOps & Interviews (Id: TA08)
A11	Continuous Delivery	Nach der erfolgreich durchgeführten Integration von Code wird die Codebasis automatisch ge-baut. Das daraus entstehende Artefakt wird den Akzeptanztests und wenn vorgesehen Performance-Tests in einer Staging-Umgebung automatisiert unterzogen. Diese ist so nah wie möglich der Produktivumgebung nachempfunden. Treten während des Prozesses Fehler auf, wird das Tech-Team darüber informiert und der Deployment-Prozess abgebrochen.	DevOps
A12	Continuous Deployment	Das durch Continuous Delivery erstellte Artefakt wird automatisch in den Produktivbetrieb bereit-gestellt.	DevOps

Tabelle 7.1: Fortsetzung: Anforderungen

Id	Name, Zusammenfassung	Beschreibung	Kategorie
A13	Entwicklungsumgebung gleicht Produktivumgebung	Die Entwicklungsumgebung ist der Produktivumgebung so nah wie möglich nachempfunden. Dies bezieht sich nur auf das Testen der Anwendung. Wird die Anwendung in der Entwicklungsumgebung testweise bereitgestellt, ist ihr durch die Entwicklungsumgebung beeinflusstes Verhalten nur so minimal abweichend von einer Anwendung, die in der Produktivumgebung läuft, dass es vernachlässigt werden kann.	DevOps & Interviews (Id: TA04)
A14	Provisionierung Testumgebung	Die eingesetzte Testumgebung wird automatisiert provisioniert und konfiguriert.	DevOps
A15	Testumgebung gleicht Produktivumgebung	Die Testumgebung ist der Produktivumgebung so nah wie möglich nachempfunden. Wird die Anwendung in der Testumgebung bereitgestellt, ist ihr durch die Testumgebung beeinflusstes Verhalten nur so minimal abweichend von einer Anwendung, die in der Produktivumgebung läuft, dass es vernachlässigt werden kann.	DevOps & Interviews (Id: TA05)
A16	Provisionierung Produktivumgebung	Die eingesetzte Produktivumgebung wird automatisiert provisioniert und konfiguriert.	DevOps

Tabelle 7.1: Fortsetzung: Anforderungen

Id	Name, Zusammenfassung	Beschreibung	Kategorie
A17	Mail-Generierung nach Produktiv Deployment	Nach einem erfolgreichen Produktiv Deployment wird automatisch eine E-Mail erzeugt, in der Namen und Ids der umgesetzten Anforderungen / Features / Bugfixes enthalten sind. Die Mail wird nicht automatisch verschickt, sondern wird manuell überarbeitet und manuell an den speziellen Power User und die restlichen Stakeholder verschickt.	Interviews (Id: TA09)
A18	Geringer Zeitaufwand für automatisierte Tests	Der Zeitaufwand, der durch das Durchführen automatisierter Tests entsteht, ist immer insignifikant genug, um vernachlässigbar zu sein. Entwickler, die automatisierte Tests anstoßen, werden dadurch nicht in ihrer Arbeit aufgehalten oder ausgebremst, es sei denn, sie sind abhängig von deren Ergebnis.	DevOps & Interviews (Id: TA13)

Erläuterung und Abgrenzung

Im Folgenden wird, nach Kategorien aufgeteilt, auf Anforderungen eingegangen, für die ein Erklärungsbedarf besteht. Unter anderem werden hier auch die vorläufigen Anforderungen aufgeführt, die nicht als konkrete Anforderungen an die Lösungsstrategie aufgenommen wurden und es wird erklärt, warum dies der Fall ist.

- **Anforderungen aus dem Projekt selbst:**
  Aus dieser Kategorie gibt es nur eine einzige Anforderung, sie wurde vom Projektleiter Herrn Gryczan gestellt und wird mit hoher Priorität aufgenommen. Die Anforderung trägt den Namen *Deployment wird bruchlos durchgeführt* und die Id *A01.*

- **Anforderungen extrahiert aus den Interviews:**
  Die folgenden vorläufigen Anforderungen wurden nicht in die Liste der Anforderungen an die Lösungsstrategie aufgenommen:

- Id: TA02 - *Vermeidung von Wissensinseln*
  Die Arbeit im Tech-Team findet bereits in Pairs statt. Diese werden immer wieder durchmischt, teilweise auch *während* der Umsetzung eines Features oder *während* des Behebens eines Bugs. Daher ist diese vorläufige Anforderung schon erfüllt.
- Id: TA07 - *Deployments finden regelmäßig statt*
  Aufgrund der Teamstruktur des Tech-Teams und dessen personeller Fluktuation erscheint es nicht sinnvoll, ein regelmäßiges Deployment vorzunehmen. Wichtiger ist es, die Möglichkeit zu bieten, jederzeit ein Deployment anstoßen zu können. Eine entsprechende Anforderung ist in Tabelle 7.1 Anforderungen unter Id: A03 - *Ein Deployment ist jederzeit möglich* zu finden.
- Id: TA10 - *Keine Automatisierung des Deployment-Prozesses*
  Im Interview mit dem technischen Projektleiter, Herrn Jörn Koch, hat dieser geäußert, dass aus seiner Sicht "die fehlende Automatisierung [des Deployment-Prozesses] [...] kein Problem [ist], sondern [...] delegs als Ausbildungsprojekt [den Tech-Team-Mitgliedern] eine gute Gelegenheit [bietet] zu lernen, was alles im Rahmen eines Deployments zu tun ist und wie wichtig dabei Sorgfalt ist [...]."[57]
  Dies wurde nicht als konkrete Anforderung ausgesprochen, allerdings als vorläufige Anforderung aufgenommen. Sie als Anforderung an die Lösungsstrategie aufzunehmen steht in direkter Konkurrenz zu der Einführung von DevOps. Wie in den vorherigen Kapiteln beschrieben, ist Automatisierung ein wichtiges Mittel, das von DevOps eingesetzt wird. Bei der Entwicklung einer Lösungsstrategie soll die Anmerkung von Herrn Koch dennoch nicht direkt ausgeschlossen, sondern in die Betrachtung mit einbezogen werden.

- **Anforderungen aus DevOps:**
  Auch einige Anforderungen, die aus DevOps extrahiert wurden, brauchen eine genauere Betrachtung:
  - Die Anforderungen *A10*, *A11* und *A12* realisieren *Continuous Everything* und betreffen den kompletten Deployment-Prozess.
    Zusammen stehen sie für ein automatisiertes Bereitstellen von Änderungen an der Anwendung, ab dem Moment, da diese Änderungen dem Versionsverwaltungssystem übergeben werden. Zwar ist Continuous Everything und die Automatisierung, die es mitbringt, für DevOps ein wichtiges Werkzeug zum Erreichen seiner Ziele.

[57] Zitiert aus dem Interview mit Herrn J. Koch (technischer Projektleiter), siehe Anhang e. Interviews mit dem zentralen Team

Es muss sich aber an die Gegebenheiten des jeweiligen Projekts anpassen. So ist es nicht immer sinnvoll, die gängigen Werkzeuge um jeden Preis einzusetzen.
Es wird an dieser Stelle entschieden, die Anforderungen A10, A11 und A12, so wie in Tabelle 7.1 Anforderungen beschrieben aufzunehmen, da sie für DevOps ein so wichtiges Werkzeug darstellen. Es ist klar, dass es nicht sinnvoll ist, diese vollständig für die Entwicklung am delegs-Editor umzusetzen. Dies wurde bereits im Abschnitt 6.9.5 Auswertung: Deployment-Prozess der Analyse festgestellt. Bei der Entwicklung der Lösungsstrategie wird versucht, die Anforderungen umzusetzen, soweit es für die Gegebenheiten des delegs-Projekts sinnvoll ist.

– Die oben schon erwähnte Anforderung *A10* betrifft Continuous Integration (CI) und braucht noch eine genauere Erklärung.
Im Projekt wird im Ist-Zustand unter Einsatz von Feature-Branchs gearbeitet. Sowohl neue Features als auch Bugfixes werden auf solchen Branchs entwickelt. Würde man nun zu *reiner* CI wechseln, würde das bedeuten, "[...] dass jeder [Entwickler] jeden Tag Commits auf die Mainline beitragen müsste. Sobald Feature-Branchs nicht kürzer als einen Tag bestehen, ist das etwas anderes als CI."[58] Feature-Branchs existieren im Projekt aber länger als einen Tag und sind wichtig für die Arbeit im Projektkontext, unter anderem wegen der personellen Fluktuation. Sie überdauern manchmal Wochen, da die Arbeit an Features oder Bugfixes sich über Wochen hinziehen kann. Die lange Lebensdauer der Branchs kann auch durch Probleme bedingt sein, deren Bearbeitung von größerer Komplexität ist (Stichwort *gewachsene Codebasis*). Die genannten Gründe für die lange Lebensdauer der Branchs sind im Projekt nicht zu umgehen. Für Feature-Branchs spricht zusätzlich noch, dass sie einen flexiblen Umgang mit der Auswahl von Features für ein kommendes Release ermöglichen. Martin Fowler nennt dies *Cherry Picking*. Um eine Feature-Auswahl zu ermöglichen, könnten natürlich auch die von Fowler genannten *Feature-Toggles* eingesetzt werden.[59] Für diese braucht es aber klare Entwürfe, unter anderem in Bezug auf die architektonische Modularisierung der Software. Das Projekt scheint dazu aber schon zu weit vorangeschritten zu sein. Die *gewachsene Codebasis* des Projekts würde ihren Einsatz schwierig machen und

[58] Zitiert aus Fowler (2009). *Mainline* meint hier Master- bzw. Development-Branch.
[59] Fowlers Aussagen zu *Cherry Picking* und *Feature-Toggles* und *Continuous Building* sind auch im Text der Quelle Fowler (2009) zu finden, aus der in diesem Kapitel bereits zitiert wurde.
Auf *Feature-Toggles* kann im Rahmen dieser Arbeit nicht eingegangen werden. Der interssierte Leser möge sich der einschlägigen Fachliteratur widmen. Auf der Webseite von *Martin Fowler* (siehe Fowler (2017)) gibt es gleich zwei Artikel zu dem Thema.

der Code würde dadurch eine noch höhere Komplexitätsstufe erreichen. An dieser Stelle wird also eine klare Entscheidung für Feature-Branchs getroffen, trotz ihrer langen Lebensdauer und gegen Feature-Toggles. Dies führt dazu, dass im Projekt kein *reines* CI eingesetzt wird, sondern eine abgeschwächte Form, die Fowler *Continuous Building (CB)* nennt.[60] Bei CB stößt ein Server nach jedem Commit auf einen Feature-Branch automatisch einen Build und auch Tests an. So wird gewährleistet, dass die Codebasis nach dem Einchecken von Code einen gewissen Grad an Fehlerfreiheit erreicht. Die Ausführung der Tests (und des Bauens der Anwendung) auf dem Server bietet den Vorteil, unter weniger Zeitaufwand durchgeführt werden zu können, wie es auch schon von CI bekannt ist. Dies wirkt sich günstig auf die Arbeitszeit der Entwickler aus, solange sie nicht abhängig vom Ergebnis der Tests sind. Trotzdem Fowler das Vorgehen als *Continuous Building* bezeichnet, wird in dieser Arbeit weiterhin von *CI* gesprochen, da CB einfach nur eine speziellere Form von CI ist.

Die folgenden Anforderungen, die sich aus DevOps extrahieren lassen, wurden nicht in die Liste der Anforderungen an die Lösungsstrategie aufgenommen:

- *Keine Trennung der Aufgabenbereiche von Entwicklung und Operations*:
  Entwicklung und Operations arbeiten zusammen als ein Team an der Entwicklung der Anwendung und sind auch zusammen für die Bereitstellung der Anwendung verantwortlich.
  Eine entsprechende Anforderung wurde nicht aufgenommen, da sie schon weitestgehend erfüllt ist. Im Tech-Team gibt es (fast) keine Aufteilung in Development und Operations. Die einzige Ausnahme macht die nicht komplett selbstverwaltete unterliegende Infrastruktur. Speziell zu diesem Thema ist aber eine Anforderung in Tabelle 7.1 unter Id: A06 - *Kontrolle über Infrastruktur* aufgenommen worden.
- *Einführung der Rollen aus DevOps:*
  Es werden die durch DevOps vorgegebenen Rollen im Tech-Team vergeben.
  Grund, warum diese Anforderung nicht aufgenommen wurde, ist die personelle Fluktuation des Tech-Teams. Rollen müssten mehrfach und immer wieder neu verteilt werden. Im Ist-Zustand kann schon jetzt jedes Teammitglied jede Aufgabe übernehmen. Es gibt also eine gemeinsame Verantwortung für Gate-Keeping, Reliability und Deployment-Tools.
  Es wäre aber im Sinne des Ausbildungsprojekts delegs sicherlich sinnvoll, dem

[60] Fowlers Aussagen zu *Continuous Building* sind auch im Text Fowler (2009) zu finden, aus dem in diesem Kapitel bereits zitiert wurde.

Tech-Team die von DevOps vorgesehenen Rollen und ihre Aufgabenbereiche vorzustellen.

- *Aufnahme der Anforderungen vom Operationsteam:*
  Das Operationsteam stellt Anforderungen an die Entwicklung der Anwendung bezüglich des Deployment-Prozesses.
  Diese Anforderung ist bereits erfüllt. Wie bereits oben genannt, übernimmt das Tech-Team bereits die Verantwortungen aus dem Operationsbereich und berücksichtigt diese automatisch.

8 Lösungsstrategie (Soll-Zustand)

Für die Integration von DevOps in das delegs-Projekt gibt es nicht *die eine* richtige Lösung. In diesem Kapitel wird eine weitestgehend vollständige Lösungsstrategie beschrieben, die den Fokus hat, die Time to Market des delegs-Editors stark zu verkürzen. Dies wird erreicht durch die Überarbeitung des Entwicklungs- und Deployment-Prozesses. Es wird eine neue Deployment-Pipeline vorgestellt, die zu großen Teilen automatisiert ist. Außerdem werden Optimierungen der Arbeitsabläufe beschrieben, speziell das Testen ist hiervon betroffen.

In diesem Kapitel werden nun zuerst Änderungen am zentralen Team, dessen Arbeit und Arbeitsabläufen vorgestellt. Der Abschnitt danach befasst sich mit Änderungen an der eingesetzten Infrastruktur und den eingesetzten Umgebungen. Dann wird das neue verteilte System vorgestellt, das die Anforderung A01 - *Deployment wird bruchlos durchgeführt* umsetzt. Die Anforderung beeinflusst die Deployment-Pipeline, die im letzten Abschnitt beschrieben ist, maßgeblich.

8.1 Struktur und Arbeit des zentralen Teams

In diesem Abschnitt wird hauptsächlich auf die Arbeit des zentralen Teams und dessen Arbeitsabläufe eingegangen. Da es, analog zur Bewertung und den Empfehlungen aus Abschnitt 6.9.1 Auswertung: Struktur und Aufgabenverteilung des zentralen Teams, nur wenige Maßnahmen zur Änderung der Teamstruktur gibt, fällt der entsprechende Teil kurz aus.

Teamstruktur und Rollenverteilung

Eine Veränderung an der Teamstruktur wird nicht vorgenommen. Aus den im Abschnitt 6.9.1 beschriebenen Gründen, wird das DevOps-Rollenmodell nicht eingeführt.
Eine Ausnahme macht die Rolle des *Gatekeepers*, der während des Deployment-Prozesses gebraucht und teilweise durch Mitglieder des Tech-Teams besetzt wird. Wann und wo genau menschliche Gatekeeper eingesetzt werden, ist dem Abschnitt 8.4 Die neue Deployment-Pipeline zu entnehmen. Die Rolle wird nicht an eine spezifische Person des Tech-Teams vergeben. Wie schon im Ist-Zustand müssen alle Teammitglieder alle anfallenden Verantwort-

lichkeiten und Aufgaben übernehmen. So verhält es sich auch mit dieser Rolle, die bei Bedarf von den anwesenden Kräften übernommen wird.

Trotzdem das DevOps-Rollenmodell nicht explizit eingeführt wird, werden dessen Rollen *Reliability Engineer*, *Gatekeeper* und *DevOps Engineer* dem Tech-Team vorgestellt, um eine klarere Idee von der Verteilung der verschiedenen Aufgaben zu vermitteln.
Es kann auch eine Diskussion über die schon beschriebene Problematik mit den beiden Rollen *Teamleiter* und *Service-Owner* angeregt werden. Allerdings wird kaum etwas an dem Problem zu ändern sein, daher bliebt dies nur ein Hinweis.

Arbeit und Arbeitsabläufe

Die meisten Änderungen an der Arbeit und den Arbeitsabläufen betreffen das Tech-Team.
Das Verringern der Abhängigkeiten vom Hardware-Anbieter und von der Administration der WPS erweitert die Aufgaben des Tech-Teams im Operations-Bereich ein wenig. Genaueres zur Erweiterung des Aufgabenbereichs des Tech-Teams ist in den folgenden Abschnitten 8.2 Infrastruktur & Umgebungen, 8.3 Bruchfreies Deployment und 8.4 Die neue Deployment-Pipeline beschrieben.
Natürlich wird weiterhin in Pairs gearbeitet, um Wissensinseln zu vermeiden und abzubauen.

Veränderungen an den Arbeitsabläufen im Tech-Team sind hauptsächlich durch die Einführung einer neuen Deployment-Pipeline bedingt, in der viel automatisiert wird und von den Veränderungen am verteilten System.
Aus den im Kapitel 6.9.2 Auswertung: Arbeit im Team genannten Gründen wird nun GitFlow konsistent eingesetzt. Aber auch die neue Pipeline baut auf GitFlow auf, weshalb dessen korrekter Einsatz nötig ist.
Auch verändert sich das Vorgehen beim Testen. Tests sind nun effizienter strukturiert, was sich nicht nur auf die Arbeitsabläufe des Tech-Teams, sondern auch ein wenig auf die des Lehrteams und des speziellen Power Users, der in das Testen involviert ist, auswirkt.
Die Details zur neuen Pipeline, dem Vorgehen beim Testen und dem neuen verteilten System sind den Kapiteln 8.3 Bruchfreies Deployment und 8.4 Die neue Deployment-Pipeline zu entnehmen.

Im neuen Konzept sind Test- und Produktivbetrieb soweit separiert, dass diese jeweils eigene Zugangsdaten bekommen. Die Daten für den Produktivbetrieb werden in einer eigenen Datei verschlüsselt gelagert, die entsprechend und klar verständlich benannt worden ist. Beim Entschlüsseln der Daten durch ein Teammitglied ist dadurch klar unterstrichen, dass ein Zugriff auf den Produktivbetrieb bevorsteht. So werden Fehler im Umgang mit dem Produktivsystem unwahrscheinlicher.

Eine Änderung, die das komplette zentrale Team betrifft, ist der Einsatz eines Tools zur Planung und Durchführung von Iterationen. An das Tool ist die Anforderung gestellt, dass es eine UI haben muss, die vom Lehrteam - also von nicht technischem Personal - bedient werden kann. Außerdem muss es mit den eingesetzten Tools der neuen Deployment-Pipeline und der eingesetzten Infrastruktur und Umgebungen kompatibel sein, wie in den folgenden Abschnitten beschrieben. Anforderungen werden fachlich in Form von User-Stories sowohl vom Tech-Team als auch vom Lehrteam festgehalten. Es gibt ein klar definiertes, nach Scrum eingesetztes Backlog.
All das trägt zur Erfüllung der Anforderung A05 - *Klarheit über Vorgehen* bei.
Aufgrund des begrenzten Rahmens dieser Arbeit kann nicht weiter auf die Auswahl des entsprechenden Tools eingegangen werden.

Wie bereits als Empfehlung ausgesprochen, soll mehr Zeit in das *Schnüren* von Paketen und das Schätzen von Aufwänden gesteckt werden. Außerdem sollen weiterhin Refactoring-Iterationen durchgeführt werden, beides mit dem Ziel die Qualität der Codebasis zu verbessern.

8.2 Infrastruktur & Umgebungen

Aufgrund der beschriebenen Problematik mit dem Anbieter, der den eingesetzten managed Hardware-Server stellt, aber zugleich auch im Sinne von DevOps, werden die Systemkomponenten des delegs-Editors in jeweils eigenständigen Containern organisiert und bereitgestellt.[61] Zudem sind Produktiv- und Staging-Umgebung in unabhängigen Orchestrationen organisiert. Als Container-Plattform wird *Docker* von der Firma *Docker Inc.* eingesetzt.

Durch den Einsatz der Container-Technologie wird ein hoher Grad an Unabhängigkeit von der unterliegenden Hardware (z.B. *managed virtual Server* oder *Cloud*), ihrer Konfiguration und zugleich von deren Anbieter erreicht.[62] Außerdem entsteht so eine klare Kapselung der einzelnen Komponenten und dadurch mehr Flexibilität bei deren Einsatz. Durch die Unabhängigkeit von der Hardware gibt es weniger und dafür klarere Anforderungen an diese. Hauptanforderung ist, dass die Hardware die Docker-Engine unterstützt. Weitere sind die gängigen Anforderungen wie *Verfügbarkeit*, *Sicherheit* etc. Sie fallen alle in den Aufgabenbereich des Anbieters der Hardware. Die Aufgaben des Tech-Teams sind die Container und alles innerhalb dieser, sowie ihre Orchestrierung und Bereitstellung und das Management der

[61] Auf die Komponenten des delegs-Systems im Soll-Zustand wird im Unterabschnitt 8.3.2 Das neue verteilte System genauer eingegangen.
[62] Dies erfüllt die Anforderung A06 - *Kontrolle über Infrastruktur.*
Genaueres zur Anforderung im Kapitel 7 Anforderungen.

Docker-Engine.
Mehr Flexibilität wird durch den Einsatz von Containern in verschiedenen Bereichen erreicht. Zum einen können die einzelnen Container und dadurch die darin enthaltenen Komponenten beliebig umgezogen werden. Zum anderen erlaubt die Container-Technologie, dass Container beliebig oft instanziiert werden. Dies kann beispielsweise genutzt werden, um ein System horizontal zu skalieren oder um Datenbanken zu Testzwecken einzusetzen, ohne auf eine Änderung am Datenbestand durch die Tests Rücksicht nehmen zu müssen. Der wohl wichtigste Vorteil der Container-Technologie ist, wie bereits an anderer Stelle erwähnt, die starke Vereinfachung des Deployments.
Die Wahl ist auf *Docker* als Container-Plattform gefallen, da diese die verbreitetste Plattform ist - es wird also auf eine ausgereifte Technologie aufgesetzt. Docker kann in viele andere Tools integriert werden bzw. von diesen genutzt werden. Außerdem gibt es aufgrund ihrer weiten Verbreitung viele Quellen im Internet, die Informationen bieten. Es ist wahrscheinlich, dass Probleme, die mit Docker im Projektbetrieb aufkommen werden, schon an anderer Stelle aufgetreten, gelöst und dokumentiert worden sind. Ein weiterer Vorteil von Docker ist, dass es ein eigenes umfangreiches Plugin-System zum Erweitern anbietet, um es an die Projektbedürfnisse anzupassen.

Die Entwicklungsumgebungen im Tech-Team werden nicht verändert, sondern nur um die Docker-Engine erweitert, sobald der Wechsel nach Docker erfolgt. Durch ihren Einsatz wird zum Testen lokal ein Software-Server hochgefahren, der der Produktivumgebung sehr ähnelt, um auf diesem manuelle Tests durchzuführen.[63] Für das Debugging muss eine Schulung in JavaScript angesetzt werden, um dann hauptsächlich mit dem Super Dev Mode zu arbeiten, Nur in schwierigen Fällen wird auf den Classic Dev Mode zurückgegriffen.

Zum Thema ServerAnbieterA gibt es verschiedene Änderungen. Der Vertrag, der den managed virtual Server betrifft, wird so angepasst, dass dieser klar definiert, welche Aufgaben ServerAnbieterA übernimmt und welche Reaktionszeiten dabei eingehalten werden müssen. Sollte das nicht möglich sein, muss ein alternativer Anbieter gefunden und zu diesem gewechselt werden.[64] Eine weitere Anforderung an ServerAnbieterA ist, dass die Docker-Engine auf dem gegenwärtig gemieteten Server bereitgestellt wird oder aber ein Umzug auf einen anderen von ServerAnbieterA bereitgestellten Server (mit Docker-Engine) ermöglicht wird.[65] Ersteres ist grundsätzlich zu bevorzugen. Es ist sehr unwahrscheinlich, dass im letzteren Fall

[63] Dies erfüllt die Anforderung A13 - *Entwicklungsumgebung gleicht Produktivumgebung.* Genaueres zur Anforderung im Kapitel 7 Anforderungen.
[64] Zu einem anderen Anbieter zu wechseln, schließt den Wechsel in eine Cloud mit ein. Mehr zum Thema *Cloud* siehe im Unterabschnitt 8.2.2 Sprung in die Cloud weiter unten.
[65] Es wäre gut möglich, dass ServerAnbieterA für ihre virtuellen Server *patched Kernels* verwendet, die nicht mit der Docker-Engine kompatibel sind.

der neue Server nicht neu aufgesetzt werden muss. Daher kann ebenso zu einem anderen Anbieter gewechselt werden, der einen besseren Service bietet. Kann ServerAnbieterA keine Docker-Engine zur Verfügung stellen, muss in jedem Fall eine Alternative für die Infrastruktur gefunden werden.

Die Kommunikation und Interaktion mit dem jeweiligen Hardware-Anbieter wird in jedem Fall direkt vom Tech-Team übernommen, ohne Umwege über die Administration der WPS.

Wie für alle Bereiche gilt auch für diesen, dass Wissensinseln vermieden werden, damit das Ausscheiden eines Mitglieds aus dem Tech-Team nicht zu Wissensverlust im Team führt.

8.2.1 Provisionierung

Das Provisionieren der im Abschnitt 8.3 Bruchfreies Deployment beschriebenen (verteilten) Produktivumgebung und der (verteilten) Testumgebung, inklusive deren Konfiguration, geschieht unter Einsatz von Skripten. Es wird in dieser Arbeit aufgrund des begrenzten Rahmens nicht weiter ausgeführt.

8.2.2 Sprung in die Cloud

Die *Cloud* gehört klar zu den Themen von DevOps. Die Frage ist, ob *der Sprung in die Cloud* für die delegs-Editor-Anwendung erfolgen sollte. Tendenziell wäre dies jederzeit möglich. Es ist aber zu empfehlen, schrittweise vorzugehen und zuerst die Anwendung in mehrere Container zu kapseln, wie es in Abschnitt 8.3 Bruchfreies Deployment beschrieben ist. Die Container sollten, sofern möglich, zuerst auf der bestehenden Infrastruktur betrieben werden. Sollte dies wegen des Anbieters nicht möglich sein, kann auch direkt in die Cloud gewechselt werden, statt zu einem ähnlichen Virtual Server (VPS) Anbieter zu wechseln.
Es ist zu beachten, dass die Kosten für den Betrieb in der Cloud, verglichen mit dem Ist-Zustand, sehr wahrscheinlich ansteigen werden. Allerdings nicht unbedingt um die, wie im Abschnitt 5.8.1 Fallbeispiel: Kosten von Cloud vs. virtuellem Server angegebenen 40%. Denn bei VPS wird oft von den Anbieter ein *patched Kernel* vorgegeben, der den Betrieb der Docker-Engine nicht unterstützt. In diesem Fall müsste also ein dedizierter Server angemietet werden. Außerdem würde für das neue verteilte System, siehe auch Abschnitt 8.3, der Einsatz mindestens eines zusätzlichen physikalischen Servers nötig sein. So würden die Kosten gegenüber dem im Ist-Zustand angemieteten VPS ansteigen und ein Wechsel in die Cloud würde in finanzieller Hinsicht keinen großen Unterschied mehr machen.
Ein Vorteil des Betriebs in der Cloud wäre, dass einige der Probleme, die mit dem VPS Anbieter des Ist-Zustands aufgetreten sind, nicht mehr entstehen. So würden beispielsweise der

Großteil der Updates der unterliegenden Infrastruktur, *der Cloud*, transparent für Anwendung, Entwickler und Operations durchgeführt werden. Diese müssten sich also nicht mehr darum kümmern. Wahrscheinlich wären dann aber andere technische Probleme zu meistern.

Da der *Sprung in die Cloud* für die Lösungsstrategie dieser Arbeit nicht wichtig ist, wird offengelassen, ob ein solcher Schritt gemacht wird oder nicht.

8.2.3 Continuous Everything-Server (CE-Server)

Für die Realisierung von Continuous Deployment (inklusive Continuous Integration und Continuous Delivery) in einer zum größten Teil automatisierten Pipeline ist ein CE-Server zuständig, auf dem ein entsprechendes Tool läuft. Gängige Tools, die einen solchen Prozess unterstützen, sind *JetBrains TeamCity* und *Jenkins*. Da beim Autor keine Erfahrung mit solchen Tools vorhanden ist und es im begrenzten Rahmen dieser Arbeit nicht mehr zur Einrichtung eines CE-Server kommen wird, wird die Entscheidung offengelassen und auf den Zeitraum vertagt, da ein solcher eingerichtet werden soll.
In jedem Fall wird auf einem CE-Server eine Deployment-Pipeline konfiguriert, die den im Abschnitt 8.4 Die neue Deployment-Pipeline beschriebenen Prozess realisiert.

8.2.4 Backup-Repository

Für die Sicherung von Datenbank-Backups, Container-Images, alten Anwendungsversionen (*.war Dateien) und Ähnlichem ist ein dedizierter Server nötig, von nun an *Backup-Repository* genannt. Die Anforderung an ein solches Repository ist, dass es physikalisch separiert von den restlich Anwendungskomponenten angelegt ist.[66] Sollte dies aus Kostengründen nicht möglich sein, könnte es auch auf der Hardware eingerichtet werden, auf der die Slave-Replikat-Datenbank läuft.[67]
Auch für das Backup-Repository sollte überlegt werden, wie dieses vor einem Datenverlust abgesichert wird. Aus Gründen des begrenzten Rahmens dieser Arbeit bleibt dies aber nur ein Hinweis, der bei Einrichtung des Backup-Repositories bedacht werden sollte.

8.2.5 Infrastructure-as-Code

Manuelle Tätigkeiten, die während des Deployment-Prozesses (inklusive des Test Deployments) ausgeführt werden, stoßen nur noch Skripte an, die alles Weitere automatisiert erledigen. Entsprechend wird auch mit anderen Tätigkeiten außerhalb des Deployments umgegangen,

[66] Die Komponenten der Anwendung sind beschrieben im Unterabschnitt 8.3.2 Das neue verteilte System.
[67] Mehr zum Slave-Replikat ist im Unterabschnitt 8.3.2 Das neue verteilte System zu finden.

die automatisierbar sind. Skripte und aller weiterer Infrastruktur-Code, wie die Konfiguration von eingesetzten Tools, werden nach dem *Infrastructure-as-Code*-Prinzip verwaltet.[68] Hierfür ist ein eigenes *Infrastruktur-Repository* angelegt. Infrastruktur-Code ist gut dokumentiert, was ein Nachvollziehen und die manuelle Durchführen dieser ermöglicht.
Alle Skripte sind so geschrieben, dass ihre Ausführung automatisch den Namen des eingesetzten Skripts, inklusive möglicher übergebener Parameter, per Logging festhält. Dies gilt auch für zukünftig eingesetzte Skripte. Tools werden über Skripte gestartet, wodurch auch ihr Einsatz aufgezeichnet wird. Zusammen mit der Automatisierung der neuen Deployment-Pipeline wird so die Anforderung A09 - *Strikte Wiederholbarkeit* erfüllt.[69]

8.3 Bruchfreies Deployment

In diesem Kapitel wird die technische Lösungsstrategie für das verteilte System des delegs-Editors beschrieben. Ziel ist es, ein System zu entwickeln, dessen Bereitstellung bruchlos vonstatten gehen kann, sodass die Time to Market für neue Versionen verkürzt wird. Zusammen mit der neuen Deployment-Pipeline werden so weitestgehend die Anforderung A01 - *Deployment wird bruchlos durchgeführt* und die Anforderung A03 - *Ein Deployment ist jederzeit möglich* erfüllt.[70] Die Beschreibung der Pipeline ist im folgenden Abschnitt 8.4 Die neue Deployment-Pipeline zu finden.

8.3.1 Client und Nutzerverhalten

Das fortgeschrittene delegs-Projekt mit seiner gewachsenen Codebasis und vor allem der Einsatz von GWT machen es kompliziert, Anpassung an der verteilten Architektur vorzunehmen. Eine Lösung zu entwickeln und umzusetzen, in der Code für ein Update vom GWT Client nachgeladen wird, wäre aufwändig und außerdem auch unnötig. Denn schaut man sich an, wie der delegs-Editor eingesetzt wird, wird klar, dass es keinen Anwendungsfall gibt, in dem ein Client - speziell eine Session - länger als 12 Stunden am Stück genutzt wird.[71] Hinzu kommt, dass der Client nicht als Software an die Anwender verteilt wird, sondern diese sich den Client jedes Mal neu laden, wenn sie die URL des Editors aufrufen oder einen Refresh im Browser durchführen. Dadurch gibt es keine Notwendigkeit, Updates an die Clients auszuliefern. Es reicht völlig aus, den Anwender über ein neues Update zu informieren und dies wird wie folgt

[68] Dies erfüllt die Anforderungen A07 - *Infrastructure-as-Code* und A08 - *Einsatz von Skripten*.
Genaueres zu den Anforderungen im Kapitel 7 Anforderungen.
[69] Genaueres zu den Anforderungen im Kapitel 7 Anforderungen.
[70] Genaueres zu den Anforderungen im Kapitel 7 Anforderungen.
[71] Siehe hierzu Kapitel 6.3 Kontext: Anwender, Stakeholder & Konkurrenz.

realisiert, ohne große Änderungen an der Architektur der Anwendung.
Ein kleines Widget, das im Editor eingeblendet wird, informiert den Anwender darüber, dass eine neue Version zur Verfügung steht und erklärt ihm, dass er nur seine Arbeit speichern und einen im Widget bereitgestellten Button drücken muss, um den Client neu zu laden. Ab dem Zeitpunkt, von dem an eine neue Version des delegs-Editors zur Verfügung steht, wird die alte Version nur noch innerhalb einer Frist von 12 Stunden verfügbar gehalten. Auch darüber wird der Anwender in dem Widget informiert. Es gibt ihm an, wie lange er noch in der alten Version des Clients arbeiten kann. Der Anwender kann das Widget ausblenden, aber es wird alle fünfzehn Minuten erneut angezeigt, wobei der Anwender die Möglichkeit hat, Letzteres zu unterdrücken. In der letzten Stunde vor Ablauf der Frist wird das Widget allerdings zwingend alle 15 Minuten angezeigt, was nicht mehr vom Anwender unterdrückt werden kann. In der letzten Minute der Frist öffnet sich erneut das Widget und informiert den Anwender darüber, dass seine Arbeit nach Ablauf der Minute automatisch gespeichert und danach die neue Version durch einen Refresh im Browser geladen wird. Wenn ein Anwender in den Dokumenten-Manager hineinwechselt, wird ebenso automatisch ein Refresh im Browser aufgerufen. Dies gilt auch, wenn er sich im Dokumenten-Manager befindet. Allerdings gibt es im letzteren Fall Einschränkungen, vor bzw. nach welchen Anwenderinteraktionen automatisch ein Refresh durchführt werden darf.[72] Um den Anwender nicht zu verwirren, wird zuvor noch für ein paar Sekunden angezeigt, dass der Editor sich nun auf eine neue Version aktualisiert. Ebenso wird nach dem Speichern eines Dokuments verfahren.

8.3.2 Das neue verteilte System

Wenn eine neue Version zur Verfügung steht, wird vom Client das oben erwähnte Widget eingeblendet. Entsprechend muss er beim Software-Server auf eine neue Version anfragen. Dieser antwortet dann entsprechend eines Datums, dass er aus einer Konfigurationsdatei ausliest. Beschreibt das Datum einen Zeitpunkt in der Zukunft des Software-Servers, steht eine neue Produktivversion bereit. Das Datum wird immer dann automatisch gesetzt, wenn eine Testversion als tauglich empfunden und daraufhin produktiv gestellt wurde. Um im laufenden Produktivbetrieb einen Wechsel von der einen Version auf eine andere zu ermöglichen, laufen immer zwei Produktivversionen der delegs-Editor-Serverkomponente parallel. Das hier angewendete Konzept ist das bereits im Unterabschnitt 5.7.2 beschriebene Blue/Green Deployment.

[72] Vor der Umsetzung muss genauer Untersucht werden, bei welchen Aktionen ein Refresh möglich ist und bei welchen nicht. Dies kann nicht in dieser Arbeit geschehen, da es über ihren Rahmen hinaus gehen würden. Ein automatisches Laden einer neuen Version durch einen Refresh, kann aber immer nur dann passieren, wenn die Anwendung nicht Gefahr läuft, dadurch Daten zu verlieren.

Es basiert auf dem parallelen Bereitstellen von verschiedenen Softwareversionen und erfordert wenig Veränderung an der schon bestehenden Anwendung.

Das neue verteilte System, das den Blue/Green Betrieb unterstützt, besteht aus den folgenden Komponenten: *Client*, *Load-Balancer*, zwei *Software-Servern* (Blue und Green), einer *Master-Datenbank* und einer replizierten *Slave-Datenbank*. Die Komponenten laufen alle in eigenen Docker-Containern. Ausnahme machen hier der Client, der weiterhin im Browser ausgeführt wird und eventuell der Load-Balancer, der beim Sprung in die Cloud auch als Service genutzt werden kann, den viele Cloud-Anbieter standardmäßig anbieten.
In den folgenden Unterabschnitten werden die Komponenten textuell beschrieben. Es wird erörtert, wie ihr Einsatz ein bruchfreies Deployment ermöglichen. Zusätzlich ist das System in den Abbildungen 8.1 und 8.2 modelliert, um einen Überblick darüber zu verschaffen. Abbildung 8.2 zeigt, wie das Test Setup parallel zum Produktivbetrieb aufgestellt ist. Wie auch das neue verteilte System berücksichtigt das Modell nicht die per WordPress bereitgestellte Webseite von delegs, Näheres dazu im Unterabschnitt 8.3.6 Bereitstellung der delegs-Webseite.

Um die folgenden Beschreibungen der Komponenten verständlicher zu gestalten, wird dort gelegentlich von *Blue* und *Green* Versionen der Komponenten gesprochen. *Blue* und *Green* bezeichnet immer zwei unterschiedliche Versionen der gleichen Komponente und normalerweise immer zwei Versionen, die chronologisch gesehen nacheinander entwickelt wurden.

Client

Der neue Client erfährt verglichen mit dem Client im Ist-Zustand wenig Veränderung. Lediglich das bereits oben beschriebene Widget, inklusive der nötigen Logik, wie der Kommunikation mit dem Software-Server, müssen eingebaut werden.

Eine weitere Veränderung betrifft die Kommunikation zwischen dem Browser als Laufzeitumgebung für den Client und dem Software-Server und nicht den Software-Client als solches. Der erste Aufruf der Produktiv- bzw. Test-URL, der nötig ist, um den delegs-Editor zu starten, geht nicht mehr direkt an den Software-Server, sondern an den Load-Balancer. Der Load-Balancer verweist den Aufrufer dann an den entsprechenden Software-Server.[73] Danach findet die Kommunikation wie schon im Ist-Zustand zwischen Client und dem zugewiesenen Software-Server statt.

Der Vollständigkeit halber sei hier erwähnt, dass es gleichzeitig mehrere Clients geben kann, die mit einem Software-Server kommunizieren. Sowohl *Blue* Clients, die mit einem *Blue*

[73] Der Software-Client ist hiervon nicht betroffen, da für ihn die Kommunikation zwischen Browser als Laufzeitumgebung und Software-Server bis zum Laden des Software-Clients transparent ist. Der Software-Client wird beim ersten Aufruf der URL technisch gesehen noch gar nicht genutzt, da er erst durch diesen in den aufrufenden Browser geladen und dort ausgeführt wird.

Server kommunizieren, *Green* Clients, die mit einem *Green* Server kommunizieren als auch *Test* Clients, die mit einem *Test* Server kommunizieren.

Load-Balancer

Der Load-Balancer ist eine der Schlüsselkomponenten, die das parallele und somit bruchfreie Bereitstellen der Anwendung ermöglicht. Er leitet die Aufrufe von Produktiv- und Test-URLs an die entsprechenden Software-Server weiter.[74] Browser, die die Produktiv-URL aufrufen, werden immer an den Produktivserver mit der höchsten Versionsnummer weitergeleitet (hier: *Blue*). Clients, die die Test-URL aufrufen, werden an eine Testversion des Servers weitergeleitet, sofern eine solche zur Verfügung steht. Die Weiterleitung ist für die Clients transparent.
Auf eine parallel noch laufende ältere Produktivversion des Servers (hier: *Green*) wird vom Load-Balancer im fehlerfreien Betrieb nicht verwiesen. Sollte allerdings der Blue Server ausfallen, leitet der Load-Balancer neu eingehende Anfragen automatisch auf die Green Version um. Auch wenn ein Fehler in der Blue Version entdeckt wird, der den Produktivbetrieb zu sehr stört, kann der Load-Balancer dazu genutzt werden, den eingehenden Verkehr auf die Green Version umzuleiten.
Zur Sicherheit ist der Load-Balancer standardmäßig so konfiguriert, dass er immer auf eine Wartungsarbeiten-Webseite verweist, wenn für eine URL kein Server angemeldet ist.

Software-Server

Auch beim Software-Server ändert sich nicht viel im Vergleich mit dem Software-Server des Ist-Zustands. Er wird immer noch als *.war Datei auf einem Tomcat-Server bereitgestellt, allerdings in einem Docker-Container. Außerdem stellt er nun zusätzlich die oben beschriebene Funktionalität, Anfragen zu beantworten, ob eine neue Version zur Verfügung steht und er kann entsprechend von außen im laufenden Betrieb konfiguriert werden.

Der Umgang mit Client-Sessions ist zur Sicherheit so konfiguriert, dass diese nicht länger als den Zeitraum von 12 Stunden, plus einem kurzen Puffer, gültig sind. Wenige Minuten nach Ablauf der Frist wird der Container, der den Tomcat-Server hält, außer Betrieb gesetzt (beim Sprung in die Cloud können so Kosten vermieden werden).
Das Datum, das angibt, wann der Server keine Clients mehr annimmt, ist in einer Konfigurationsdatei hinterlegt.
Es ist sehr unwahrscheinlich, dass im delegs-Projekt öfter als alle 12 Stunden ein Produktiv

[74] Die Beschreibung der Produktiv- und Test-URLs ist im Abschnitt 6.2 Technische Beschreibung des Systems zu finden.

Deployment vorgenommen wird. Sollte dies doch der Fall sein, müssten die 12-Stunden-Regulierung und wahrscheinlich auch das gesamte Deployment-Konzept überdacht werden.

Vom Software-Server können gleichzeitig drei unterschiedliche Versionen aktiv sein: Zwei Produktivversionen *Blue* und *Green* und eine Testversion. Im Normalfall existieren Blue und Green immer parallel. Eine Ausnahme macht der Zeitpunkt, zu dem eine alte Version aus dem Betrieb genommen wird und durch eine neue ersetzt wird.

Master-Datenbank

Die Master-Datenbank übernimmt die gleiche Aufgabe, die die Produktivdatenbank im Ist-Zustand übernommen hat. Auf ihr werden die Produktivdaten persistiert und für die Anwendung bereitgehalten. Die Master-Datenbank läuft in einem eigenen Container und repliziert alle Schreiboperationen auf das Slave-Replikat, das seinerseits in einem eigenen Container läuft.

Eine Transition vom Ist-Zustand mit nur einer (Master-)Datenbank im Produktivbetrieb zum Soll-Zustand mit Master-Slave-Replikation ist mit minimaler Ausfallzeit des MySQL-Servers möglich. Entsprechend kurz ist dadurch der Ausfall der Produktivversion des delegs-Editors. In der initialen Phase muss eine neue Konfiguration und ein darauf folgender kurzer Neustart des MySQL-Servers vorgenommen werden. Der MySQL-Server wird für den Einsatz von *Binary Logging* konfiguriert. Binary Logging ermöglicht es, sich den Zustand einer Datenbank zu einem definierten Zeitpunkt zu '*merken*'. Zu einem späteren Zeitpunkt kann dann vom 'gemerkten' Zustand Stück für Stück eine konsistente Kopie auf eine andere Datenbank gespiegelt werden, ohne dabei eine Lesesperre zu erzeugen. So kann das Slave-Replikat synchron gehalten werden, ohne die Performance des Masters negativ zu beeinflussen.

Zwei weitere Themen betreffen noch die Datenbank. Diese werden separat in den Unterabschnitten 8.3.4 Änderungen am Datenbankschema und 8.3.5 Änderungen am Dokumentformat weiter unten behandelt.

Slave-Replikat-Datenbank

Die Slave-Datenbank agiert nur als Replikat, auf das von den Produktiv- und Testservern nicht zugegriffen wird. Es wird lediglich synchron zu der Master-Datenbank gehalten und läuft in einem eigenen Container. Die Slave-Datenbank dient als Sicherung, auf die bei Ausfall der Produktivdatenbank zurückgegriffen werden kann. Würde nur eine einzigen Datenbank eingesetzt werden, so bestünde die Gefahr, alle Produktivdaten zu verlieren, sollte diese einen Defekt erleiden oder zerstört werden.

Trotz des Einsatzes eines Slave-Replikats werden zusätzlich noch Backups der Datenbank angelegt, so zum Beispiel beim Produktiv Deployment. Dies hat den Grund, dass es Fälle gibt, in denen der aktuelle Datensatz der Slave-Datenbank sich nicht als Backup eignet. Ein Beispiel dafür ist im Abschnitt 8.3.5 Änderungen am Dokumentformat weiter unten aufgeführt.

Der Container der Slave-Datenbank muss auf einem zum Container der Master-Datenbank separaten Hardware-Server laufen bzw. nach dem Sprung in die Cloud auf einem separaten Service aufsetzen. So wird sichergestellt, dass bei einem Fehler, der zum Datenverlust auf der Master-Datenbank führt, die Daten auf dem Replikat nicht davon betroffen sind.

8.3.3 Umgang mit der Testversion

Mit der neuen Lösungsstrategie wird nicht mehr wie im Ist-Zustand der relativ umständliche Weg gewählt, zuerst eine Testversion bereitzustellen und diese nach erfolgreich bestandenen Tests wieder aus dem Testbetrieb zu nehmen, um sie dann neu zu kompilieren und danach produktiv zu stellen. Stattdessen wird die Testversion parallel zu den Blue und Green Versionen bereitgestellt. Siehe hierzu auch Abbildung 8.2. Eine bereitgestellte Testversion ist, wie schon im Ist-Zustand, über die Test-URL erreichbar. Allerdings wird nun auch hier der Load-Balancer eingesetzt, der beim Aufruf der Test-URL auf den Testserver weiterleitet. Wie auch im Produktivbetrieb ist dies für den Test Client transparent.
Die Testversion arbeitet, im Gegensatz zum Produktivbetrieb, nur auf einer Testdatenbank ohne Replikation. Wird eine neue Testversion bereitgestellt, wird die Testdatenbank mit den aktuellen Produktivdaten befüllt. Dafür wird per Binary Logging ein Dump der aktuellen Slave-Datenbank gezogen.

Nach erfolgreich absolvierten Tests ersetzt die Testversion die ältere der Blue und Green Versionen und wird die neue Produktivversion. Dies geschieht automatisch nach folgendem grob beschriebenen Schema:[75]

- die ältere der beiden Produktivversionen wird vom Netz genommen (hier: Blue)
- die Testversion wird umkonfiguriert und hat danach Zugriff auf die Produktivdatenbank
- die ehemalige Testversion wird als neue Produktivversion (Blue) bereitgestellt
- der Load-Balancer wird so konfiguriert, dass er nur noch auf die neue Version (Blue) weiterleitet

[75] Eine detailiertere Beschreibung des kompletten Prozesses ist im Abschnitt 8.4 Die neue Deployment-Pipeline zu finden.

- die Green Version wird so konfiguriert, dass sie das oben beschriebenen Widget anzeigt und von nun an auf die neue Version des delegs-Editors hinweist
- nach Verstreichen der 12-Stunden-Frist, stellt die Green Version die Bearbeitung von noch offenen Sessions ein

Zusammen mit der neuen Deployment-Pipeline, beschrieben im Abschnitt 8.4, erfüllt die Testumgebung, die in diesem Unterabschnitt vorgestellt wurde, die Anforderung A15 - *Testumgebung gleicht Produktivumgebung*.[76]

[76] Genaueres zur Anforderung im Kapitel 7 Anforderungen.

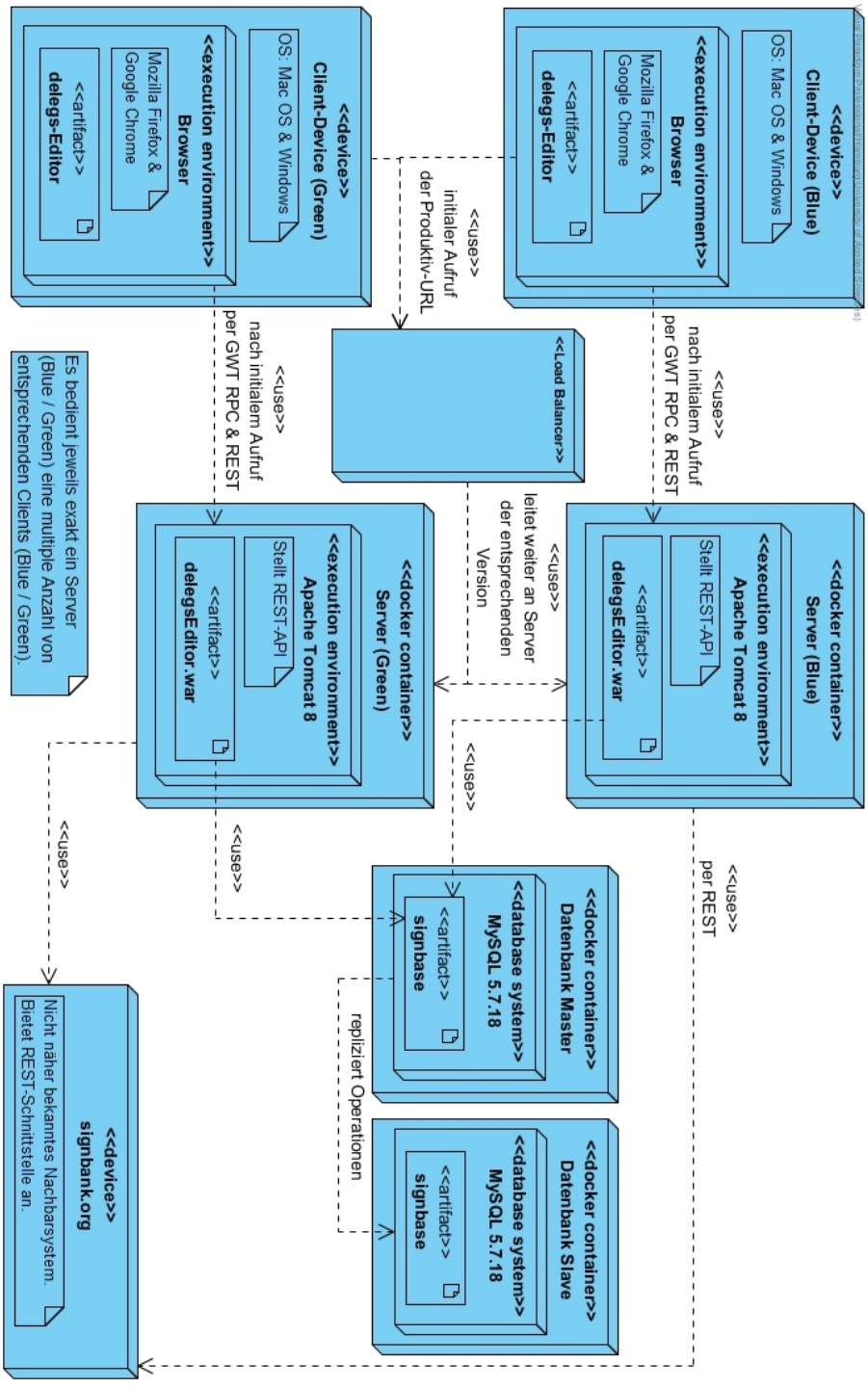

Abbildung 8.1: Soll-Zustand des verteilten Systems (ohne Test Setup)

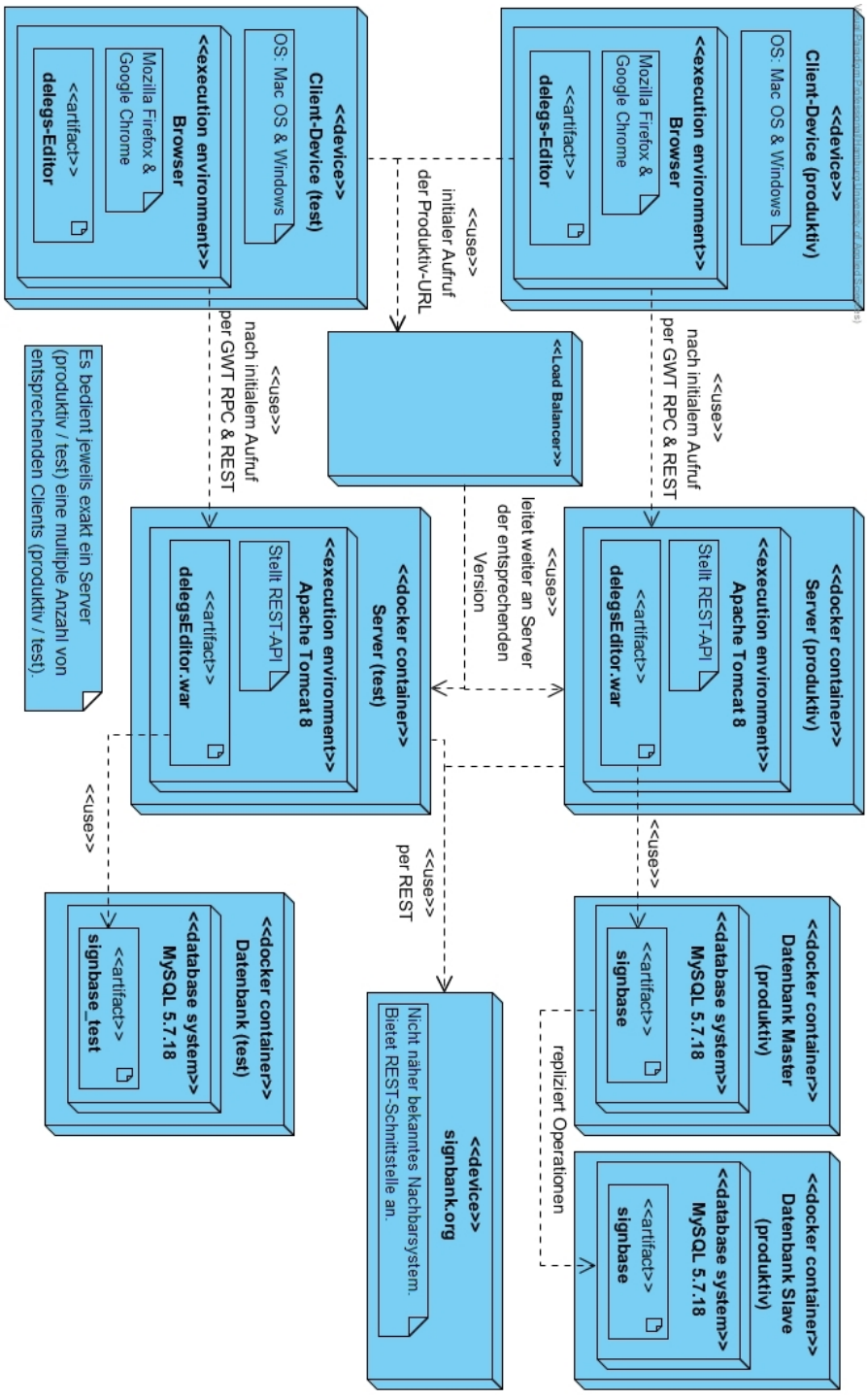

Abbildung 8.2: Soll-Zustand des **Test Setups** im verteilten System

Anmerkung zu den Abbildungen 8.1 und 8.2:
Aus Gründen der Übersichtlichkeit ist in Abbildung 8.1 das Setup für eine Testversion nicht mit aufgeführt. Dafür ist dieses in der Abbildung 8.2 dargestellt, in der die produktive Blue/Green Landschaft nur beispielhaft skizziert ist und mit dem Hinweis *'produktiv'* markiert wurde. Das Test Setup wird aber trotzdem parallel zur produktiven Blue/Green Landschaft aufgesetzt, so wie oben im Unterabschnitt 8.3.3 Umgang mit der Testversion beschrieben.

8.3.4 Änderungen am Datenbankschema

Änderungen am Datenbankschema führen mit dem neuen, in diesem Abschnitt (8.3) vorgestellten Konzept immer noch zum Ausfall des Produktivbetriebs. Ein System zu entwerfen, umzusetzen und zu warten, das bei keiner Art von Deployment ausfällt, selbst nicht bei der Änderung am Datenbankschema, würde eine sehr viel höhere Komplexität in die Anwendung bringen. Eine solches System ist nicht gerechtfertigt für den Anwendungskontext des delegs-Editors. So gibt es dafür beispielsweise eine zu geringe Anzahl von Anwendern. Außerdem wird die Anwendung nicht in einem Bereich eingesetzt, in dem ein Ausfall hohe Kosten verursacht oder ein hohes Sicherheitsrisiko bedeutet. Hinzu kommt, dass Änderungen am Schema der Datenbank eher selten vorkommen. Es wurde daher darauf verzichtet, ein entsprechendes System zu entwerfen.

Mit der Lösungsstrategie dieser Arbeit wird eine Anhebung der Version der Datenbank nur noch durchgeführt, wenn sich ihr Schema ändert. Dadurch wird nicht mehr, wie im Ist-Zustand, mit jeder neuen Version der Anwendung auch automatisch das Durchführen eines Migrationsskriptes auf der Datenbank nötig. Im Fall eines Deployments, das eine Datenbankmigration mit sich bringt, wird die Anwendung für den Zeitraum der Migration außer Betrieb genommen. Der Load-Balancer leitet dann die Produktiv-URL auf seine standardmäßig gesetzte Wartungsarbeiten-Webseite um. Bevor das Deployment angestoßen wird, werden die aktiven Anwender mit einem Widget über die Wartungsarbeiten informiert. Dieses verhält sich ähnlich zu dem im Unterabschnitt 8.3.1 Client und Nutzerverhalten beschriebenen Widget. Es warnt die Anwender rechtzeitig und ermahnt sie, ihre Arbeit zu speichern.

Eine vielversprechende Herangehensweise an die Problematik der Datenbankmigration scheint der Einsatz von Datenbank-Refactoring zu sein. Die Vertiefung dieses Themas geht leider über den Rahmen dieser Arbeit hinaus. Für einen Einstieg in das Datenbank-Refactoring, kann an dieser Stelle aber auf die Webseite von *Pramod Sadalage* verwiesen werden (siehe Sadalage (2016)), die unter anderem von *Martin Fowler* empfohlen wurde.

8.3.5 Änderungen am Dokumentformat

Eine bereitgestellte neue Version des delegs-Editors (*Blue*) kann eine Änderung am Format der im Editor erstellbaren Dokumente mit sich bringen. Es müssen dabei zwei Dinge beachtet werden, um die Kompatibilität der aktuellen Blue Version und der älteren Green Version mit allen Dokumenten, unabhängig der von ihnen unterstützten Formatversion, zu gewährleisten:

1. Es ist eine Konvertierung aller Dokumente erforderlich, die mit älteren Versionen als der neuen Blue Version erstellt wurden. Im Ist-Zustand ist es bereits so, dass alte Dokumente immer erst dann konvertiert werden, wenn diese durch eine neuere Editor-Version geöffnet werden. So soll vorerst auch im neuen System verfahren werden.
   Auf lange Sicht sollte jedoch darüber nachgedacht werden, eine andere Konvertierungslogik einzusetzen. Der Code des momentan eingesetzten Konvertierungsalgorithmus bläht sich schon jetzt immer weiter auf und ist unübersichtlich. Es ist allerdings eine komplexe Aufgabe, eine neue Konvertierungslogik zu entwickeln, die nicht im Rahmen dieser Arbeit gelöst werden kann. Eine Lösung könnte wahrscheinlich gut auf die im Unterabschnitt 9.6 grob skizzierte Erweiterung des Systems, der Auswertung der Lösungsstrategie aufgesetzt werden. Diesem stehen zwei parallele '*Master*' Produktivdatenbanken zur Verfügung, die den Umgang mit der Konvertierung einfacher gestalten sollten.

2. Während der Übergangsphase von Green zu Blue können Clients beider Versionen aktiv sein. Es kann passieren, dass ein Blue Client ein Dokument im neuen Format erstellt und speichert und ein Green Client daraufhin dieses öffnen möchte. In diesem Fall muss dem Green Client, wieder unter Einsatz des oben beschriebenen Widgets, Bescheid gegeben werden, dass er, um das Dokument öffnen zu können, einen Refresh im Browser durchführen muss. Mit seinem neu geladenen Blue Client, kann er dann das Dokument öffnen.

Folgendes ist nach einem Fehlerfall und einem darauffolgenden Rollback auf die alte Green Version zu beachten: Die Green Version unterstützt das neue Dokumentenformat noch nicht. Das kann zu Problemen führen, da auf der Datenbank vorhandene Dokumente eventuell bereits in das neue Format konvertiert wurden. Im Falle eines Rollbacks muss daher auch die Datenbank auf einen Stand vor dem Bereitstellen der neue Blue Version zurückgesetzt werden, weswegen bei einem Produktiv Deployment immer eine Sicherung der Produktivdatenbank angelegt wird.

8.3.6 Bereitstellung der delegs-Webseite

Die Webseite von delegs und ihre Bereitstellung sind nicht Teil dieser Arbeit. Da die Webseite im Ist-Zustand Teil des Systems ist, auf dem auch der delegs-Editor bereitgestellt wird, muss an dieser Stelle kurz auf sie eingegangen werden.

Für die Webseite ergeben sich durch die oben beschriebene Umstrukturierung des delegs-Editors keine Auswirkungen. Sie und der Editor standen niemals in Abhängigkeit und daher kann sie weiterhin genauso betrieben werden, wie es schon im Abschnitt 6.2 Technische Beschreibung des Systems beschrieben wurde.

Anmerkung des Autors: Eine tiefergehende Beschreibung des neuen verteilten Systems ist im begrenzten Rahmen dieser Arbeit nicht möglich.

8.4 Die neue Deployment-Pipeline

In diesem Abschnitt wird die neue Deployment-Pipeline vorgestellt und entsprechend mit ihr auch der neue Deployment-Prozess. Die Entwicklung der Pipeline ist wichtig, um eine kürzere Time to Market für neue Versionen des delegs-Editors zu erzielen. Die im Abschnitt 8.3 Bruchfreies Deployment vorgenommenen Änderungen am verteilten System wurden auch im Sinne dieses Ziels entworfen. System und Pipeline zusammen erfüllen weitestgehend die Anforderung A01 - *Deployment wird bruchlos durchgeführt* und die Anforderung A03 - *Ein Deployment ist jederzeit möglich.*

In den folgenden Unterabschnitten wird zuerst auf nötige Änderungen im Umgang mit manuellen und automatisierten Tests eingegangen und die daraus resultierenden Test-Suites werden beschrieben. Danach wird auf die zwei Teilprozesse der Deployment-Pipeline eingegangen: *Test Deployment* und *Produktiv Deployment.* Abschließend wird noch auf Monitoring und auf mögliche Fehlerfälle Bezug genommen und es werden allgemeine Anmerkungen zum Deployment-Prozess gemacht.

8.4.1 Das Durchführen von Tests

Im Deployment-Prozess des Ist-Zustands werden an verschiedenen Stellen sowohl manuelle als auch automatisierte Tests an der bereitzustellenden Anwendung durchgeführt.[77] Die Aufstellung der Tests im Ist-Zustand ist nicht effizient genug gewählt, was zu unnötigem Zeitaufwand

[77] Die Beschreibung des Deployment-Prozesses im Ist-Zustand ist im Abschnitt 6.7 Deployment-Prozess (Ist-Zustand) zu finden.

führt. Ein weitere Quelle, die das Deployment unnötig hinauszögert, ist die *Testwoche*, die zwischen Test- und Produktiv-Deployment stattfindet. Der ihr zugestandene Zeitraum ist zu lang. In diesem Unterabschnitt ist beschrieben, wie die Aufstellung und das Durchführen der Tests für die neue Deployment-Pipeline optimiert werden. Hierzu werden die Tests in verschiedene Test-Suites unterteilt, auf die in der Beschreibung der Deployment-Pipeline Bezug genommen wird. Diese sind *Pre-Commit-Tests*, *Post-Commit-Tests*, *Test Deployment-Tests*, *Akzeptanz-Tests*, *Performance-Tests* und *Produktiv Deployment-Tests*. Sie werden im Folgenden kurz beschrieben.

Aufgrund des begrenzten Rahmens dieser Arbeit kann auf die einzelnen Tests, die die Suites beinhalten, nicht konkret eingegangen werden.

- **Pre-Commit-Tests**
  Diese Test-Suite soll zum Testen der Anwendung während der Entwicklungsphase eingesetzt werden. Sie soll ausgeführt werden, um grob testen zu können, ob die an der Codebasis vorgenommenen Änderungen die Funktionalität der Anwendung negativ beeinflussen und zu Fehlern führen. Da die Suite während des Entwicklungsprozesses ausgeführt werden soll, muss sie so zusammengestellt sein, dass wenig Wartezeit für die Entwickler entsteht. Trotz kurzer Wartezeit soll sie aber eine Testabdeckung erzielen, die ausreicht, um ein Feedback an die Entwickler darüber zu liefern, ob grobe Fehler eingebaut wurden. Ziel ist es, dass ein positiver Durchlauf dieser Suite mit hoher Wahrscheinlichkeit auch zu einem positiven Durchlauf der *Post-Commit-Tests* führt.

  Die Test-Suite besteht aus einer Reihe von Unit- und Smoke-Tests und überprüft außerdem das Einhalten von Code-Konventionen. Genutzte Services werden nur als Mocks angebunden. Die Unit-Tests sollten auch immer um die Tests erweitert werden, die den neu entwickelten Code testen.

- **Post-Commit-Tests**
  Diese Test-Suite wird jedes Mal nach einem Commit automatisch vom Continuous Everything-Server ausgeführt. Da der Server über mehr Rechenleistung verfügt, können weitaus mehr Tests durchgeführt werden, als in der *Pre-Commit-Test-Suite*. Da die Entwickler aber immer noch abhängig vom Ergebnis der Tests der Suite sind, darf der Zeitaufwand, sie durchzuführen, trotzdem nicht allzu groß sein. Zugleich gibt es an die Suite die Anforderung, dass ein positiver Durchlauf mit hoher Wahrscheinlichkeit zu einem positiven Durchlauf der *Test Deployment-Tests* führen.

  Die Test-Suite besteht aus einer Reihe von Unit- und Integrations-Tests und überprüft außerdem das Einhalten von Code-Konventionen. Genutzte Services werden in Form von Mocks simuliert.

- **Test Deployment-Tests**
  Diese Test-Suite wird im Zuge des Test Deployments durchgeführt. Sie hat das Ziel, die Korrektheit bzw. die fehlerfreie Funktion der Anwendung zu prüfen. An ihren Einsatz ist keine Anforderung gestellt, in kurzer Zeit durchzulaufen. Ihr Fokus liegt darauf, möglichst *viele* Fehler aufzudecken. Daher sind in ihr alle automatisierten Tests enthalten. Die Tests arbeiten auf einer Testdatenbank, die aus einem Testdatenbank-Container-Image erzeugt werden und wenn nötig mit Kopien der Daten der Produktivdatenbank befüllt werden. Da externe Services nur per Lesezugriff genutzt werden, wird in dieser Suite auf den echten Services und nicht auf Mocks gearbeitet.

- **Akzeptanz-Tests und Performance-Tests**
  Diese zwei Test-Suites beinhaltet nur die manuell durchzuführenden Akzeptanztests bzw. die Performance-Tests. Teilmengen von ihnen werden durch die Entwickler durchgeführt und Teilmengen von ihnen durch den speziellen Power User und die restlichen Stakeholder.

- **Produktiv Deployment-Tests**
  Diese Test-Suite beinhaltet Tests, die wieder nur grob die Funktionalität der Anwendung testen. Sie wird ausgeführt, nachdem die zuvor bereitgestellte Testversion als tauglich für den Produktivbetrieb empfunden und in diesen überführt wurde. Zu besagten Zeitpunkt ist die Anwendung schon grundlegend getestet worden. Ziel der enthaltenen Tests ist es daher, nur zu bestätigen, dass der Wechsel in den Produktivbetrieb ohne Fehler vonstatten gegangen ist und die Anwendung wie erwartet antwortet.

Jeder automatisierte Test, der während des Test Deployment-Prozesses durchgeführt wird und auf einer Datenbank arbeitet, bekommt seine eigene Test-Datenbank-Instanz. Diese wird aus einem Testdatenbank-Container-Image erzeugt, welches im Projekt-Repository liegt. Die eingesetzten Instanzen werden nach dem Testen wieder gelöscht. So müssen Änderungen an den Daten, die durch die Tests entstehen, nicht zurückgedreht werden. Dies ermöglicht es, Tests, die auf einer Datenbank arbeiten, unabhängig voneinander und darum parallel auszuführen. So wird mit allen Tests verfahren, die vor der Bereitstellung der Testversion durchgeführt werden. Die Datenbank, der sich der delegs-Editor bedient, wenn er lokal auf einem Entwicklerarbeitsplatz gestartet wird, wird auch aus dem Testdatenbank-Container-Image erzeugt.

Die manuellen Tests, die während der Testwoche durchgeführt werden, sind vom Aufwand her innerhalb eines Tages zu bewältigen. Da die Tests aber von verschiedenen Personen durchgeführt werden, wird ein zusätzlicher Tag Spielraum hinzugerechnet. Damit ist die

Testwoche nur noch zwei Tage lang.[78] Es sollte im Projektalltag überprüft werden, ob es möglich ist, den Tests soviel Priorität zuzusprechen, dass die Testwoche noch weiter gekürzt werden kann und innerhalb eines Tages durchgeführt wird.

Mit dem unten beschriebenen Einsatz in der neuen Deployment-Pipeline erfüllen die Test-Suites die Anforderung A18 - *Geringer Zeitaufwand für automatisierte Tests.*[79]

8.4.2 Beschreibung der Test Deployment-Pipeline

Die Test Deployment-Pipeline ist das erste der zwei Teilstücke der Deployment-Pipeline. Manuell angestoßen hebt sie eine neue Version der delegs-Anwendung in eine Staging-Umgebung, in der die Anwendung manuell getestet werden kann. Während des Durchlaufens dieses Teilstücks wird die Anwendung automatisierten Tests unterzogen.

In der folgenden Abbildung 8.3 ist die Test Deployment-Pipeline dargestellt. Anschließend wird textuell auf die einzelnen Schritte der Pipeline eingegangen.

[78] Die *Testwoche* verdient eigentlich einen anderen Namen. Da es aber einfacher ist, sich darauf zu beziehen, wird der Name in dieser Arbeit nicht geändert.

[79] Genaueres zur Anforderung im Kapitel 7 Anforderungen.

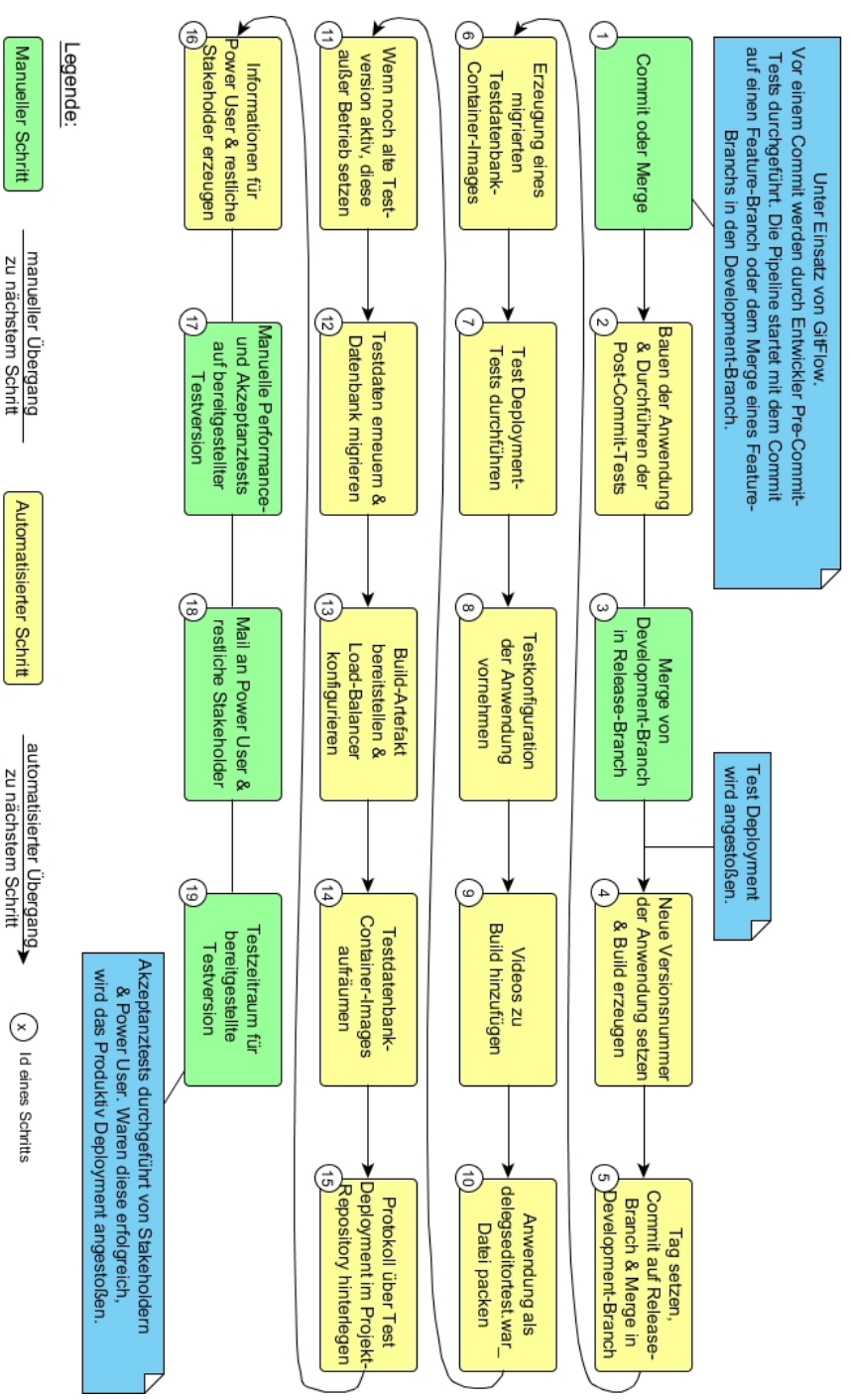

Abbildung 8.3: Pipeline für Test Deployment (SOLL-Zustand)

Folgend werden die einzelnen Schritte der Test Deployment-Pipeline aufgeführt und wenn nötig, erörtert. Die Pipeline ist in Abbildung 8.3 dargestellt.
(Manuelle Schritte sind als solche gekennzeichnet. Die anderen nicht markierten Schritte sind solche, die automatisch durchgeführt werden. Die Nummern vor den einzelnen Punkten im Text isindst analog vergeben zur Nummerierung in der Abbildung.)

1. **Commit oder Merge**
   (manuell durchgeführt)
   Die Pipeline startet mit dem Commit auf einen Feature-Branch oder dem Merge eines Feature-Branchs in den Development-Branch. Vor einem Commit werden durch die Entwickler die *Pre-Commit-Tests* durchgeführt.

2. **Bauen der Anwendung & durchführen der Post-Commit-Tests**
   Bei einem **Commit** auf einen Feature-Branch wird automatisch ein Merging des Development-Branchs in den Feature-Branch vorgenommen. Danach wird ein Build angestoßen und die Anwendung den *Post-Commit-Tests* unterzogen. Laufen der Build oder die darauf folgenden Tests in einen Fehler, wird der Commit verworfen. Der Entwickler, der den Commit eingebracht hat, wird über den Fehler informiert.
   Bei Merge eines **Feature-Branchs** in den Development-Branch entsteht ein neuer Merge-Commit. Auch hierauf wird ein Build angestoßen und die Anwendung den *Post-Commit-Tests* unterzogen. Verlaufen Build oder Tests fehlerhaft, wird der Merge-Commit verworfen und kommt nicht auf den Development-Branch. Stattdessen wird er auf einen temporären Branch ausgelagert, auf dem die Entwickler sich den Fehler anschauen und wenn gewollt, dort beheben können. Der Entwickler, der den Merge angestoßen hat, wird über den Fehler und den temporären Branch informiert.

3. **Merge von Development-Branch in Release-Branch**
   (manuell durchgeführt)
   Ein menschlicher Gatekeeper führt den Merge durch und stößt damit das automatische Deployment einer neuen Testversion an.

4. **Neue Versionsnummer der Anwendung setzen & Build erzeugen**

5. **Tag setzen, Commit auf Release-Branch & Merge in Development-Branch**
   Es wird ein Commit erzeugt. Danach wird ein Tag mit der Versionsnummer der neuen Anwendungsversion als Bezeichner angelegt. Der Tag markiert den neuen Commit. Durch ein Merging des Release-Branchs in den Development-Branch wird der Commit auch in den Development-Branch übertragen.

6. **Erzeugung eines migrierten Testdatenbank-Container-Images**
   Es wird überprüft, ob ein Datenbank-Migrationsskript im Projekt-Repository hinterlegt wurde. Ist dies der Fall, ist mit der neuen Softwareversion eine Migration der Datenbank nötig. Um die Tests auf einer migrierten Datenbankversion ausführen zu können, wird eine Instanz des Testdatenbank-Container-Images erzeugt und darauf das Migrationsskript ausgeführt. Aus der Instanz wird wieder ein Image gebacken. Von dem neuen Image werden bis zum Bereitstellen der neuen Testversion alle automatisierten Tests, die eine Testdatenbank benötigen, mit Instanzen versorgt.

7. **Test Deployment-Tests durchführen**
   Die dafür vorgesehene *Test Deployment-Test*-Suite wird durchgeführt.

8. **Testkonfiguration der Anwendung vornehmen**

9. **Videos zu Build hinzufügen**
   Das Build-Artefakt ist nun vollständig.

10. **Anwendung als delegseditortest.war_ Datei packen**

11. **Wenn noch alte Testversion aktiv, diese außer Betrieb setzen**
    Wurde nach dem Bereitstellen der vorherigen Testversion in dieser ein Fehler gefunden, ist es möglich, dass sie noch in Betrieb ist, da sie nicht durch ein automatisches Produktiv Deployment in den Produktivbetrieb übergegangen ist. Näheres hierzu im folgenden Unterabschnitt 8.4.3 Beschreibung der Produktiv Deployment-Pipeline.

12. **Testdaten erneuern & Datenbank migrieren**
    Der Inhalt der Testdatenbank des bereitgestellten Testdatenbank-Containers wird gelöscht und danach mit einem Dump des Produktivdatenbank-Slaves befüllt. Ist ein Migrationsskript vorhanden, wird dieses auf der Testdatenbank ausgeführt. Die Testdatenbank ist damit bereit für den Betrieb mit der neuen Testversion.

13. **Build-Artefakt bereitstellen & Load-Balancer konfigurieren**
    Der Tomcat-Container der Testanwendung wird mit der neuen delegseditortest.war Datei beladen und bereitgestellt. Danach wird der Load-Balancer so konfiguriert, dass er Aufrufe der Test-URL an den Container weiterleitet.
    Nach Durchführung dieses Schrittes steht die Testversion online bereit, um darauf manuell zu testen.

14. **Testdatenbank-Container-Images aufräumen**
    Sollte eine Migration auf einer Kopie des Testdatenbank-Container-Images durchgeführt

worden sein, werden diese und der Original-Container, aus dem sie erzeugt wurde, im Projekt-Repository gesichert. Das noch im Projekt-Repository vorhandene ältere Image wird im Backup-Repository abgelegt.

15. **Protokoll über Test Deployment im Projekt-Repository hinterlegen**

16. **Informationen für Power User & restliche Stakeholder erzeugen**
    Es werden automatisch Informationen für speziellen Power User und die restlichen Stakeholder in einer Textdatei zusammengestellt. Diese sollen vor der Verwendung noch von den Entwicklern überarbeitet werden. Sie werden aus den Ids und den Benennungen der einzelnen Bugfixes und Features erzeugt, die mit der neuen Testversion bereitgestellt wurden.

17. **Manuelle Performance- und Akzeptanztests auf bereitgestellter Testversion**
    (manuell durchgeführt)
    Das Tech-Team testet die Performance und ausgiebig die Funktionalität der Anwendung.

18. **Mail an Power User & restliche Stakeholder**
    (manuell durchgeführt)
    Waren die manuellen Tests erfolgreich, wird die Testversion freigegeben, sodass sie durch den speziellen Power User und die restlichen Stakeholder getestet werden kann. Diesen wird eine Mail durch einen Entwickler zugeschickt. Sie beinhaltet die in Schritt 16. automatisch erzeugten Informationen, die zuvor manuell überarbeitet wurden.

19. **Testzeitraum für bereitgestellte Testversion**
    (manuell durchgeführt)
    Testwoche: Akzeptanztests der Anwendung durch speziellen Power User und restliche Stakeholder. Waren diese erfolgreich, wird manuell ein Produktiv Deployment angestoßen. Näheres zum Produktiv Deployment im folgenden Abschnitt 8.4.3 Beschreibung der Produktiv Deployment-Pipeline.

Die automatisierten Schritte geben sowohl in kurzer Form im Positivfall und ausführlicher im Negativfall Rückmeldung über ihren Verlauf.

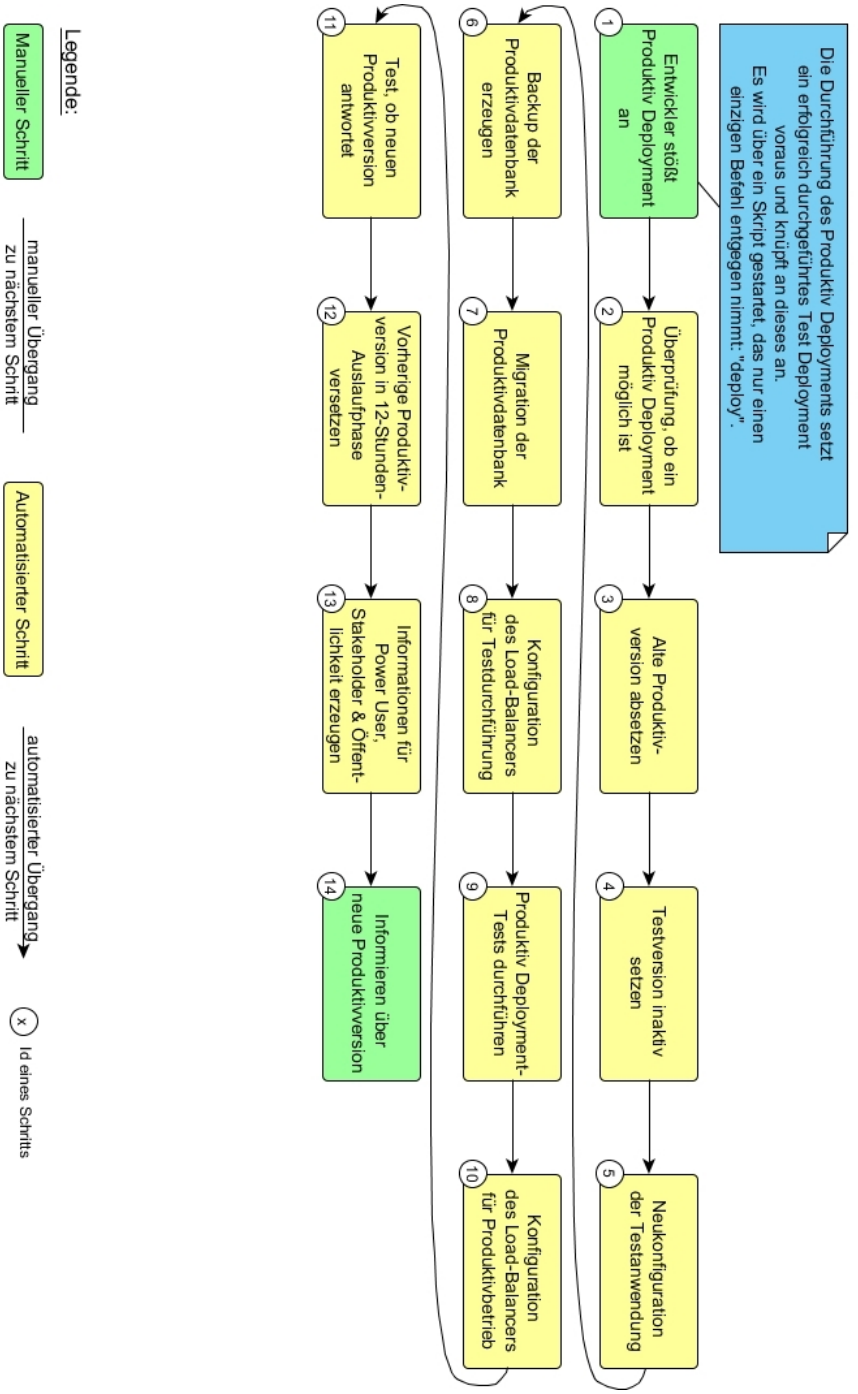

Abbildung 8.4: Pipeline für Produktiv Deployment (SOLL-Zustand)

8.4.3 Beschreibung der Produktiv Deployment-Pipeline

Nachdem eine Version in der Staging-Umgebung getestet wurde, kann sie in den Produktivbetrieb gehoben werden. Sie wird hierzu beim Durchlaufen der Produktiv Deployment-Pipeline die ältere der beiden Blue und Green Versionen ersetzen und als die aktuelle Produktivversion eingesetzt. Auch die Produktiv Deployment-Pipeline wird als zweiter Teil der Gesamt-Pipeline manuell angestoßen.
Das Produktiv Deployment soll nur durchgeführt werden, wenn zuvor eine bereitgestellte Testversion erfolgreich alle manuellen Tests bestanden hat.

Zwischen Test und Produktiv Deployment ist kein *partieller* Produktivbetrieb vorgesehen, da der delegs-Editor keine sicherheitskritische Anwendung ist und ein Ausfall keine Kosten verursacht. Außerdem würde ein partieller Produktivbetrieb aufgrund des Kontextes der Anwendung nicht genug Informationen einbringen. Der Editor wird schlicht von zu wenig Anwendern genutzt.

Folgend werden die einzelnen Schritte der Pipeline, wie sie in Abbildung 8.4 dargestellt ist, erörtert. Im Text wird zur Vereinfachung von dem Fall ausgegangen, dass die Green Version durch die Testversion ersetzt wird.
(Manuelle Schritte sind als solche gekennzeichnet. Die anderen nicht markierten Schritte sind solche, die automatisch durchgeführt werden. Die Nummern vor den einzelnen Punkten im Text sind analog vergeben zur Nummerierung in der Abbildung.)

1. **Entwickler stößt Produktiv Deployment an**
   (manuell durchgeführt)
   Das Produktiv Deployment wird durch einen menschlichen Gatekeeper gestartet. Er führt dazu ein Skript aus, das nur einen einzigen Befehl entgegen nimmt: "*deploy*".

2. **Überprüfung, ob ein Produktiv Deployment möglich ist**
   Es wird überprüft, ob auf der älteren Produktivversion (Green) die 12-Stunden-Auslaufphase bereits komplett durchlaufen wurde. Ist dies *nicht* der Fall, bricht die Deployment-Pipeline an dieser Stelle ab und gibt einen entsprechenden Hinweis aus.

3. **Alte Produktivversion absetzen**
   Aus dem stillgelegten Tomcat-Server-Container wird die delegseditor.war Datei der Green Version entfernt und im Backup-Repository abgelegt.

4. **Testversion inaktiv setzen**
   Die aktuell bereitgestellte Testversion wird inaktiv gesetzt. Dazu wird der Load-Balancer

so konfiguriert, dass dieser Aufrufe der Test-URL nicht mehr an die Testversion weiterleitet. Noch laufende Session werden invalidiert.

5. **Neukonfiguration der Testanwendung**
   Die Testversion wird für den Produktivbetrieb konfiguriert und dadurch mit der Produktivdatenbank verbunden. Das *delegseditortest.war* Artefakt wird zu *delegseditor.war* umbenannt. Die Testversion wird von nun an als die *neue Produktivversion* bezeichnet.

6. **Backup der Produktivdatenbank erzeugen**
   Von der Slave-Datenbank wird ein Backup erzeugt und im Backup-Repository abgelegt.

7. **Migration der Produktivdatenbank**
   Dieser Schritt wird nur durchgeführt, wenn eine Migration der Produktivdatenbank nötig ist.
   Die aktuell produktiv laufende Anwendung (Blue) muss aus dem Betrieb genommen werden. Hierzu wird der Load-Balancer so konfiguriert, dass er Aufrufe der Produktiv-URL auf eine Wartungsarbeiten-Webseite umleitet, die den Aufrufer darüber informiert, dass die Anwendung kurzzeitig nicht erreichbar ist.
   Die Anwender sollten eventuell im Voraus über die Wartungsarbeiten informiert worden sein. Dies könnte beispielsweise unter Einsatz eines Widgets geschehen, das sich sehr ähnlich zu dem im Abschnitt 8.3 Bruchfreies Deployment beschriebenen Widget verhält.
   Nachdem auf die Wartungsarbeiten-Website umgeleitet wurde, wird die Migration der Produktivdatenbank durchgeführt.

8. **Konfiguration des Load-Balancers für Testdurchführung**
   Von diesem Zeitpunkt an leitet der Load-Balancer Aufrufe der *Test-URL* auf die neue Produktivversion um.

9. **Produktiv Deployment-Tests durchführen**
   Unter Anwendung der *Produktiv Deployment-Test-Suite* wird grob die Funktionalität der Anwendung sichergestellt.

10. **Konfiguration des Load-Balancers für Produktivbetrieb**
    Von diesem Zeitpunkt an leitet der Load-Balancer Aufrufe der Produktiv-URL und nicht mehr Aufrufe der Test-URL auf die neue Produktivversion um. Die neue Version läuft nun produktiv.

11. **Test, ob neuen Produktivversion antwortet**
    Nur minimaler Testaufwand.

Im Fehlerfall: Antwortet die Anwendung nicht, leitet der Load-Balancer wieder auf die vorherige Produktivversion (Blue) um. Wurde eine Datenbankmigration durchgeführt, muss zuvor die migrierte Datenbank durch das in Schritt 6. erstellte Backup ersetzt werden.

12. **Vorherige Produktivversion in 12-Stunden-Auslaufphase versetzen**
    Die vorherige Produktivversion Blue wird in die 12-Stunden-Auslaufphase versetzt. Details hierzu im Kapitel 8.3 Bruchfreies Deployment.

13. **Informationen für Power User, Stakeholder & Öffentlichkeit erzeugen**
    Es werden automatisch Informationen in einer Textdatei zusammengestellt, damit diese von den Entwicklern überarbeitet und eingesetzt werden können. Die Informationen werden aus den Ids und den Benennungen der einzelnen Bugfixes und Features erzeugt, die mit der neuen Produktivversion bereitgestellt wurden.

14. **Informieren über neue Produktivversion**
    (manuell durchgeführt)
    Der spezielle Power User und die restlichen Stakeholder werden über die neue Produktivversion und die dabei vorgenommenen Änderungen und Erweiterungen per E-Mail informiert. Zusätzlich wird auf die neue Version und die vorgenommenen Änderungen und Erweiterungen auf der Homepage und auf der Facebook-Seite von delegs hingewiesen. Hierzu werden die im vorherigen Schritt automatisch generierten Informationen genutzt und zuvor von den Entwicklern überarbeitet.[80]

8.4.4 Allgemeines zum neuen Deployment-Prozess

Nun noch ein paar allgemeine Dinge zum neuen Deployment-Prozess.

Das Deployment wird auf einem dafür dedizierten Server durchgeführt und nicht mehr wie im Ist-Zustand auf den Entwicklerrechnern. Darum müssen auch vor und nach einem Build keine Änderung mehr an der Konfiguration der Anwendung, das Debugging betreffend, vorgenommen werden.

Führt die Entwicklung auf einem Feature-Branch zu einer Migration und damit zu einer Änderung an der Datenbank, wird diese beim Merging des Feature-Branchs in den Development-Branch für die gesamte Entwicklung wirksam. Haben Entwickler lokale Kopien der Datenbank auf ihren Entwicklerrechnern, müssen diese ebenfalls migriert werden. Dies geschieht unter

[80] Mit den Schritten 13. und 14. wird die Anforderung A17 - *Mail-Generierung nach Produktiv Deployment* erfüllt. Genaueres zu den Anforderungen im Kapitel 7 Anforderungen.

Nutzung des entsprechenden Migrationsskripts, das bei der Entwicklung des verantwortlichen Features geschrieben und eingesetzt wurde.
Nicht betroffen von den Veränderungen an der Datenbank ist die Entwicklung auf Feature-Branchs, solange in die Branchs kein Merging des Development-Branchs vorgenommen wird.

Mit dem in dieser Arbeit neu aufgestellten Deployment-Prozess werden die *Continuous Everything* Anforderungen A10 bis A12 nicht vollständig, aber größtenteils erfüllt. Zusammen fordern diese einen durchgängigen Deployment-Prozess ohne Unterbrechungen, was der in diesem Abschnitt vorgestellte Prozess nicht hundertprozentig erfüllt. Im Kapitel 7 Anforderungen wird genauer auf diesen Umstand eingegangen.

Der (Teil-)Automatisierung der Pipeline steht der Einwand des technischen Projektleiters Herrn Koch gegenüber. Dieser ist im Unterabschnitt 6.9.5, dem Punkt *Einwand des technischen Projektleiters zur Automatisierung* näher beschrieben. Im neuen Deployment-Prozess werden zwar fast alle Arbeiten automatisch ausgeführt. Das neue verteilte System und die neue Deployment-Pipeline und ihr zugehöriger Prozess werden aber in dieser Arbeit in den Abschnitten 8.3 Bruchfreies Deployment und 8.4 Die neue Deployment-Pipeline beschrieben. Die Beschreibungen werden in das Projekt-Repository eingefügt, sodass alle Teammitglieder darauf Zugriff haben. So ist es möglich, einem Mitarbeiter die Aufgabe zu geben, den Deployment-Prozess anhand der Beschreibung manuell auszuführen. Dieser kann dann dadurch erfahren und lernen, welche Arbeiten nötig sind, um das komplette Deployment durchzuführen.
Das Ablegen der Dokumente im Repository erfüllt auch die Anforderung A02 - *Beschriebene, einsehbare Deployment-Pipeline.*[81]

8.4.5 Im Fehlerfall

Vollständige Lösungen, wie bei einzelnen Schritten mit Fehlern umgegangen wird, würden den Rahmen dieser Arbeit sprengen. In Einzelfällen werden aber dennoch Lösungen anfänglich beschrieben.

Vor der Umsetzung der Pipeline müssen alle Fehlerfälle, die während ihres Betriebs auftreten können, bedacht werden. Ein auftretender Fehler sollte dabei so abgehandelt werden können, dass in dem Schritt, in dem der Fehler aufgetreten ist, das Deployment weiter fortgesetzt werden kann. Notfalls kann auch wenige Schritte weiter vorn in der Pipeline wieder eingestiegen werden.

[81] Genaueres zur Anforderung im Kapitel 7 Anforderungen.

8.4.6 Monitoring & Rückkopplung durch die Pipeline

Die Pipeline gibt dem Tech-Team für Menschen lesbare Rückmeldungen darüber, welche Schritte bereits durchgeführt wurden und wenn möglich auch über den aktuellen Verlauf eines Schrittes. Dies geschieht unabhängig vom Verlauf, ob nun erfolgreich oder fehlerhaft. Zusätzlich schreiben beide Deployment-Subprozesse den Fortschritt und alle anfallenden Events jeweils in eine separate Log-Datei.

Eine genauere Betrachtung des Monitorings sprengt leider wieder den für diese Arbeit vorgesehenen Rahmen. An dieser Stelle kann nur darauf hingewiesen werden, dass die Rückmeldungen aus der Pipeline ein wichtiger Mechanismus für deren Einsatz sind, um mit ihr arbeiten zu können. Die entsprechende Anforderung dazu ist A04 - *Rückkopplung durch Deployment-Pipeline*.[82]

[82] Genaueres zur Anforderung im Kapitel 7 Anforderungen.

Teil IV

Auswertung, Ausblick, Fazit

9 Auswertung der Lösungsstrategie

Die in dieser Arbeit vorgestellte Lösungsstrategie beschreibt viele Veränderungen in verschiedensten Bereichen des delegs-Projekts. Eine Umsetzung der Strategie bietet diverse Vorteile gegenüber dem Ist-Zustand. So wird, mit Ausnahme des Einbringens eines neuen Datenbankschemas, ein *bruchfreies* Deployment ermöglicht. Weitere Vorteile sind eine kürzere Time to Market, eine höhere Verfügbarkeit, eine gesteigerte Produktivität in der Entwicklung, bessere Skalierbarkeit und die Entkopplung von der unterliegenden Hardware. Auf einige der Vorteile wird im Folgenden noch einmal eingegangen.

9.1 Kürzere Time to Market

Die Time to Market wird, verglichen mit dem Ist-Zustand, erheblich verkürzt. Diese Zeitersparnis wird durch die Automatisierung des größten Teils der Deployment-Pipeline erzielt. Auch durch die Optimierung des Einsatzes der automatisierten Tests und durch die Optimierung der manuell durchzuführenden Tests, wird der Zeitaufwand stark reduziert. Eine erhebliche Auswirkung hat hier *die Testwoche* / *der Testzeitraum*, mit einer Dauer von noch maximal zwei Tagen im Soll-Zustand. Auch gibt es aus Rücksicht auf die Anwender keine Wartezeiten mehr. Wo zuvor im Ist-Zustand ein Deployment mit Anwendern abgestimmt werden musste, kann dies im Soll-Zustand direkt ohne Absprachen durchgeführt werden.

Die Umsetzung der Lösungsstrategie wird voraussichtlich eine Verkürzung der Time to Market von ca. 10 Tagen auf dann 2,54 Tage ermöglichen.[83] **Es werden so annähernd 75% des ursprünglichen Zeitaufwands eingespart.** Das bedeutet eine sehr gute Verbesserung gegenüber dem Ist-Zustand.
Es wäre sogar möglich, einen weiteren Tag für die Durchführung des gesamten Deployment-Prozesses einzusparen, würde man den Testzeitraum nach dem Test Deployment noch besser koordinieren und entsprechend kürzer definieren.

[83] Die Tabelle 9.1 zeigt eine genauere Gegenüberstellung des Zeitaufwands von Ist- und Soll-Zustand.

Tabelle 9.1: Vergleich Zeitaufwand von Ist- und Soll-Zustand des Deployment-Prozesses

Zeitaufwand in Tagen Ist-Zustand	**Zeitaufwand in Tagen Soll-Zustand**	**Tätigkeit**
ca. 0,5 Tage*	ca. 0,5 Tage* (hauptsächlich bedingt durch manuelle Tests)	Durchführung Test Deployment
5 Tage	2 Tage	Testwoche / Testzeitraum: Tests an bereitgestellter Testversion
2 Tage	keiner	Wartezeit für passenden Termin für Produktiv Deployment (wegen speziellem Power User)
ca. 0,5 Tage*	ca. 0,03125 Tage* (entspricht 15 Minuten)	Durchführung Produktiv Deployment
ca. 10 Tage (inklusive zwei Tagen Wochenende)	**ca. 2,54 Tage (aufgerundet auf 2 Nachkommastellen)**	**Gesamtaufwand Deployment-Prozess**

Anmerkung zu Tabelle 9.1:
In dem angegebenen Gesamtzeitaufwand des Ist-Zustands ist ein Wochenende mit eingerechnet worden, da dieser mehr als 5 Tage beträgt und somit automatisch ein Wochenende enthält. Beim Gesamtzeitaufwand des Soll-Zustands ist dies nicht der Fall. Es wäre nicht sinnvoll, ein Deployment über ein Wochenende durchzuführen und das Deployment dadurch künstlich zu verlängern.
In der Tabelle entspricht ein Arbeitstag 8 Stunden. Um Personentage zu errechnen, müssen die angegebenen Werte mindestens mit zwei multipliziert werden, da die Arbeit bei delegs immer mindestens in Pairs durchgeführt wird.
Bei angegebenen Zeitaufwänden, die mit * versehen sind, handelt es sich um vom Autor geschätzte Werte.

9.2 Höhere Verfügbarkeit

Es gibt kein Service Level Agreement (SLA) mit den Kunden bzw. den Anwendern des delegs-Editors, in dem die Verfügbarkeit der Anwendung garantiert wird und Zeitkontingente für die Ausführung von Wartungsarbeiten festgelegt sind. Auch gibt es keine Werte darüber, wie hoch die Verfügbarkeit bzw. die Ausfallzeiten und die Zeit für Wartungsarbeiten des Editors für das vergangene Jahr oder überhaupt für einen festgelegten Zeitraum waren. Entsprechend fehlt eine Basis für den Vergleich mit erwarteten Werten des Systems im Soll-Zustand. Allerdings werden sich die Ausfallzeiten, die durch das Deployment einer neuen Produktivversion entstehen, nach der Umsetzung der Lösungsstrategie, massiv verringern. Es wird zwar noch gelegentlich einen Ausfall des Systems geben, wenn eine Änderung des Datenbankschemas vorgenommen wird, aber dies ist ebenso vom Deployment im Ist-Zustand bekannt. In allen anderen Fällen wird es keinen Ausfall mehr geben, außer beim eventuellen Auftreten von Fehlern. Kommt es zu Fehlerfällen beim Produktiv Deployment, bietet der Soll-Zustand aber den Vorteil, dass auf eine alte Version zurückgewechselt werden kann, im Ist-Zustand war dies nicht möglich.
Für Anwender wird mit der neuen Lösungsstrategie ein Deployment einer Version also größtenteils transparent sein.

9.3 Gesteigerte Produktivität des Tech-Teams

Die Umsetzung der Lösungsstrategie wird zu einer gesteigerten Produktivität des Tech-Teams führen. Das Team wird sehr viel weniger Zeit für die Durchführung des Deployments eingespannt sein. Es wird in diversen Bereichen, wie beispielsweise Operations, sehr viel weniger Wartezeit für die Entwickler geben und Arbeitsvorgänge und Tests können effizienter durchgeführt werden. Sollte es zu Fehlern kommen, werden diese besser nachvollziehbar sein und somit schneller behoben werden können.

9.4 Skalierbarkeit

Das neue System ließe sich sehr viel einfacher skalieren, als es mit dem System des Ist-Zustands möglich ist. Dies gilt sowohl in der Horizontalen als auch in der Vertikalen, was dem Aufbau des Systems und der Container-Technologie zu verdanken ist. Im Vergleich mit dem System des Ist-Zustands wird so beispielsweise sehr viel weniger Arbeitsaufwand nötig sein, um eine vertikale Skalierung vorzunehmen, ohne dass die Anwendung dabei ausfällt.

9.5 Entkopplung von unterliegender Hardware

Das neue verteilte System ist nicht mehr abhängig von der unterliegenden Hardware und somit auch nicht mehr vom Anbieter einer solchen. Container des Systems können jederzeit und ohne großen Aufwand umgezogen werden.

9.6 Bruchfreies Deployment bei Änderung des Datenbankschemas

An dieser Stelle soll grob skizziert werden, wie man das in der Lösungsstrategie vorgestellte verteilte System erweitern könnte, um auch im Falle einer Änderung des Datenbankschemas ein bruchfreies Deployment zu ermöglichen. Es handelt sich hierbei nicht um ein vollständiges Konzept. Es braucht definitiv noch eine weitere Bearbeitung.
Man könnte das System um zwei Komponenten erweitern und die Datenbank komplexer gestalten.
Das ganze würde dann wie folgt aussehen:

- **Message-Queue-System**
  Das Message-Queue-System (MQ-System) kommuniziert mit den *Blue* und *Green* Software-Servern und einem *Datenbanken-Adapter*, der unten beschrieben wird und verfügt über mehrere Queues. Nachrichten von den Software-Servern sind mit deren Versionsnummern versehen. Nachrichten vom Testsystem werden zusätzlich in der Versionsnummer kenntlich gemacht, damit diese nicht fälschlicherweise den Produktivbetrieb beeinflussen. Im MQ-System werden alle Nachrichten in einer einzigen Queue gespeichert, unabhängig davon, ob sie von *Blue*, *Green* oder *Test* Servern stammen. Diese werden dort vorgehalten, bis sie vom Datenbanken-Adapter abgeholt werden. Nachrichten vom Datenbanken-Adapter an einen Software-Server werden im MQ-System hingegen in spezifische Queues geladen. Es existiert dort jeweils eine Queue für die entsprechenden Versionen Blue, Green und Test. Sie werden dort vorgehalten, bis der entsprechende Server diese abgeholt hat.
  Das MQ-System wird gebraucht, um beim Bereitstellen einer neuen Version des delegs-Editors inklusive Änderung am Datenbankschema den Datenbanken-Adapter austauschen zu können, ohne dass der Produktivbetrieb davon beeinträchtigt wird. Beim Austauschen wird die Kommunikation zwischen Servern und Datenbanken kurzzeitig unterbrochen, was durch das MQ-System abgepuffert wird. Nach dem Bereitstellen des

neuen Adapters arbeitet dieser die gepufferten Nachrichten ab und danach läuft die Kommunikation wieder normal weiter.

- **Datenbanken-Adapter**
  Der Datenbanken-Adapter liest Nachrichten von den Software-Servern aus dem MQ-System aus. Die Nachrichten enthalten Lese- und Schreibtransaktionen der Software-Server auf die Datenbanken. Aufgabe des Datenbank-Adapters ist es, die Transaktionen auf *beiden* unten beschriebenen Datenbanken *Blue* und *Green* zu spiegeln. Der Adapter ist für die Replikation zuständig und konvertiert die Transaktionen entsprechend der Schemata der Datenbanken, wenn sich diese gerade unterscheiden. Er muss also für einen solchen Fall mit einer entsprechenden Replikationslogik ausgestattet werden.
  Ergebnisse aus den Transaktionen auf den Datenbanken verpackt der Adapter in Nachrichten mit entsprechenden Versionsnummern und schickt diese zurück in Richtung Software-Server, an das MQ-System.
  Die Kommunikation zwischen den Testversionen von Server und Datenbank leitet der Datenbank-Adapter ohne Replikation durch.
  Wird mit der Bereitstellung einer neuen Version auch ein verändertes Datenbankschema eingeführt, muss auch eine neue Version des Datenbank-Adapters bereitgestellt werden.

- **Zwei synchron gehaltene Datenbanken inklusive Slaves**
  Für die *Blue* und *Green* Version der Software-Server wird jeweils eine Datenbank gestellt. Jede dieser Datenbanken betreibt zusätzlich Master-Slave-Replikation, wie auch schon für den Soll-Zustand beschrieben..
  Die Datenbanken werden synchron gehalten. Ausnahme machen hier natürlich die Daten, die Aufgrund unterschiedlicher Schemata in den Datenbanken auseinanderdriften, wie beispielsweise, wenn ein zusätzliches Feld in einer Datenbanktabelle eingeführt wurde.
  Durch die zwei parallellaufenden Datenbanken, können Änderungen am Datenbankschema einfacher umgesetzt werden. So gibt es den gleichen, wie schon im Soll-Zustand für die Anwendung beschriebenen Vorteil von Blue/Green-Deployment:
  Im Falle eines Rollbacks einer Version, bei der das Datenbankschema geändert wurde, kann einfach wieder auf die Datenbank der alten Version gewechselt werden, genau wie mit dem Software-Server. Dies geschieht ohne Operationen, die eine Konvertierung des Schemas wieder rückgängig machen oder die einen alten Produktivbestand wieder einspielen. Die ältere Serverversion kann einfach wieder zusammen mit der älteren Datenbank eingesetzt werden und Teile der Daten, die seit dem Wechsel auf die neue

Version bereits geschriebenen wurden, bleiben dabei sogar erhalten.
Auch für das Test Deployment wird mindestens eine Test Datenbank eingesetzt. Bei Bereitstellung einer Testversion arbeitet diese, wie schon für den Soll-Zustand beschrieben, auf einer Kopie der aktuellen Datenbank.

Das hier grob skizzierte System würde auch einen besseren Umgang mit der im Unterabschnitt 8.3.5 Änderungen am Dokumentformat beschriebenen Problematik ermöglichen.

Generell ist allerdings zu bedenken, dass die Komplexität des Systems, verglichen mit dem Soll-Zustand, extrem ansteigen würde. Es wäre zu komplex, sodass es für das delegs-Projekt nicht sinnvoll erscheint.
Denn mit der Einführung eines erweiterten Systems müsste analog auch ein Konzept zur Erweiterung des Deployment-Prozesses und der Pipeline erarbeitet und parallel umgesetzt werden. Auch dadurch würde sich die Komplexität erhöhen. Eventuell könnte der Aufwand für ein Deployment verringert werden, wenn der Datenbank-Adapter modular aufgebaut wäre und er über ein Blue Modul und ein Green Module verfügen würde, die die Konvertierungslogik enthielten und die unabhängig voneinander ausgetauscht werden könnten.
Zusätzlich würde auch das Testen aufwendiger werden. So müsste beispielsweise bei einem Test Deployment inklusive verändertem Datenbankschema dann auch noch der Austausch des Datenbank-Adapters und dessen Kommunikation mit den Datenbanken und dem MQ-System getestet werden.

Fortes fortuna adiuvat[84]

[84] "[...] Lässt sich übersetzen als 'Den Mutigen (Tüchtigen) hilft das Glück'." (Wikipedia (2017a))

10 Ausblick

In diesem Kapitel wird ein Ausblick gegeben, wie die Lösungsstrategie, die in dieser Arbeit vorgestellt wurde, noch erweitert werden könnte.
Im IT-Bereich werden ständig neue Ideen, Konzepte und Technologien vorgestellt. So steht eine große Auswahl an Möglichkeiten auf verschiedensten Ebenen zur Verfügung, die Lösungsstrategie zu erweitern. In diesem Ausblick wird sich jedoch darauf beschränkt, Themen aufzubringen, die relativ nahe an der Lösungsstrategie sind.
Eines der wichtigsten Themen für den Ausblick ist sicherlich die Umsetzung der Lösungsstrategie, es werden aber auch noch einige weitere Anregungen gegeben.

10.1 Umsetzung der Lösungsstrategie

Die Umsetzung der Lösungsstrategie ist eine große Unternehmung, die in vielen Bereichen des delegs-Projekts Veränderungen vornimmt. Vor der Umsetzung der Lösungsstrategie wird empfohlen, gründlich den dabei anfallenden Aufwand zu schätzen und diesen dem erwarteten Nutzen gegenüberzustellen.

Es wird empfohlen, auch bei der Umsetzung weiterhin agil vorzugehen. Ein wichtiger Grund dafür ist, dass so während der Umsetzung sehr viel weniger Gefahr besteht, sich in einem Durcheinander an Änderungen zu verlieren. Außerdem kann dann, falls nötig, die Lösungsstrategie bzw. der Kurs deren Umsetzung schnell an schwierige Umstände angepasst werden. Ein schrittweises Vorgehen ist also sinnvoll und außerdem nötig, um die Anwendung während der Umstellung kontinuierlich verfügbar zu halten.

Die nötigen Änderungen können in vier Bereiche unterteilt werden:

- Änderungen beim Team und dessen Vorgehen
- Änderungen bei der unterliegenden Hardware und deren Anbieter
- Änderungen an Umgebungen und weiterer Infrastruktur
- Änderungen an der Anwendung selbst

Diese werden in den folgenden Unterabschnitten kurz erörtert. Viele der Umsetzungsschritte der verschiedenen Bereiche können parallel durchgeführt werden, teilweise sogar innerhalb eines Bereichs, andere stehen zueinander in Abhängigkeit.

10.1.1 Änderungen bei Team und Vorgehen

Ein Großteil der Schritte, die in diesem Bereich anstehen, werden hier aufgezählt. Folgeschritte sind als Unterpunkte dargestellt:

- Integration von konsistentem GitFlow
- Aufarbeitung der manuellen und automatisierten Tests
  - Erstellung der Test-Suites
  - Umstrukturierung der Testwoche / des Testzeitraums
  - Einsatz von Datenbank-Images für automatisierte Tests
    Dies kann erst geschehen, nachdem die Datenbanken in Container umgezogen wurden.
- Schulung im DevOps-Rollenmodell
- Schulung in JavaScript
  - Umstellen auf hauptsächliches Arbeiten mit *Super Dev Mode*
- Automatisierung manueller Tätigkeiten durch Skripte
  Natürlich nur dort, wo auch möglich und sinnvoll.
- Einsatz des neuen CE-Servers
  Schrittweises Vorgehen, parallel zur Fertigstellung des CE-Servers (siehe unten): zuerst Continuous Integration, später dann Continuous Deployment.
- Suche eines Tools zum Verwalten von Iterationen und Backlog
  Das Tool muss auch vom Lehrteam bedient werden können und sollte möglichst auch mit dem CE-Tool zusammen betrieben werden können.

Tatsächlich wurde bereits gegen Ende der Entwicklung der Lösungsstrategie, in Zusammenarbeit zwischen Tech-Team und Autor (der Teil des Tech-Teams ist), konsistentes GitFlow als neuer Workflow für die Entwicklung eingeführt. Dies war unabhängig von der Umsetzung der

Lösungsstrategie generell wichtig für ein strukturiertes Vorgehen während der Entwicklung. Zugleich wurde so aber auch eine wichtige Basis für die weitere Umsetzung der Lösungsstrategie geschaffen.

10.1.2 Änderungen unterliegender Hardware und Anbieter

In diesem Bereich müssen hauptsächlich Entscheidungen getroffen werden. Hierfür wurde ein Entscheidungsbaum angelegt, der im Anhang d. Entscheidungsbaum: Änderungen bei Hardware-Anbieter zu finden ist. Er leitet durch alle offenen Fragen hin zur daraus resultierenden Umsetzung.

10.1.3 Änderungen an Umgebungen und weiterer Infrastruktur

Ein Großteil der Schritte, die in diesem Bereich anstehen, werden hier aufgezählt. Folgeschritte sind als Unterpunkte dargestellt:

- Docker-Engine auf Entwicklungsumgebung einrichten
- Entscheiden, welches Continuous Everything Tool (CE-Tool) eingesetzt werden soll
  - Continuous Everything-Server (CE-Server) aufsetzen
    schrittweise zuerst Continuous Integration-Unterstützung einrichten
    und danach Continuous Deployment-Unterstützung einrichten
- Backup-Repository organisieren und einrichten

10.1.4 Änderungen an der Anwendung

Ein Großteil der Schritte, die in diesem Bereich anstehen, werden hier aufgezählt. Folgeschritte sind als Unterpunkte dargestellt:

- Konzept für Monitoring erarbeiten
  Siehe dazu den Unterabschnitt 10.1.5 Konzept für Monitoring & Rückkopplung durch die Deployment-Pipeline
- Load-Balancer vorschalten für Test- und Produktivbetrieb.
  Dieser wird mit einer Wartungsarbeiten-Webseite versehen.
  - Widget für Client implementieren
    Dieses warnt vorerst nur vor Ausfall bei Deployment einer neuen Produktivversion.
    Entsprechend müssen die nötigen Änderungen am Server vorgenommen werden.

- Master-Slave-Replikation für Produktiv-DB einrichten
- Systemkomponenten Schritt für Schritt in Docker-Container verschieben
  Erst umsetzen, nachdem die Entscheidungen zu den Änderungen an der unterliegenden Hardware und deren Anbieter getroffen wurden und die Hardware einsatzbereit ist (siehe dazu Unterabschnitt 10.1.2 Änderungen unterliegender Hardware und Anbieter).
  - Umbau der Anwendung hin zum Blue/Green-Deployment
    * Zugangsdaten für Test- und Produktivbetrieb auftrennen
    * 12-Stunden-Frist einführen
      Dafür muss das Widget erweitert und entsprechend die Server-Konfiguration angepasst werden.

10.1.5 Konzept für Monitoring & Rückkopplung durch die Deployment-Pipeline

Wie schon im Unterabschnitt 8.4.6 Monitoring & Rückkopplung durch die Pipeline beschrieben, muss noch ein Konzept für die Rückkopplung der Deployment-Pipeline und deren Monitoring erarbeitet werden. Hier sollte direkt für das gesamte Monitoring des neuen verteilten Systems ein Konzept entworfen werden. Wichtig ist, dass nicht einfach nur Daten aufgezeichnet werden. Die Daten müssen in für Menschen verständliche Informationen aufbereitet und wo möglich visualisiert werden. Eine gute Visualisierung ermöglicht Menschen, ein schnelleres und korrekteres Erfassen und Verarbeiten von Informationen und somit auch ein schnelleres und korrekteres Reagieren, so zum Beispiel bei Problemen.

10.2 Weitere Anregungen

Folgend werden noch einige weitere Anregungen gegeben, wie der Lösungsansatz zusätzlich vorangetrieben werden kann:

- **Verbesserung der Team-Kommunikation**
  Es wäre sinnvoll für das zentrale Team, eine gute Lösung zu finden, Informationen unter den Teammitgliedern einfach zu verteilen. Denn eine solche Lösung gibt es für das Team noch nicht. Hauptgrund hierfür ist, dass eigentlich nie alle Teammitglieder gleichzeitig am Arbeitsplatz sind.
- **Selbständiges Verwalten von Zertifikaten**
  Digitale Zertifikate für die/den Server könnten vom Tech-Team selbstständig verwaltet

werden. Holen und Verwalten der Zertifikate wäre automatisiert möglich und könnte beispielsweise unter Nutzung der *https://letsencrypt.org* Webseite der *Internet Security Research Group* realisiert werden.

- **Automatisierte GUI-Tests**
  Es könnte überprüft werden, ob GUI-Tests automatisiert werden können. Allerdings gestaltet sich das Aufsetzen und das Warten der Tests oftmals als sehr aufwändig. Letzteres ist immer dann nötig, wenn sich die GUI der Anwendung ändert.
  Auch können automatisierte GUI-Test kaum einen menschlichen Tester ersetzen. Es könnte aber eine Möglichkeit sein, automatisierte GUI-Tests einzusetzen, die die wichtigste Grundfunktionen der Anwendung testen. So würde die Wiederholbarkeit von Tests erhöht werden.

11 Fazit

Die anfängliche Fragestellung an diese Arbeit lautete, wie groß der zu erwartende Zeitgewinn durch die Umwandlung des delegs-Editor-Softwareprojekts in ein DevOps getriebenes Projekt wäre. Daraus ergab sich die für diese Arbeit formulierte Zielsetzung, eine entsprechende Lösungsstrategie zu entwickeln und auf deren Basis die Fragestellung zu beantworten.
In der in dieser Arbeit vorgestellten Lösungsstrategie werden viele Änderungen am Projekt beschrieben. Einige mögen auf den ersten Blick so wirken, als würden sie nicht der Fragestellung an diese Arbeit zuspielen. Dem ist aber nicht so. Denn es wurde eine ganzheitliche Lösungsstrategie zur Einführung von DevOps in den Projektbetrieb entwickelt, mit dem Ziel die Time to Market zu verkürzen, ganz im Sinne der DevOps-Philosophie.

Bei der Analyse des Softwareprojekts wurden Hindernisse in den unterschiedlichsten Bereichen aufgedeckt, die direkt oder indirekt die Geschwindigkeit der Entwicklung und des Deployments der delegs-Editor-Anwendung negativ beeinflussen. Die drei größten Hindernisse sind dabei der ineffizient durchgeführte Deployment-Prozess, die Abhängigkeit von der unterliegenden Hardware und die personelle Fluktuation.

Mit der beschriebenen Lösungsstrategie wird das Ziel dieser Arbeit erreicht. Die in ihr vorgestellten Veränderungen betreffen das komplette Softwareprojekt und verwandeln dieses bei Umsetzung der Lösungsstrategie in ein DevOps getriebenes Projekt. Dies gelingt zwar nicht zu 100%, aber zu einem sehr großen Teil. Von der Lösungsstrategie betroffen sind Entwicklung und Deployment und jeweils deren kompletter Kontext, wie Teamstruktur, Vorgehensweisen, Umgebungen, Infrastruktur und Tools.

Von den erhobenen Anforderungen konnte nur eine aufgrund des begrenzten Rahmens dieser Arbeit nicht mit in die Lösungsstrategie einbezogenen werden. Es handelt sich dabei um die Anforderung A04 - *Rückkopplung durch Deployment-Pipeline*, die das Thema *Monitoring* betrifft. Alle anderen Anforderungen wurden bearbeitet und so bieten diese ein solides Fundament für die Lösungsstrategie.

Für zwei der drei größten oben genannten Hindernisse bietet die Lösungsstrategie Abhilfe: Sie beschreibt einen komplett neu entworfenen, hoch automatisierten und effizient ausgelegten Deployment-Prozess. Außerdem beschreibt sie, wie die Anwendung vollständig aus der

Abhängigkeit der unterliegenden Hardware gelöst werden kann, durch das Aufsetzen auf eine Container-Technologie.
Bedingt durch den Rahmen des delegs-Projekts ist es nicht möglich, das Problem der personellen Fluktuation zu beheben und so bleibt diese Ausgangssituation des Projekts, auf die sich die Fragestellung an diese Arbeit stützt, erhalten.

Durch den großen Umfang von Analyse und Lösungsstrategie war es in dieser Arbeit nicht mehr möglich, wie anfänglich vorgesehen, eine Teilumsetzung der Lösungsstrategie vorzunehmen. Die agile Herangehensweise an diese Arbeit ermöglichte aber das Anpassen des Fokus zugunsten einer vollständigeren Analyse und Lösungsstrategie - allerdings auf Kosten der Umsetzung. Diese Anpassung war sinnvoll, denn die Beantwortung der Fragestellung dieser Arbeit wird dadurch kaum beeinflusst. Ursprünglich vorgesehen war nur eine *anfängliche* Umsetzung. Aus einer solchen *Teilumsetzung* waren nur wenige aussagekräftige Informationen zum Zeitgewinn durch die Umgestaltung des Projekts zu erwarten. Die Entwicklung einer vollständigeren Lösungsstrategie führt hingegen dazu, dass diese sehr viel mehr Aussagekraft bekommt.

Unerwartet bei der Entwicklung der Lösungsstrategie war der Aufwand, der durch die Einarbeitung der Erfüllung der Anforderung A01 - *Deployment wird bruchlos durchgeführt* entstanden ist. Sie sticht hervor aus den anderen Anforderungen und aus der Arbeit selbst, da sie ihren Ursprung nicht in der DevOps-Philosophie hat. Auch ist gegen Ende der Entwicklung der Lösungsstrategie klargeworden, dass ihre Erfüllung nicht notwendig für die Beantwortung der Fragestellung oder das Erreichen des Ziels dieser Arbeit ist.
Trotzdem erfüllt die Lösungsstrategie die Anforderung. Sie stellt ein neues verteiltes System vor, das ein bruchloses Deployment einer neuen Anwendungsversion in fast allen Fällen ermöglicht.

Eine präzise Beantwortung der Fragestellung ist auf der Grundlage dieser Arbeit nicht möglich. Zu viele Zeitfaktoren können nur als Schätzung und teilweise gar nicht erhoben werden und Vergleiche zwischen Ist- und Soll-Zustand ermöglichen daher nur eine vage Annäherung.
Die Auswertung der Lösungsstrategie ergab, das ihre Umsetzung eine geschätzte Zeiteinsparung von annähernd 75% beim Deployment bedeuten würde. Einen Großteil dieser Zeiteinsparung machen die effizienter ausgelegten Tests aus, speziell der um mehr als die Hälfte verkürzte Testzeitraum. Außerdem bewirken auch das Wegfallen bzw. die Verkürzung von Wartezeiten größere Zeiteinsparungen.
Die Umsetzung der Lösungsstrategie führt voraussichtlich zu einer gesteigerten Produktivität des technischen Teams in diversen Bereichen. Das Team muss nach der Umsetzung, bedingt durch die starke Automatisierung, sehr viel weniger aktiv den Deployment-Prozess durch-

führen und kann sich so mehr auf den Entwicklungsprozess konzentrieren, der durch etliche beschriebene Optimierungen effektiver durchgeführt werden kann.

Für das delegs-Projekt kann also durch die Umsetzung der Lösungsstrategie insgesamt mit einer massiven Zeitersparnis gerechnet werden. Wie gut sich diese Erkenntnis für andere Projekte verallgemeinern lässt, ist allerdings fragwürdig. Fakt ist aber, dass die DevOps-Philosophie aus gutem Grund entstanden ist und eingesetzt wird, denn sie optimiert die Time to Market gegenüber anderen Projekten, die nicht DevOps getrieben sind. Diese Arbeit zeigt ein weiteres Fallbeispiel auf, dass diesen Nutzen von DevOps klar bestätigt.

Interessant im Hinblick auf die Weiterführung dieser Arbeit wäre es, auszuwerten, wie hoch der geschätzte Aufwand für die Umsetzung der Lösungsstrategie ist und ob dieser durch die erwartete höhere Produktivität gerechtfertigt wird. Nach einer vollständigen Umsetzung, wenn das Projekt eine Weile entsprechend durchgeführt wurde, könnte am konkreten Fall evaluiert werden, wie viel Zeitgewinn und erhöhte Produktivität tatsächlich erreicht werden.

Für Rückfragen ist der Autor dieser Arbeit erreichbar unter:
fritz.oscar.berngruber@gmx.de

Literaturverzeichnis

[Amazon Inc. 2017a] Amazon Inc.: *Amazon EC2: Preise für EBS*. 2017. – URL https://aws.amazon.com/de/ebs/pricing/. – Zugriffsdatum: 2017-04-23

[Amazon Inc. 2017b] Amazon Inc.: *Amazon EC2: Preise für Reserved Instances*. 2017. – URL https://aws.amazon.com/de/ec2/pricing/reserved-instances/pricing/. – Zugriffsdatum: 2017-04-23

[Bass u. a. 2015] Bass, Len ; Weber, Ingo ; Zhu, Liming: *DevOps: A Software Architect's Perspective (SEI Series in Software Engineering)*. Pearson Education, Inc., 2015. – 315 S. – ISBN 0134049845

[Driessen 2010] Driessen, Vincent: *GitFlow*. 2010. – URL http://nvie.com/posts/a-successful-git-branching-model/. – Zugriffsdatum: 2017-03-22

[Duden 2017] Duden: *Schreibung von Fremdwörtern aus dem Englischen*. 2017. – URL http://www.duden.de/sprachwissen/sprachratgeber/schreibung-von-fremdwoertern-aus-dem-englischen. – Zugriffsdatum: 2017-01-20

[Fehling und Prof. Dr. Leymannm 2017] Fehling, Christoph ; Prof. Dr. Leymannm, Frank: *Provisionierung von Ressourcen*. 2017. – URL http://wirtschaftslexikon.gabler.de/Archiv/1057702/provisionierung-v3.html. – Zugriffsdatum: 2017-01-06

[Fowler 2009] Fowler, Martin: *FeatureBranch*. 2009. – URL https://martinfowler.com/bliki/FeatureBranch.html. – Zugriffsdatum: 2017-03-25

[Fowler 2013] Fowler, Martin: *DeplyomentPipeline*. 2013. – URL https://martinfowler.com/bliki/DeploymentPipeline.html. – Zugriffsdatum: 2017-03-12

[Fowler 2017] Fowler, Martin: *Martin Fowlers Webseite*. 2017. – URL https://martinfowler.com. – Zugriffsdatum: 2017-05-15

[Gillis 2015] Gillis, Tom: *Kosten: Data Center vs. Public Cloud.* 2015. – URL https://www.forbes.com/sites/tomgillis/2015/09/02/cost-wars-data-center-vs-public-cloud/{#}6b9727ee923f. – Zugriffsdatum: 2017-04-23

[Google Inc. 2017] Google Inc.: *Google Analytics.* 2017. – URL https://analytics.google.com. – Zugriffsdatum: 2017-04-29

[Grüner 2017] Grüner, Sebastian: *Golem: Cloud-Hoster löscht versehentlich Primärdatenbank.* 2017. – URL https://www.golem.de/news/digital-ocean-cloud-hoster-loescht-versehentlich-primaerdatenbank-1704-127210.html. – Zugriffsdatum: 2017-04-25

[GWT 2017] GWT: *GWT Webpage: Überblick.* 2017. – URL http://www.gwtproject.org/overview.html. – Zugriffsdatum: 2017-01-15

[Hänel-Faulhaber 2015] Hänel-Faulhaber, Prof. D.: Projektantrag: Schriftspracherwerb gehörloser Menschen zur Förderung inklusiver Teilhabe am Arbeitsmarkt - Kurzfassung. (2015), S. 1–7

[Host Europe 2017] Host Europe: *Host Europe: Preise für VPS.* 2017. – URL https://www.hosteurope.de/Server/Virtual-Server/. – Zugriffsdatum: 2017-04-23

[Lakuzz und Mohdit 2015] Lakuzz ; Mohdit, Taoufik: *stack overflow: Continuous Integration vs. Continuous Delivery vs. Continuous Deployment.* 2015. – URL http://stackoverflow.com/questions/28608015/continuous-integration-vs-continuous-delivery-vs-continuous-deployment. – Zugriffsdatum: 2016-12-02

[Musk Elon 2014] Musk Elon: *Elon Musk at the 2014 Export-Import Bank conference in Washington, DC on Tesla and SpaceX & keeping engineering close to production.* 2014. – URL https://www.c-span.org/video/?319035-4/elon-musk-tesla-spacex. – Zugriffsdatum: 2016-11-25

[Sadalage 2016] Sadalage, Pramod: *Refactoring Databases.* 2016. – URL http://databaserefactoring.com. – Zugriffsdatum: 2017-04-17

[Wikipedia 2017a] Wikipedia: *Fortes fortuna adiuvat*. 2017. – URL https://de.wikipedia.org/wiki/Fortes_fortuna_adiuvat. – Zugriffsdatum: 2017-05-11

[Wikipedia 2017b] Wikipedia: *Liste von Gebärdensprachen*. 2017. – URL https://en.wikipedia.org/wiki/List_of_sign_languages. – Zugriffsdatum: 2017-03-22

[Züllighoven 2005] Züllighoven, Heinz: *Object-Oriented Construction Handbook*. Elsevier, 2005. – 500 S

Zitate aus englischsprachigen Quellen sind Eigenübersetzungen vom Autor dieser Arbeit.

Glossar

Commit:
Bedeutung: Ein Entwickler macht einen Beitrag (englisch: *Commit*) zu der zu entwickelnden Software. Dieser Beitrag wird der bestehenden Codebasis in Form von neu entwickeltem Code hinzufügt. Typischerweise schickt er die Änderungen an ein Versionsverwaltungssystem (wie z.B. *Git*). Das Versionsverwaltungssystem registriert den Commit und fügt ihn der Codebasis hinzu.

provisionieren:
Bedeutung: Eine Provisionierung durchführen.
Siehe: *Provisionierung*

Provisionierung:
Bedeutung: "engl.: Provisioning; Bezeichnung für die automatisierte Bereitstellung von IT-Ressourcen. Diese können danach durch Deprovisionierung wieder freigegeben werden." (Fehling und Prof. Dr. Leymannm (2017))

Smoke-Tests:
Bedeutung: (Automatisierte) Tests, die das generelle Zusammenspiel aller Komponenten einer Software testen. Die Tests decken dabei nicht die komplette Funktionalität einer Software ab. Sie sind dazu gedacht, zusammen mit den Unit-Tests ausgeführt zu werden und grob abzusichern, dass Änderungen an der Software nicht Fehler an der generellen Zusammenarbeit der Komponenten und deren Kernfunktionen erzeugt haben. Smoketests sollten keine lange Laufzeit haben und testen daher oft nur einige Positivfälle.

Anhang

a. Umgang mit englischen Fremdwörtern in der deutschen Rechtschreibung

Der folgende Artikel ist zitiert von der Webseite des Duden. Er wird in dieser Arbeit als Referenz für den Umgang mit englischen Fremdwörtern genutzt.

"**Schreibung von Fremdwörtern aus dem Englischen**

Im Grunde gelten bei der Frage Getrennt- oder Zusammenschreibung von Wörtern aus dem Englischen die gleichen Regeln wie bei deutschen Zusammensetzungen. Verbindungen aus zwei Substantiven werden zusammengeschrieben, also beispielsweise *Economyclass*, *Poleposition*, *Shoppingcenter*, *Braintrust*, *Bottleparty*. Alternativ dazu kann auch ein Bindestrich gesetzt werden, beide Substantive müssen dann natürlich großgeschrieben werden: *Economy-Class*, *Pole-Position*. Die Bindestrichvariante bietet sich vor allem dann an, wenn solche Verbindungen in der Zusammenschreibung schwer lesbar sind, wie etwa bei dem Wort *Desktoppublishing*, hier ist die Schreibung *Desktop-Publishing* zu empfehlen.

Ist der erste Bestandteil ein Adjektiv, kann zusammengeschrieben werden, wenn die Hauptbetonung auf dem ersten Teil liegt, daneben ist auch die Getrenntschreibung möglich: *Blackbox* oder *Black Box*, *Hotspot* oder *Hot Spot*. Wir empfehlen hier jeweils die Zusammenschreibung. Ansonsten gilt in Anlehnung an die Herkunftssprache nur die Getrenntschreibung: *High Fidelity*, *Electronic Banking*, *Top Ten*.
Bei Verbindungen aus Verb und Partikel sind Zusammen- und Bindestrichschreibung möglich: *Hangover*, *Blackout*, *Countdown*, *Handout* neben *Hang-over*, *Black-out* etc. Hier empfehlen wir auch die Zusammenschreibung.

Bei Aneinanderreihungen und Zusammensetzungen mit Wortgruppen muss grundsätzlich

mit Bindestrich durchgekoppelt werden: *Current-Account-Bereich*, *Public-Relations-Abteilung*, *Do-it-yourself-Programm*, *Multiple-Choice-Aufgabe*." (Duden (2017))

b. Technische Abhängigkeiten & eingesetzte Technologien im Ist-Zustand

Die im delegs-Projekt eingesetzten Technologien sind im Folgenden aufgelistet und teilweise mit Anmerkungen versehen.

Entwicklungsumgebung

Diese Liste enthält alle Technologien, die während der Entwicklung in den Entwicklungsumgebungen der Entwickler eingesetzt werden.

Bedeutung der Markierungen in der folgenden Auflistung:
[v] Technologien, die durch das Dokument zur Einrichtung des Entwicklerarbeitsplatzes **v**orgeschrieben sind
[e] Technologien, deren Einsatz für das Projekt **e**mpfohlen werden (sowohl vom Dokument zur Einrichtung des Entwicklerarbeitsplatzes als auch von Mitarbeitern)
[o] Technologien, deren Einsatz für das Projekt **o**ptional sind

- Java 1.7 [v]
- Eclipse Neon.2 [v]
- JUnit 4
  wird automatisch bei der Installation der angegebenen Eclipse Version mitgeliefert
- GWT SDK 2.7.0 [v]
- GWT Eclipse Plugin (aktuellste Version) [v]
- Git [v]
  das Repository des delegs-Projekts wird von der WPS gehostet
- Atlassian SourceTree [e]
  erlaubt die Nutzung von Git mit Hilfe einer GUI
- KDiff3 [o]
  Tool zum Feststellen von Änderungen an verschiedenen Dateiversionen,auch eingesetzt, um Merge Konflikte zu beheben, die bei der Arbeit mit Git auftreten können
- MySQL [v]
- MySQL Workbench [v]
  erlaubt das Verwalten und Bearbeiten von MySQL-Datenbanken

- PuTTY [v]
  genutzt für Kommandozeilenzugriff auf den delegs-Server
- Cyberduck [e]
  ein FTP-Client mit GUI, der das Verschieben von Dateien auf den Server vereinfacht
- Gpg4win 2.3.1 [v]
  Datei Ver- & Entschlüsselung
- Firefox Version 24.0 [v]
  in dieser Version wird noch das *GWT Developer Plugin* unterstützt
- GWT Developer Plugin für Firefox [v]
- Firebug Plugin für Firefox [v]
  Monitoring, Debugging & Bearbeitung von HTML, CSS &JavaScript bei aufgerufenen Webseiten
- CCleaner [o]
  eingesetzt in den Entwicklungsumgebungen, um temporäre Dateien zu löschen, die beim lokalen Testen von Client Sitzungen angelegt und nicht richtig abgeräumt wurden oder zu spät abgeräumt werden würden
- **Kommunikations- und Planungsunterstützung für das Team:**
  - xPlanner [v]
    von der WPS betriebenes, online erreichbares Tool zur Projektplanung, Im speziellen werden Backlogs zur Verfügung gestellt, wie sie in der agilen Softwareentwicklung eingesetzt werden. User Stories und können unter diesen Tasks angelegt werden.
  - Office, speziell Outlook [e]
  - https://www.commsy.uni-hamburg.de [v]
    Webseite der Universität Hamburg, hier werden teilweise Dokumente hinterlegt & Termine geplant. Unter anderem kann hier die Anleitung zur Einrichtung eines Entwicklerarbeitsplatzes gefunden werden.
  - WhatsApp [o]
- HiDrive [v]
  vom Lehrteam eingesetzte Cloud-Anwendung (ähnlich *Dropbox* oder *TeamDrive*) zum Austausch von Dateien

Es wird kein spezielles Betriebssystem vorgegeben, beim Stand dieser Arbeit haben alle Entwickler mit Microsoft Betriebssystemen gearbeitet.

Drittanbieter APIs

Die folgende Liste enthält alle Drittanbieter APIs, von denen das delegs-Projekt abhängig ist.

- c3p0-0.9.5.1
- c3p0-oracle
- commons-email-1.3.2
- commons-io-1.3.1
- iText-4.2.0
- javax.mail
- jdom
- jsoup-1.7.2
- log4j-1.2.16
- mchange-commons-java-0.2.10
- mysql-connector-java-5.1.18
- serializer-2.7.0
- simplecaptcha-1.2.1

Serverseitiger Test- und Produktivbetrieb

Die folgende Liste enthält alle Technologien, die im Test- und Produktivbetrieb serverseitig eingesetzt werden.

- Apache Tomcat 8
- Icinga mit Nagios (Monitoring)
- Fail2Ban (automatisierte Auswertung von Log-Dateien und daraus resultierende Aktionen wie Firewallkonfiguration oder Benachrichtigung per Mail)

c. Beschreibung des Sonderfalls im Workflow des Ist-Zustands

Mit dem Workflow kann es zu einem Sonderfall kommen, wenn eine Testversion bereitgestellt und in dieser ein Fehler gefunden wurde und zugleich schon an der Folgeversion der Anwendung weiterentwickelt wurde. Der Sonderfall macht den Umgang mit dem Workflow kompliziert.

Um den Sonderfall einfacher erklären zu können, wird ein solcher im folgenden Beispiel beschrieben. Die mit dem Test Deployment bereitgestellte Testversion erhält in dem Beispiel die Bezeichnung *v1.71*:
Nach der Bereitstellung wird v1.71 eine Woche lang getestet. Während dieser Zeit wird parallel schon mit der Entwicklung an der nächsten Version *v1.72* begonnen. Features, die für die v1.72 fertiggestellt werden, werden wie oben beschrieben auf den Master-Branch übertragen.
Muss nun an der Testversion v1.71 nachgebessert werden, wird der Stand der Entwicklung an der nächsten Version v1.72 vom Master-Branch auf einen neuen Branch verschoben, der in der Bezeichnung die Versionsnummer der neuen Version trägt, in diesem Beispiel: *v1.72_Master*. Der echte Master-Branch wird auf den Stand des angelegten Tags der Testversion v1.71 zurückgesetzt. Dieses Vorgehen ermöglicht dem Team, sowohl an der Fehlerbehebung der Version v1.71 als auch an der Entwicklung der neuen Version v1.72 unter Anwendung des oben beschriebenen Workflows fortzufahren. Ist die Fehlerbehebung abgeschlossen, wird das Tag der Version v1.71 auf den neuen Stand des Master-Branchs gesetzt und der v1.72_Master mit diesem zusammengeführt, sodass die behobenen Fehler aus v1.71 in die Entwicklung von v1.72 einfließen.
Wurde die Version v1.71 getestet, ohne dass weitere Fehler gefunden wurden, kann sie produktiv gestellt werden. Es wird ein Merging des behelfsweise angelegten v1.72_Master-Branchs in den *echten* Master-Branch vorgenommen und der v1.72_Master-Branch wird gelöscht. Danach läuft die Entwicklung an der neuen Version v1.72 mit dem *normalen* Workflow weiter.

d. Entscheidungsbaum: Änderungen bei Hardware-Anbieter

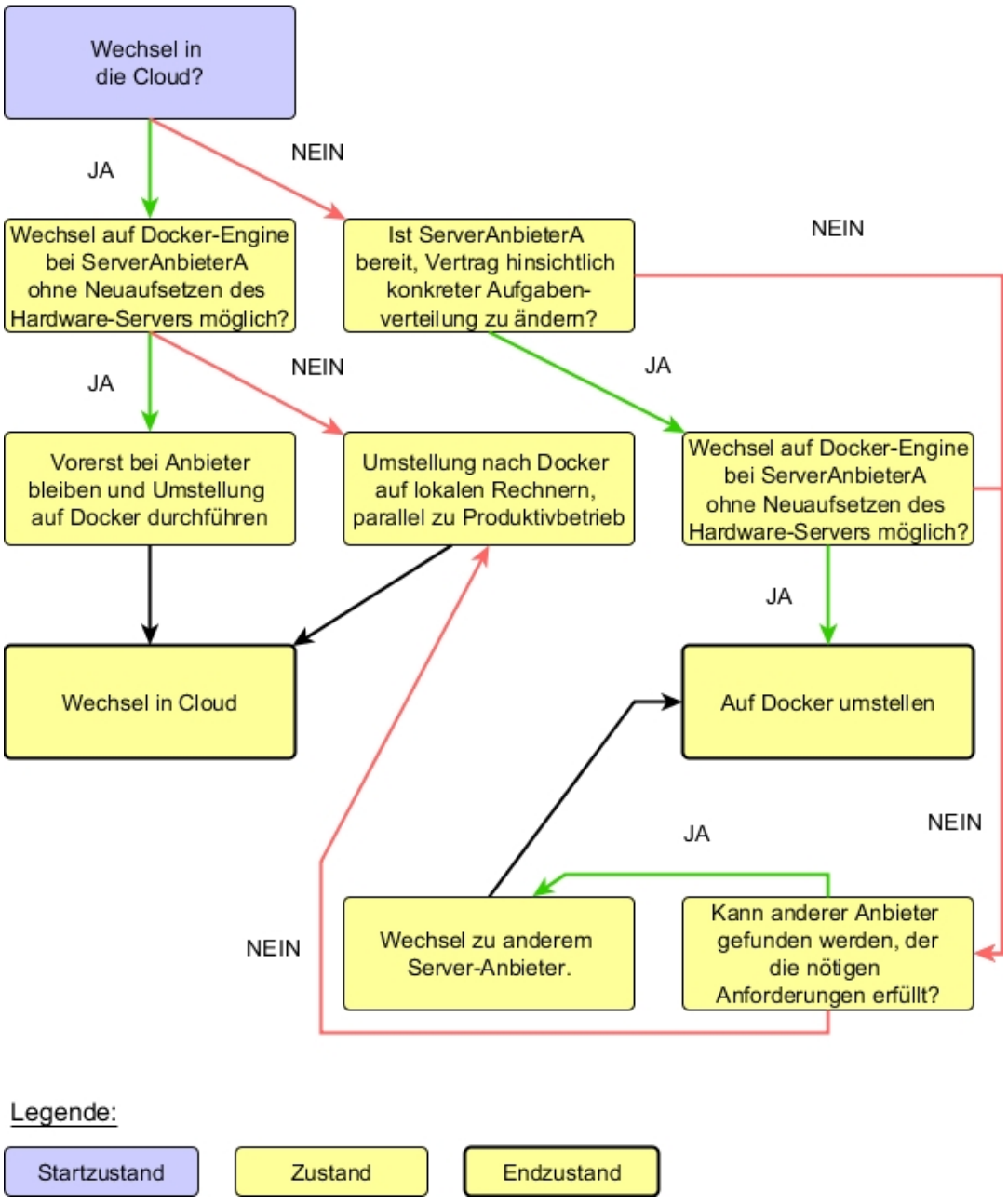

Abbildung 1: Entscheidungsbaum: Änderung bei Hardware-Anbieter

e. Interviews mit dem zentralen Team

Eine vollständige Aufführung aller geführter Interviews mit den Mitgliedern des Tech-Teams und des Lehrteams.
Alle intervieweten Personen dem Abdrucken ihres Interviews zugestimmt.

Interviews mit dem Tech-Team

Beim Entwurf des Interviews an das Tech-Team wurde bewusst darauf geachtet, die Fragen projektunabhängig aufzubauen und zu formulieren, sodass das Interview auch für andere Projekte nutzbar ist. Unter anderem wurden die Interview-Fragen an die SIG DEVOPS Gruppe der WPS weitergegeben.

Interview mit Jörn Koch

Interviewte Person: Jörn Koch
Kontext: delegs-Projekt / DevOps
Datum des Interviews: 11.12.2016
Da der Begriff "Deployment-Prozess" im IT-Bereich mit unterschiedlicher Bedeutung belegt ist, wurde dem Befragten zu Beginn des Interviews eine Definition zu dem Begriff gegeben. So wurde zwischen Interviewer und Befragten eine gemeinsame Basis geschaffen, auf der das Interview stattfinden konnten. Der Begriff "Deployment-Prozess" wurde wie folgt beschrieben:

> Der Deployment-Prozess ist Teil des Softwareentwicklungsprozesses und setzt genau an der Stelle ein, an der Code durch Softwareentwickler, wie z.B. in Form eines neuen Features, fertiggestellt wurde. Das Deployment ist das Bereitstellen dieses Codes in Form einer neuen Version in Test- oder Produktivumgebungen, z.B. auf einem virtuellen oder physischen Hardware-Server. Unter den Begriff *Deployment-Prozess* kann auch schon das Integrieren von Code in die Codebasis fallen, ebenso Produktivbetrieb von Software und deren Wartung.

Folgende Fragen wurden während des Interviews gestellt:
1.) Was ist Ihre Aufgabe im Projekt?
Antwort:
Projektleiter
2.) Gibt es mehrere Teams, die an der Software arbeiten?
Antwort:
—

2.1) Wenn ja: Wie viele Entwickler haben die jeweiligen Teams?
Antwort:
—

3.) Wie viele Entwickler arbeiten in Ihrem Projektteam?
Antwort:
Fünf.

4.) Administratorische Aufgaben im Projekt:

4.1) Gibt es Administratoren / Operators, die für Ihr Projekt zuständig sind oder die Teil Ihres Projektteams sind und im Team mitarbeiten?
Antwort:
Es gibt keine dedizierten Operators. Alle Aufgaben während des Deployment-Prozesses werden vom gesamten Tech-Team bearbeitet.

4.1.1) Wenn ja, wie viele und welche Aufgaben haben diese?
Antwort:
—

4.2) Bei mehreren Teams:
Gibt es Administratoren / Operators, die für andere Teams ihres Projekts zuständig sind oder die als Teammitglied in den anderen Teams mitarbeiten?
Antwort:
—

4.2.1) Wenn ja, wie viele und welche Aufgaben haben diese?
Antwort:
—

4.3) Werden administratorische Aufgaben / Operations-Aufgaben von Entwicklern übernommen?
Antwort:
Ja.

4.3.1) Wenn ja, welche?
Antwort:
Alle.

5.) Wie oft, also in welchem Rhythmus oder nach welchen Zeiträumen, ist in Ihrem Projekt ein Produktiv-Deployment von neuen Features, Versionen, oder Ähnlichem vorgesehen?
Antwort:
Ziel ist alle 2 Wochen bis maximal 4 Wochen.

5.1) Wenn Rhythmus vorgesehen: Wird der vorgesehene Rhythmus eingehalten?
Antwort:
Nein, der Rhythmus wird teilweise deutlich überschritten. Er bewegt sich zwischen 2 Wochen und 3 Monaten.

5.1.1) Wenn nein: Was sind die Gründe, die ein Deployment verzögern?
Antwort:
Features, die nicht rechtzeitig, nicht vollständig abgeschlossen werden.

5.2) Wenn nein: Sollte es Ihrer Meinung nach einen festen Deployment-Rhythmus geben?
Antwort:
—

5.2.1) Wenn ja, warum?
Antwort:
—

6.) Der Deployment-Prozess (bei mehreren Prozessen die folgenden Fragen je Prozess durchgehen):

6.1) Gibt es einen oder mehrere Deployment-Prozesse (z.B. Test- & Produktiv Deployment)?
Antwort:
Es gibt nur einen, Test- und Produktiv Deployment sind Teil eines großen Prozesses.

6.2) Sind Sie am Deployment-Prozess beteiligt?
Antwort:
Nein.

Wenn nein: Weiter bei 6.10)

6.3) Beschreiben Sie den Deployment-Prozess.
Antwort:
—

6.4) Welche Infrastruktur wird genutzt (Tools, Server, etc.)?
Antwort:
—

6.5) Gibt es klar verteilte Aufgaben und Verantwortlichkeiten für das Deployment?
Antwort:
—

6.5.1) Wer ist zuständig für den Deployment-Prozess? Beschreiben Sie die

Verteilung der Aufgaben und die Aufgaben selbst.

Antwort:

—

6.5.2) Wer ist zuständig für die während des Deployment-Prozesses genutzte Infrastruktur?

Antwort:

—

6.6) Fehlerfälle während des Deployment-Prozesses:

6.6.1) Gibt es in einem Fehlerfall während des Deployment-Prozesses ein festgelegtes Vorgehen?

Antwort:

—

6.6.1.1) Wenn ja: Beschreiben Sie dieses!

Antwort:

—

6.6.2) Wie wird, unabhängig von möglichen Vorgehensvorschriften, tatsächlich reagiert, welche Maßnahmen werden ergriffen?

Antwort:

—

6.7) Empfinden Sie den Deployment-Prozess als leichtgewichtig oder schwergewichtig?

Antwort:

—

6.7.1) Ist er leicht zu verstehen?

Antwort:

—

6.7.2) Ist er leicht umzusetzen?

Antwort:

—

6.8) Ist der Deployment-Prozess, bzw. das Vorgehen bei diesem, ausreichend und verständlich dokumentiert und ist die Dokumentation leicht zugänglich?

Antwort:

—

6.9) Gibt es während des Deployment-Prozesses Feedback (Monitoring),

das klar verständlich ist, sodass Sie sich sicher genug fühlen in der Durchführung des Deployments?

Antwort:

—

6.9.1) Gibt es genug Feedback, wenn Probleme auftreten?

Antwort:

—

6.9.2) Gibt es genug Feedback, wenn keine Probleme auftreten?

Antwort:

—

6.10) Gibt es generell Probleme, die Sie im Deployment-Prozess wahrnehmen / erfahren?

Antwort:

Die Downtime während des Deployments ist aus Benutzersicht störend. Die fehlende Automatisierung ist aus meiner Sicht kein Problem, sondern für delegs als Ausbildungsprojekt eine gute Gelegenheit zu lernen, was alles im Rahmen eines Deployments zu tun ist und wie wichtig dabei Sorgfalt ist und wie hilfreich Automatisierung von Teilen des Prozesses ist.

6.10.1) Wenn ja, beschreiben Sie diese.

Antwort:

—

6.11) Sehen Sie Behinderungen, die den Fluss des Deployment-Prozesses aufhalten?

Antwort:

Die fehlende Automatisierung - wie gesagt, aber so gewünscht.

6.11.1) Wenn ja, welche?

Antwort:

—

7.) Gibt es Wünsche oder Verbesserungsvorschläge für Ihr Projekt,

7.1) in Bezug auf den Deployment-Prozess?

Antwort:

Ein automatisierter Build Server wäre gut. Er sollte nach dem Einchecken von Code ermöglichen, auf Knopfdruck ein Test Deployment anzustoßen.

7.2) in Bezug auf den Informationsfluss, hinsichtlich der Rückmeldungen durch den Deployment-Prozess (sowohl an die direkt beteiligten Kräfte, als auch an die nicht direkt beteiligten Kräfte)?

Antwort:

Es wäre gut, nach jedem Produktiv Deployment eine Mail an alle Stakeholder zu schicken. In der Mail sollten neuen Features (Story-Namen und Ids) beschrieben sein. Die Sprache der Mail sollte nicht zu technisch formuliert sein, sodass auch Interessenten außerhalb des Tech-Teams diese verstehen können. Sie könnte automatisiert erstellt werden, sollte aber manuell überarbeitet und abgeschickt werden.

7.3) in Bezug auf das Testen?

Antwort:

Eine Übersicht über die Testabdeckung wäre gut.

7.4) in Bezug auf die Verteilung von Aufgaben und Verantwortlichkeiten?

Antwort:

Alle im Team sollen wissen, wie das Deployment funktioniert und es durchführen können. Das Wissen soll auf alle verbreitet sein.

8.) Haben Sie noch weitere Anmerkungen?

Antwort:

Es könnte deutlich mehr automatisiert werden, andererseits handelt es sich bei delegs um ein Ausbildungsprojekt, in dem es gut ist, wenn die Teammitglieder die einzelnen Schritte sehen und verstehen können und sie erkennen, welche Disziplin nötig ist, damit das Deployment reibungslos durchgeführt werden kann.

Die Projektstruktur und das Deployment sollten so einfach wie möglich gehalten werden. Maven scheint zu komplex zu sein für delegs. Es erscheint nicht wirklich sinnvoll, delegs in Vertikalen zu unterteilen. Es macht mehr Sinn delegs monolithisch zu lassen. Maven ist einfach zu aufwendig, um es auf ein monolithisches System anzuwenden.

Deployment ermöglichen ohne Ausfallzeiten zu haben und nicht eine Mail schicken zu müssen"Speichert alle eure Arbeit, wir stellen in 15 Minuten eine neue Produktivversion bereit".

Interview mit Adrian Metzner

Interviewte Person: Adrian Metzner

Kontext: delegs-Projekt / DevOps

Datum des Interviews: 14.12.2016

Da der Begriff "Deployment-Prozess" im IT-Bereich mit unterschiedlicher Bedeutung belegt ist, wurde dem Befragten zu Beginn des Interviews eine Definition zu dem Begriff gegeben. So wurde zwischen Interviewer und Befragten eine gemeinsame Basis geschaffen, auf der das Interview stattfinden konnten. Der Begriff "Deployment-Prozess" wurde wie folgt beschrieben:

Der Deployment-Prozess ist Teil des Softwareentwicklungsprozesses und setzt genau an der Stelle ein, an der Code durch Softwareentwickler, wie z.B. in Form eines neuen Features, fertiggestellt wurde. Das Deployment ist das Bereitstellen dieses Codes in Form einer neuen Version in Test- oder Produktivumgebungen, z.B. auf einem virtuellen oder physischen Hardware-Server. Unter den Begriff *Deployment-Prozess* kann auch schon das Integrieren von Code in die Codebasis fallen und auch Produktivbetrieb von Software und deren Wartung.

Folgende Fragen wurden während des Interviews gestellt:

1.) Was ist Ihre Aufgabe im Projekt?

Antwort:

Ich bin Softwareentwickler im Projekt.

2.) Gibt es mehrere Teams, die an der Software arbeiten?

Antwort:

Nein.

2.1) Wenn ja: Wie viele Entwickler haben die jeweiligen Teams?

Antwort:

—

3.) Wie viele Entwickler arbeiten in Ihrem Projektteam?

Antwort:

Fünf.

4.) Administratorische Aufgaben im Projekt:

4.1) Gibt es Administratoren / Operators, die für Ihr Projekt zuständig sind oder die Teil Ihres Projektteams sind und im Team mitarbeiten?

Antwort:

Alle Entwickler übernehmen Operations- und Admistrationsaufgaben.

4.1.1) Wenn ja, wie viele und welche Aufgaben haben diese?

Antwort:

Es wird dafür gesorgt dass die Software auf dem Server läuft. Logs und Access-Rights werden überwacht. Es werden keine Serverupdates gemacht. Es wird alles das gemacht, was wichtig ist, damit der Server sicher laufen kann.

4.2) Bei mehreren Teams:

Gibt es Administratoren / Operators, die für andere Teams ihres Projekts zuständig sind oder die als Teammitglied in den anderen Teams mitarbeiten?

Antwort:

—

4.2.1) Wenn ja, wie viele und welche Aufgaben haben diese?

Antwort:

—

4.3) Werden administratorische Aufgaben / Operations-Aufgaben von Entwicklern übernommen?

Antwort:

Ja.

4.3.1) Wenn ja, welche?

Antwort:

Die oben genannten.

5.) Wie oft, also in welchem Rhythmus oder nach welchen Zeiträumen, ist in Ihrem Projekt ein Produktiv-Deployment von neuen Features, Versionen, oder Ähnlichem vorgesehen?

Antwort:

Eigentlich alle zwei Wochen. Es wird versucht, das Deployment nach diesem Rhythmus durchzuführen. Es kann sein, dass Features nicht rechtzeitig fertig werden und sich dadurch das Deployment verzögert.

5.1) Wenn Rhythmus vorgesehen: Wird der vorgesehene Rhythmus eingehalten?

Antwort:

Siehe oben.

5.1.1) Wenn nein: Was sind die Gründe, die ein Deployment verzögern?

Antwort:

Der Aufwand für ein Deployment ist so groß, dass sich dessen Durchführung für kleine Änderungen nicht lohnt. Außerdem ist die Downtime der Anwendung zu groß, wenn man ein Deployment durchführt, auch darum lohnt sich ein Deployment bei nur kleinen Änderungen an der Anwendung nicht.

5.2) Wenn nein: Sollte es Ihrer Meinung nach einen festen Deployment-Rhythmus geben?

Antwort:

—

5.2.1) Wenn ja, warum?
Antwort:
—

6.) Der Deployment-Prozess (bei mehreren Prozessen die folgenden Fragen je Prozess durchgehen):

6.1) Gibt es einen oder mehrere Deployment-Prozesse (z.B. Test- & Produktiv Deployment)?
Antwort:
Es gibt zwei Deployment-Prozesse: Test Deployment in Testumgebung und Produktiv Deployment in Produktivumgebung.

6.2) Sind Sie am Deployment-Prozess beteiligt?
Antwort:
Ja.

Wenn nein: Weiter bei 6.10)

6.3) Beschreiben Sie den Deployment-Prozess.
Antwort:
Für das Deployment gibt es folgenden Ablauf:
Der Stand auf der Master Branch wird getestet durch Ausführen von Unit-Tests und manuellen GUI-Tests.
Es wird ein Tag für die neue Version erstellt.
Handelt es sich um ein Produktiv Deployment und nicht um ein Test Deployment wird 15 Minuten vor Abschaltung der alten Anwendungsversion eine Mail an alle Stakeholder und Hauptnutzer geschickt, die auf einen Ausfall der Anwendung wegen des Deployment hinweist.
Ein Build der neuen Version wird erstellt. Der Zeitaufwand hierbei beträgt 10-20 Minuten.
Die alte Version wird abgeschaltet.
Bei Produktiv Deployment wird der aktuelle Stand der Datenbank gesichert (Dump) und die alte Anwendungsversion wird gesichert.
Wenn nötig, wird eine Datenbankmigration durchgeführt.
Die neue Version wird auf den Server geladen und gestartet.
Die manuellen GUI-Tests werden auf der neuen Version, die jetzt live ist, getestet.
Es wird eine E-Mail an alle Hauptnutzer und Stakeholder gesendet, die über die Veränderungen an der Anwendung informiert.
Es wird ein Wiki Eintrag für die Nutzer und Stakeholder erstellt, die über die Verände-

rungen an der Anwendung informiert.
Als letztes wird auch noch ein Beitrag auf der Website für alle Nutzer geschrieben.
6.4) Welche Infrastruktur wird genutzt (Tools, Server, etc.)?
Antwort:
Eclipse mit GWT-Plugin,
JUnit,
GWT JUnit,
Virtueller Server von ServerAnbieterA,
Cyberduck,
MySQL Workbench,
Tomcat und Apache als Webserver
6.5) Gibt es klar verteilte Aufgaben und Verantwortlichkeiten für das Deployment?
Antwort:
Nein. Die Personen aus dem Tech-Team, die das Deployment durchführen, haben die volle Verantwortung. Jedes Mitglied aus dem Tech-Team ist fähig, das Deployment durchzuführen.

6.5.1) Wer ist zuständig für den Deployment-Prozess? Beschreiben Sie die Verteilung der Aufgaben und die Aufgaben selbst.
Antwort:
—

6.5.2) Wer ist zuständig für die während des Deployment-Prozesses genutzte Infrastruktur?
Antwort:
Auch die entsprechenden Mitarbeiter aus dem Tech-Team.

6.6) Fehlerfälle während des Deployment-Prozesses:

6.6.1) Gibt es in einem Fehlerfall während des Deployment-Prozesses ein festgelegtes Vorgehen?
Antwort:
Ja. Das Vorgehen unterscheidet sich zwischen Produktiv- und Test Deployment.

6.6.1.1) Wenn ja: Beschreiben Sie dieses!
Antwort:
Produktiv Deployment:
Wenn ein schwerer Bug aufgetreten ist oder das Deployment fehlschlägt, wird die alte Version wieder online gestellt. Gab

es eine Datenbankmigration wird diese wieder zurückgerollt, indem der zur Sicherheit erstellte Dump wieder eingespielt wird. Danach werden die Fehler behoben und es wird weiter fortgefahren mit einem Test Deployment, auf das dann ein Produktiv Deployment folgt.

Test Deployment:
Wenn ein Deployment nicht funktioniert, muss der Grund dafür gefunden und behoben werden. Wenn der Bug gefunden wird, gibt es kein Rollback, das gemacht werden muss, sondern die Version mit dem behobenen Fehler wird online gestellt als Testversion und die Testperiode erweitert sich um eine Woche in der weiter getestet wird.

In beiden Fällen, bei Auffinden eines Bugs, entscheidet das Lehrteam wie schlimm dieser ist. Wird der Bug als harmlos eingestuft, wird trotz des Bugs ein Deployment durchgeführt. Wenn aber der Bug schwerer ist, muss mit dem nächsten dieser vor dem nächsten Deployment gefixt werden.

6.6.2) Wie wird, unabhängig von möglichen Vorgehensvorschriften, tatsächlich reagiert, welche Maßnahmen werden ergriffen?

Antwort:

—

6.7) Empfinden Sie den Deployment-Prozess als leichtgewichtig oder schwergewichtig?

Antwort:

Er ist relativ schwergewichtig.

6.7.1) Ist er leicht zu verstehen?

Antwort:

Ja.

6.7.2) Ist er leicht umzusetzen?

Antwort:

Vom intellektuellen Aufwand, also den nötige Fähigkeiten, ja, allerdings ist er schon sehr zeitaufwendig.

6.8) Ist der Deployment-Prozess, bzw. das Vorgehen bei diesem, ausreichend und

verständlich dokumentiert und ist die Dokumentation leicht zugänglich?

Antwort:

Ja.

6.9) Gibt es während des Deployment-Prozesses Feedback (Monitoring), das klar verständlich ist, sodass Sie sich sicher genug fühlen in der Durchführung des Deployments?

Antwort:

Ja. Wenn Tests fehlschlagen, ist das zu merken und wenn das Deployment fehlschlägt auch. Jede Phase gibt entsprechend Feedback.

6.9.1) Gibt es genug Feedback, wenn Probleme auftreten?

Antwort:

Größtenteils. Bis zu dem Moment, an dem ein Build der Anwendung erstellt wurde, gibt es genug Feedback bei möglichen Problemen. Wenn der Build an den Apache TomCat-Server übergeben wird und dort scheitert, kann das Feedback sehr kryptisch und umständlich zu verstehen sein.

6.9.2) Gibt es genug Feedback, wenn keine Probleme auftreten?

Antwort:

Ja, wie schon erwähnt.

6.10) Gibt es generell Probleme, die Sie im Deployment-Prozess wahrnehmen / erfahren?

Antwort:

Ja.

6.10.1) Wenn ja, beschreiben Sie diese.

Antwort:

Hoher zeitlich und manueller Aufwand.

6.11) Sehen Sie Behinderungen, die den Fluss des Deployment-Prozesses aufhalten?

Antwort:

Ja.

6.11.1) Wenn ja, welche?

Antwort:

Es dauert lange, einen Build zu erzeugen, außerdem haben wird lange Testzeiten und zu lange Ausfallzeiten beim Deployment.

7.) Gibt es Wünsche oder Verbesserungsvorschläge für Ihr Projekt,

7.1) in Bezug auf den Deployment-Prozess?

Antwort:

Ein höherer Grad an Automatisierung.
Unabhängigkeit von Entwicklerhardware:
Eine einheitliche Hardware- und Softwarekonfiguration, die für das Bauen zuständig ist, sodass Eigenheiten von Entwicklerrechnern nicht mehr zu beachten sind.
Dadurch sind Fehler leichter zu verstehen, da immer die gleiche Soft- und Hardware genutzt wird (z.B. Java Version und OS, etc.)

7.2) in Bezug auf den Informationsfluss, hinsichtlich der Rückmeldungen durch den Deployment-Prozess (sowohl an die direkt beteiligten Kräfte, als auch an die nicht direkt beteiligten Kräfte)?

Antwort:
Da alles automatisiert läuft, werden auch Informationen automatisch generiert und aggregiert. Dadurch werden direkt klar verständliche Informationen über den Status des Deployment-Prozesses erzeugt.

7.3) in Bezug auf das Testen?

Antwort:
Continuous Integration:
Bei Einchecken auf den Master (oder Stable Branch) automatisches Bauen und Testen der Anwendung. Danach eine automatisierte Nachricht an die Entwickler. Bei fehlerhaften Test Commit abweisen. Dadurch ist die Wahrscheinlichkeit, dass der Master Stabil ist, sehr viel höher. Im optimalen Fall wird ein Branch des Masters erzeugt, auf dem der ?kaputte Commit? abgelegt ist.

7.4) in Bezug auf die Verteilung von Aufgaben und Verantwortlichkeiten?

Antwort:
Nein. Alle sollen den Deployment-Prozess ausführen und betreuen können.

8.) Haben Sie noch weitere Anmerkungen?

Antwort:

—

Interview mit Cenan Akcicek

Interviewte Person: Cenan Akcicek
Kontext: delegs-Projekt / DevOps
Datum des Interviews: 14.12.2016
Da der Begriff "Deployment-Prozess" im IT-Bereich mit unterschiedlicher Bedeutung belegt ist, wurde dem Befragten zu Beginn des Interviews eine Definition zu dem Begriff gegeben.

So wurde zwischen Interviewer und Befragten eine gemeinsame Basis geschaffen, auf der das Interview stattfinden konnten. Der Begriff "Deployment-Prozess" wurde wie folgt beschrieben:

> Der Deployment-Prozess ist Teil des Softwareentwicklungsprozesses und setzt genau an der Stelle ein, an der Code durch Softwareentwickler, wie z.B. in Form eines neuen Features, fertiggestellt wurde. Das Deployment ist das Bereitstellen dieses Codes, in Form einer neuen Version, in Test- oder Produktivumgebungen, z.B. auf einem virtuellen oder physischen Hardware-Server. Unter den Begriff *Deployment-Prozess* kann auch schon das Integrieren von Code in die Codebasis fallen und auch Produktivbetrieb von Software und deren Wartung.

Folgende Fragen wurden während des Interviews gestellt:

1.) Was ist Ihre Aufgabe im Projekt?

Antwort:

Softwareentwickler.

2.) Gibt es mehrere Teams, die an der Software arbeiten?

Antwort:

Nein, es gibt keine weiteren Entwicklerteams.

2.1) Wenn ja: Wie viele Entwickler haben die jeweiligen Teams?

Antwort:

—

3.) Wie viele Entwickler arbeiten in Ihrem Projektteam?

Antwort:

Fünf. Und ein Projektleiter, der an der Entwicklung nicht teilnimmt.

4.) Administratorische Aufgaben im Projekt:

4.1) Gibt es Administratoren / Operators, die für Ihr Projekt zuständig sind oder die Teil Ihres Projektteams sind und im Team mitarbeiten?

Antwort:

Es gibt einen Mitarbeiter des Teams, der, aufgrund seines Wissens und geschuldet der Tatsache, dass er schon sehr lange im Projekt ist, im Team die Stellung eines Operators hat. Es gibt noch einen projektexternen Administrator.

4.1.1) Wenn ja, wie viele und welche Aufgaben haben diese?

Antwort:

Aufgaben des internen Mitarbeiters ist Serveradministration. Aufgaben der externen Administratoren ist die Wartung des Servers, der von einem Drittanbieter gehostet wird.

4.2) Bei mehreren Teams:
Gibt es Administratoren / Operators, die für andere Teams ihres Projekts zuständig sind oder die als Teammitglied in den anderen Teams mitarbeiten?
Antwort:
—

4.2.1) Wenn ja, wie viele und welche Aufgaben haben diese?
Antwort:
—

4.3) Werden administratorische Aufgaben / Operations-Aufgaben von Entwicklern übernommen?
Antwort:
Ja.

4.3.1) Wenn ja, welche?
Antwort:
Die eben genannten. Außerdem prüft ein Entwickler jeden Montag die Log Dateien des Servers auf sicherheitsrelevante Einträge.

5.) Wie oft, also in welchem Rhythmus oder nach welchen Zeiträumen, ist in Ihrem Projekt ein Produktiv-Deployment von neuen Features, Versionen, oder Ähnlichem vorgesehen?
Antwort:
Es gibt kein festes zeitliches Intervall.

5.1) Wenn Rhythmus vorgesehen: Wird der vorgesehene Rhythmus eingehalten?
Antwort:
—

5.1.1) Wenn nein: Was sind die Gründe, die ein Deployment verzögern?
Antwort:
—

5.2) Wenn nein: Sollte es Ihrer Meinung nach einen festen Deployment-Rhythmus geben?
Antwort:
Ja.

5.2.1) Wenn ja, warum?
Antwort:
Da man dadurch das Vorgehen besser planen kann und einen Feedback-

Mechanismus bekommt, der einem zeigt was erreicht wurde und was nicht, ähnlich wie auch bei den Iterationen im Scrum-Prozess.

6.) Der Deployment-Prozess (bei mehreren Prozessen die folgenden Fragen je Prozess durchgehen):

6.1) Gibt es einen oder mehrere Deployment-Prozesse (z.B. Test- & Produktiv Deployment)?

Antwort:

Es gibt einen Deployment-Prozess.

6.2) Sind Sie am Deployment-Prozess beteiligt?

Antwort:

Ja.

Wenn nein: Weiter bei 6.10)

6.3) Beschreiben Sie den Deployment-Prozess.

Antwort:

Am Anfang steht das Beitragen von Code in die Codebasis im Versionsverwaltungssystem. Deployments werden nicht nach festen zeitlichen Intervallen durchgeführt, sondern wenn im Team darüber entschieden wurde.

Es gibt eine Sammlung von Aufgaben (Bug-Fixes, Features, kleine Aufgaben), die vor dem nächsten Release umgesetzt werden müssen. Diese Liste unterliegt stetigen Änderungen, manchmal kommen Aufgaben hinzu, manche werden gestrichen. Wenn alle Aufgaben der Liste erfüllt wurden, wird entschieden, ob eine neue Version bereitgestellt werden soll. Zuerst wird ein Test Deployment angestoßen, das dazu dient, dass die Stakeholder die neue Version testen können.

Das Test Deployment wird von einem Teil der Entwickler durchgeführt und geschieht wie folgt: Es werden alle Unit- und Integrationstests ausgeführt. Die getestete Anwendung wird auf dem Server hochgeladen. Nachdem das Test Deployment eine Woche gelaufen ist, ohne dass die Stakeholder Fehler gefunden haben und die Stakeholder keine Änderungswünsche haben, wird die Testversion Produktiv genommen.

Das Deployment der Produktivversion wird entsprechend dem Test Deployment durchgeführt. Unterschied ist, dass noch manuelle GUI Tests an der neuen Version nach deren Deployment durchgeführt werden. Wenn diese erfolgreich waren, werden die Stakeholder und Anwender über die neue Version per Mail informiert.

6.4) Welche Infrastruktur wird genutzt (Tools, Server, etc.)?

Antwort:

Eclipse

Git (VMS) mit SourceTree
Virtueller Linux Debian Server für Produktiv- und Testsystem, gehostet von ServerAnbieterA
Migrationsskripte in SQL
KDiff
Datenbank: MySQL ? lokal auf Entwicklerrechnern und auf dem gleichen virtuellen Server wie Produktiv- und Testversion der Anwendung

6.5) Gibt es klar verteilte Aufgaben und Verantwortlichkeiten für das Deployment?

Antwort:

Nein.

6.5.1) Wer ist zuständig für den Deployment-Prozess? Beschreiben Sie die Verteilung der Aufgaben und die Aufgaben selbst.

Antwort:

Zuständig sind jeweils die Mitglieder der Tech-Teams. Diese übernehmen alle Aufgaben, die im Deployment-Prozess anfallen. Aufgrund der nicht gleichbleibenden Besetzung des Teams über die Arbeitswoche, kann der Deployment-Prozesses auch von einem Teilteam angefangen und von einem anderen zu Ende geführt werden.

6.5.2) Wer ist zuständig für die während des Deployment-Prozesses genutzte Infrastruktur?

Antwort:

Alle, also niemand speziell.

6.6) Fehlerfälle während des Deployment-Prozesses:

6.6.1) Gibt es in einem Fehlerfall während des Deployment-Prozesses ein festgelegtes Vorgehen?

Antwort:

Nein. Außer den Fehler zu beheben und den Deployment-Prozess neu anzufangen.

6.6.1.1) Wenn ja: Beschreiben Sie dieses!

Antwort:

—

6.6.2) Wie wird, unabhängig von möglichen Vorgehensvorschriften, tatsächlich reagiert, welche Maßnahmen werden ergriffen?

Antwort:

Fehler finden und Deployment-Prozess neu starten.

6.7) Empfinden Sie den Deployment-Prozess als leichtgewichtig oder schwergewichtig?

Antwort:

Er ist aufwendig, vor allem, weil er nicht regelmäßig durchgeführt wird und die Routine fehlt.

6.7.1) Ist er leicht zu verstehen?

Antwort:

Ja.

6.7.2) Ist er leicht umzusetzen?

Antwort:

Es ist mühsam, aber nicht schwer zu verstehen. Der Prozess ist dokumentiert. Nur Fehlerfälle machen wirklich Probleme.

6.8) Ist der Deployment-Prozess, bzw. das Vorgehen bei diesem, ausreichend und verständlich dokumentiert und ist die Dokumentation leicht zugänglich?

Antwort:

Ja. Er ist ausreichend dokumentiert und zugänglich, da die Dokumentation im Repository hinterlegt ist.

6.9) Gibt es während des Deployment-Prozesses Feedback (Monitoring), das klar verständlich ist, sodass Sie sich sicher genug fühlen in der Durchführung des Deployments?

Antwort:

Nein. Es gibt nur die GUI-Tests, die nach dem Deployment manuell durchgeführt werden und einem ein sicheres Gefühl geben.

6.9.1) Gibt es genug Feedback, wenn Probleme auftreten?

Antwort:

—

6.9.2) Gibt es genug Feedback, wenn keine Probleme auftreten?

Antwort:

—

6.10) Gibt es generell Probleme, die Sie im Deployment-Prozess wahrnehmen / erfahren?

Antwort:

Nein. Es wird alles manuell durchgeführt, auch die Tests die sicherstellen, dass das Deployment erfolgreich war.

6.10.1) Wenn ja, beschreiben Sie diese.

Antwort:

—

6.11) Sehen Sie Behinderungen, die den Fluss des Deployment-Prozesses aufhalten?

Antwort:

Nein. Es wäre schön, wenn das Deployment automatisch ausgeführt würde.

6.11.1) Wenn ja, welche?

Antwort:

—

7.) Gibt es Wünsche oder Verbesserungsvorschläge für Ihr Projekt,

7.1) in Bezug auf den Deployment-Prozess?

Antwort:

Automatisierung.

7.2) in Bezug auf den Informationsfluss, hinsichtlich der Rückmeldungen durch den Deployment-Prozess (sowohl an die direkt beteiligten Kräfte, als auch an die nicht direkt beteiligten Kräfte)?

Antwort:

Nein.

7.3) in Bezug auf das Testen?

Antwort:

Automatisieren. Automatisierte GUI-Tests, wie es z.B mit Selenium möglich ist.

7.4) in Bezug auf die Verteilung von Aufgaben und Verantwortlichkeiten?

Antwort:

Nein.

8.) Haben Sie noch weitere Anmerkungen?

Antwort:

—

Interview mit Lisa Ahlers

Interviewte Person: Lisa Ahlers

Kontext: delegs-Projekt / DevOps

Datum des Interviews: 02.01.2017

Da der Begriff "Deployment-Prozess" im IT-Bereich mit unterschiedlicher Bedeutung belegt ist, wurde dem Befragten zu Beginn des Interviews eine Definition zu dem Begriff gegeben. So wurde zwischen Interviewer und Befragten eine gemeinsame Basis geschaffen, auf der das Interview stattfinden konnten. Der Begriff "Deployment-Prozess" wurde wie folgt beschrieben:

> Der Deployment-Prozess ist Teil des Softwareentwicklungsprozesses und setzt genau an der Stelle ein, an der Code durch Softwareentwickler, wie z.B. in Form eines neuen Features, fertiggestellt wurde. Das Deployment ist das Bereitstellen dieses Codes, in Form einer neuen Version, in Test- oder Produktivumgebungen, z.B. auf einem virtuellen oder physischen Hardware-Server. Unter den Begriff *Deployment-Prozess* kann auch schon das Integrieren von Code in die Codebasis fallen und auch Produktivbetrieb von Software und deren Wartung.

Folgende Fragen wurden während des Interviews gestellt:

1.) Was ist Ihre Aufgabe im Projekt?

Antwort:

Softwareentwicklung.

2.) Gibt es mehrere Teams, die an der Software arbeiten?

Antwort:

Nein, es gibt nur ein Team.

2.1) Wenn ja: Wie viele Entwickler haben die jeweiligen Teams?

Antwort:

—

3.) Wie viele Entwickler arbeiten in Ihrem Projektteam?

Antwort:

Momentan fünf Entwickler.

4.) Administratorische Aufgaben im Projekt:

4.1) Gibt es Administratoren / Operators, die für Ihr Projekt zuständig sind oder die Teil Ihres Projektteams sind und im Team mitarbeiten?

Antwort:

—

4.1.1) Wenn ja, wie viele und welche Aufgaben haben diese?

Antwort:

—

4.2) Bei mehreren Teams:

Gibt es Administratoren / Operators, die für andere Teams ihres Projekts zuständig sind oder die als Teammitglied in den anderen Teams mitarbeiten?
Antwort:
—

4.2.1) Wenn ja, wie viele und welche Aufgaben haben diese?
Antwort:
—

4.3) Werden administratorische Aufgaben / Operations-Aufgaben von Entwicklern übernommen?
Antwort:
—

4.3.1) Wenn ja, welche?
Antwort:
—

5.) Wie oft, also in welchem Rhythmus oder nach welchen Zeiträumen, ist in Ihrem Projekt ein Produktiv-Deployment von neuen Features, Versionen, oder Ähnlichem vorgesehen?
Antwort:
Soweit ich weiß, gibt es keinen festen zeitlichen Rhythmus. Sobald alle Stories einer Iteration abgeschlossen sind und neue Bugs nicht als kritisch eingestuft werden, wird ein Deployment vorgenommen.

5.1) Wenn Rhythmus vorgesehen: Wird der vorgesehene Rhythmus eingehalten?
Antwort:
—

5.1.1) Wenn nein: Was sind die Gründe, die ein Deployment verzögern?
Antwort:
—

5.2) Wenn nein: Sollte es Ihrer Meinung nach einen festen Deployment-Rhythmus geben?
Antwort:
Da wir keinen großen Zeitdruck von außen haben, denke ich, dass es so okay ist, wobei ein Rhythmus natürlich dafür sorgen würde, dass die Anwender schneller von behobenen Fehlern und neuen Features in der Produktivversion profitieren könnten.

5.2.1) Wenn ja, warum?
Antwort:
—

6.) Der Deployment-Prozess (bei mehreren Prozessen die folgenden Fragen je Prozess durchgehen):

6.1) Gibt es einen oder mehrere Deployment-Prozesse (z.B. Test- & Produktiv Deployment)?
Antwort:
Es gibt Test Deployment und Produktiv Deployment.

6.2) Sind Sie am Deployment-Prozess beteiligt?
Antwort:
Ja.

Wenn nein: Weiter bei 6.10)

6.3) Beschreiben Sie den Deployment-Prozess.
Antwort:
Test Deployment:
*Zuerst muss das Projekt auf den aktuellen Stand gebracht werden, dann werden alle Tests gestartet. Anschließend muss noch eine Liste von manuellen GUI-Tests ausgeführt werden. Als nächstes wird die Datenbankversion aktualisiert und in ein Migrationsskript eingefügt. Dieses wird im Projekt Repository abgelegt und anschließend wird ein Tag für die neue Version im Projekt Repository angelegt. Als nächstes wird das Projekt mit dem ein GWT Compiler gebaut und anschließend der Inhalt des war Ordners als *.war Datei gezippt. Nun wird auf den ServerAnbieterA-Server zugegriffen und die alte Testanwendung deaktiviert. Es wird im Browser überprüft, ob die alte Testversion auch wirklich nicht mehr verfügbar ist. Anschließend wird die Datenbank angepasst. Dafür werde die Daten der Produktivdatenbank in die Testdatenbank übernommen und anschließend das aktuelle Migrationsskript darauf ausgeführt. Danach wird die neue Testanwendung auf den Server kopiert. Am Ende werden manuell ein paar Performance-Tests und erneut die GUI-Tests ausgeführt. Anschließend werden alle Kollegen über das Test Deployment informiert.*

Produktiv Deployment:
Bevor ein Produktiv Deployment erfolgt, muss die Testanwendung bereitgestellt worden sein und von den Nutzern getestet und abgenommen worden sein. Einige Stunden bevor das Deployment vorgenommen werden soll, wird eine Mail an alle Nutzer geschickt, die

*darüber informiert, dass die Produktivversion ausgetauscht werden soll und zu welchem Zeitpunkt dies geschehen soll. Etwa 15 min vor dem Deployment wird erneut eine solche Mail geschickt. Das Projekt wird dann im Repository auf den Tag der neuen Version gesetzt und mit dem GWT Compiler gebaut. Der war Ordner wird als *.war Datei gezippt. Anschließend wird die auf dem Server gegenwärtig laufende Version der Anwendung außer Betrieb gesetzt (durch umbenennen oder löschen) und es wird geprüft, ob dies erfolgreich war. Die Anwendung ist dann nicht mehr erreichbar. Anschließend wird das Migrationsskript auf der Produktivversion der Datenbank ausgeführt. Nun kann die neue Version der Anwendung auf den Server hochgeladen werden. Abschließend werden alle manuellen GUI-Tests durchgeführt und danach eine Mail an alle Kollegen geschickt, welche eine Liste der neuen Features enthält und die URL der Software.*

6.4) Welche Infrastruktur wird genutzt (Tools, Server, etc.)?

Antwort:

Zum Testen wird Firefox mit GWT Plugin benutzt, nach dem Deployment dann ein beliebiger Browser. Für den Zugriff auf den Server wird Cyberduck (oder ein ähnliches Programm) benutzt. Um auf die Datenbank zuzugreifen, wird die MySQL Workbench eingesetzt.

6.5) Gibt es klar verteilte Aufgaben und Verantwortlichkeiten für das Deployment?

Antwort:

Nein. Das Deployment kann von jedem Entwickler durchgeführt werden.

6.5.1) Wer ist zuständig für den Deployment-Prozess? Beschreiben Sie die Verteilung der Aufgaben und die Aufgaben selbst.

Antwort:

—

6.5.2) Wer ist zuständig für die während des Deployment-Prozesses genutzte Infrastruktur?

Antwort:

—

6.6) Fehlerfälle während des Deployment-Prozesses:

6.6.1) Gibt es in einem Fehlerfall während des Deployment-Prozesses ein festgelegtes Vorgehen?

Antwort:

Nein, nicht das ich wüsste.

6.6.1.1) Wenn ja: Beschreiben Sie dieses!

Antwort:

—

6.6.2) Wie wird, unabhängig von möglichen Vorgehensvorschriften, tatsächlich reagiert, welche Maßnahmen werden ergriffen?

Antwort:

Es wird versucht, die Fehlerquelle zu finden und diesen zu beheben. Es kann passieren, dass der ganze Deployment-Prozess von Neuem angefangen werden muss.

6.7) Empfinden Sie den Deployment-Prozess als leichtgewichtig oder schwergewichtig?

Antwort:

Eher schwergewichtig, da er durch viele manuelle Tests doch etwas länger dauern kann.

6.7.1) Ist er leicht zu verstehen?

Antwort:

Durch ausführliche Protokolle, die während der Prozesse ausgefüllt werden müssen, ist es leichter sich in den Prozess hineinzufinden, wobei es aber gerade beim ersten Ausführen besser ist, wenn jemand mit Erfahrung mitmacht, da doch einige Schritte unklar sein können.

6.7.2) Ist er leicht umzusetzen?

Antwort:

Wenn man weiß, was zu tun ist, ist der Deployment-Prozess nicht schwer umzusetzen.

6.8) Ist der Deployment-Prozess, bzw. das Vorgehen bei diesem, ausreichend und verständlich dokumentiert und ist die Dokumentation leicht zugänglich?

Antwort:

Ja, es gibt Protokolle für beide Deployment-Prozesse, die im Projekt-Repository zu finden sind. Diese werden immer weiter angepasst und erweitert, um sie so verständlich wie möglich zu machen. Macht beispielsweise jemand ein Deployment zum ersten Mal und findet einen Schritt im Protokoll nicht verständlich, wird das Protokoll angepasst, um diesen verständlicher zu machen.

6.9) Gibt es während des Deployment-Prozesses Feedback (Monitoring), das klar verständlich ist, sodass Sie sich sicher genug fühlen in der Durchführung des Deployments?

Antwort:

Nicht wirklich.

6.9.1) Gibt es genug Feedback, wenn Probleme auftreten?

Antwort:

Nein, wird das Projekt beispielsweise mit einer falschen Java-Compiler-Version gebaut, kann kein erfolgreiches Deployment der Anwendung vorgenommen werden. Um einen solchen Fehler zu finden, muss man aber wissen, welche Log-Dateien man sich dafür ansehen muss, um dann den Fehler zu finden. Es wird nicht direkt auf den Fehler hingewiesen.

6.9.2) Gibt es genug Feedback, wenn keine Probleme auftreten?

Antwort:

Im Fall ohne Probleme finde ich es nicht so schlimm, dass man keine extra Meldungen hat.

6.10) Gibt es generell Probleme, die Sie im Deployment-Prozess wahrnehmen / erfahren?

Antwort:

Ja.

6.10.1) Wenn ja, beschreiben Sie diese.

Antwort:

Das Problem, das beim Einsetzen der falschen Java-Compiler-Version entsteht, ist schon mehrfach aufgetreten, da an unterschiedlichen Rechnern Deployments vorgenommen wurden und vorher nicht überprüft wurde, ob die Compiler Version stimmt.

6.11) Sehen Sie Behinderungen, die den Fluss des Deployment-Prozesses aufhalten?

Antwort:

Ja.

6.11.1) Wenn ja, welche?

Antwort:

Durch das Durchführen der Tests, das Exportieren und Importieren von Daten aus der Datenbank und das Kompilieren entstehen im Deployment-Prozess immer wieder Wartezeiten, die den Fluss aufhalten.

7.) Gibt es Wünsche oder Verbesserungsvorschläge für Ihr Projekt,

7.1) in Bezug auf den Deployment-Prozess?

Antwort:

Weniger manuelle Schritte wären gut, da dadurch einige Fehler ausgeschlossen werden

könnten.

7.2) in Bezug auf den Informationsfluss, hinsichtlich der Rückmeldungen durch den Deployment-Prozess (sowohl an die direkt beteiligten Kräfte, als auch an die nicht direkt beteiligten Kräfte)?

Antwort:

Nein.

7.3) in Bezug auf das Testen?

Antwort:

Automatisierte GUI-Tests wären praktisch, da das manuelle Testen viel Zeit kostet.

7.4) in Bezug auf die Verteilung von Aufgaben und Verantwortlichkeiten?

Antwort:

Nein.

8.) Haben Sie noch weitere Anmerkungen?

Antwort:

—

Interview mit Vadim Holstein

Interviewte Person: Vadim Holstein
Kontext: delegs-Projekt / DevOps
Datum des Interviews: 07.01.2017

Da der Begriff "Deployment-Prozess" im IT-Bereich mit unterschiedlicher Bedeutung belegt ist, wurde dem Befragten zu Beginn des Interviews eine Definition zu dem Begriff gegeben. So wurde zwischen Interviewer und Befragten eine gemeinsame Basis geschaffen, auf der das Interview stattfinden konnten. Der Begriff "Deployment-Prozess" wurde wie folgt beschrieben:

> Der Deployment-Prozess ist Teil des Softwareentwicklungsprozesses und setzt genau an der Stelle ein, an der Code durch Softwareentwickler, wie z.B. in Form eines neuen Features, fertiggestellt wurde. Das Deployment ist das Bereitstellen dieses Codes, in Form einer neuen Version, in Test- oder Produktivumgebungen, z.B. auf einem virtuellen oder physischen Hardware-Server. Unter den Begriff *Deployment-Prozess* kann auch schon das Integrieren von Code in die Codebasis fallen und auch Produktivbetrieb von Software und deren Wartung.

Folgende Fragen wurden während des Interviews gestellt:

1.) Was ist Ihre Aufgabe im Projekt?

Antwort:

Softwareentwickler.

2.) Gibt es mehrere Teams, die an der Software arbeiten?

Antwort:

Nein, es gibt nur ein Team.

2.1) Wenn ja: Wie viele Entwickler haben die jeweiligen Teams?

Antwort:

—Nein, es gibt nur ein Team.

3.) Wie viele Entwickler arbeiten in Ihrem Projektteam?

Antwort:

Ein Festangestellter und fünf Werkstudenten.

4.) Administratorische Aufgaben im Projekt:

4.1) Gibt es Administratoren / Operators, die für Ihr Projekt zuständig sind oder die Teil Ihres Projektteams sind und im Team mitarbeiten?

Antwort:

Es gibt einen Projektleiter und das Entwicklungsteam, aber keine direkten Operators oder Administratoren.

4.1.1) Wenn ja, wie viele und welche Aufgaben haben diese?

Antwort:

—

4.2) Bei mehreren Teams:

Gibt es Administratoren / Operators, die für andere Teams ihres Projekts zuständig sind oder die als Teammitglied in den anderen Teams mitarbeiten?

Antwort:

—

4.2.1) Wenn ja, wie viele und welche Aufgaben haben diese?

Antwort:

—

4.3) Werden administratorische Aufgaben / Operations-Aufgaben von Entwicklern übernommen?

Antwort:

Ja.

4.3.1) Wenn ja, welche?

Antwort:

Prinzipiell alle, da es keine Administratoren oder Operators gibt.

5.) Wie oft, also in welchem Rhythmus oder nach welchen Zeiträumen, ist in Ihrem Projekt

ein Produktiv-Deployment von neuen Features, Versionen, oder Ähnlichem vorgesehen?

Antwort:

Das ist unterschiedlich. Es wird immer ein Deployment vorgenommen, wenn alle Stories in der gegenwärtigen Iteration abgeschlossen sind und keine schwerwiegenden Fehler in der Testversion entdeckt werden. Ursprünglich wurde geplant, jeden Montag zu prüfen, ob ein Deployment vorgenommen werden kann.

5.1) Wenn Rhythmus vorgesehen: Wird der vorgesehene Rhythmus eingehalten?

Antwort:

Nein, da es keinen festen Rhythmus gibt.

5.1.1) Wenn nein: Was sind die Gründe, die ein Deployment verzögern?

Antwort:

Ein Produktiv Deployment kann durch Fehler, die in der Testversion entdeckt werden verzögert werden. Da das Team nur aus einer Vollzeitkraft und mehreren Teilzeitkräften besteht, ist es schwierig, regelmäßig ein Deployment vorzunehmen.

5.2) Wenn nein: Sollte es Ihrer Meinung nach einen festen Deployment-Rhythmus geben?

Antwort:

Schwer zu sagen, da es nicht wirklich abschätzbar ist, wie lange eine Iteration zum Entwickeln braucht. Ich würde eher nein sagen. Die Ausnahme wäre, wenn man konsequent nach Scrum entwickeln würde. Dann würden feste Deployment-Rhythmen Sinn machen.

5.2.1) Wenn ja, warum?

Antwort:

—

6.) Der Deployment-Prozess (bei mehreren Prozessen die folgenden Fragen je Prozess durchgehen):

6.1) Gibt es einen oder mehrere Deployment-Prozesse (z.B. Test- & Produktiv Deployment)?

Antwort:

Ja, es gibt ein Test Deployment und ein Produktiv Deployment.

6.2) Sind Sie am Deployment-Prozess beteiligt?

Antwort:

Ja, ich bin oft an beiden Deployment-Prozessen beteiligt.

Wenn nein: Weiter bei 6.10)

6.3) Beschreiben Sie den Deployment-Prozess.

Antwort:

Test Deployment: Es wird eine Testversion auf unserem Server aufgesetzt. Die relevanten Tester werden benachrichtigt und es wird eine einwöchige Testphase gestartet.

Produktiv Deployment: Sollten nach der einwöchigen Testphase keine schwerwiegenden Fehler entdeckt worden sein, wird das Produktiv Deployment ausgeführt.

6.4) Welche Infrastruktur wird genutzt (Tools, Server, etc.)?

Antwort:

Zum Kompilieren wird der GWT Compiler verwendet. Die Anwendung wird auf einem Apache Tomcat 8 Server bereitgestellt.

6.5) Gibt es klar verteilte Aufgaben und Verantwortlichkeiten für das Deployment?

Antwort:

Nein, das Team, welches die Test- oder Produktivversion bereitstellt, ist gemeinsam verantwortlich für den gesamten Prozess.

6.5.1) Wer ist zuständig für den Deployment-Prozess? Beschreiben Sie die Verteilung der Aufgaben und die Aufgaben selbst.

Antwort:

Das Deployment Team, welches durch Entwickler des Tech-Teams besetzt ist.

6.5.2) Wer ist zuständig für die während des Deployment-Prozesses genutzte Infrastruktur?

Antwort:

Das Deployment Team.

6.6) Fehlerfälle während des Deployment-Prozesses:

6.6.1) Gibt es in einem Fehlerfall während des Deployment-Prozesses ein festgelegtes Vorgehen?

Antwort:

Es wurde kein Vorgehen festgelegt, aber es werden Log Dateien, die während des Deployment-Prozesses entstanden sind, daraufhin untersucht, was beim Deployment schiefgelaufen ist.

6.6.1.1) Wenn ja: Beschreiben Sie dieses!
Antwort:
—

6.6.2) Wie wird, unabhängig von möglichen Vorgehensvorschriften, tatsächlich reagiert, welche Maßnahmen werden ergriffen?
Antwort:
Es wird meist als erstes in die Catalina Log Dateien geschaut. Anschließend werden die Fehler behoben und der Deployment-Prozess erneut gestartet.

6.7) Empfinden Sie den Deployment-Prozess als leichtgewichtig oder schwergewichtig?
Antwort:
Schwer zu sagen. Der Deployment-Prozess ist nicht sehr kompliziert, sobald man ihn erlernt hat. Jedoch muss man aufpassen, keine Fehler zu machen, da sonst erneut der Deployment-Prozess wiederholt werden muss, was zeitaufwändig ist.

6.7.1) Ist er leicht zu verstehen?
Antwort:
Durch die vorhandene Anleitung ist der Deployment-Prozess mittlerweile verständlich, wenn auch etwas kompliziert, wenn man zum ersten Mal ein Deployment vornimmt.

6.7.2) Ist er leicht umzusetzen?
Antwort:
Der Prozess ist teilweise sehr zeitaufwändig. Er ist relativ leicht umzusetzen, sobald man eine Einweisung bekommen hat und die Anleitung zur Hilfe nimmt.

6.8) Ist der Deployment-Prozess, bzw. das Vorgehen bei diesem, ausreichend und verständlich dokumentiert und ist die Dokumentation leicht zugänglich?
Antwort:
Die Dokumentation ist als Textdatei im Repository abgespeichert und dementsprechend leicht zugänglich (wenn man weiß wo sie liegt). Die Dokumentation wurde mehrfach um Einträge ergänzt, da sie teilweise schwer verständlich war.

6.9) Gibt es während des Deployment-Prozesses Feedback (Monitoring), das klar verständlich ist, sodass Sie sich sicher genug fühlen in der Durchführung des Deployments?
Antwort:
Eher nein.

6.9.1) Gibt es genug Feedback, wenn Probleme auftreten?
Antwort:
Meiner Meinung nach nein. Anhand der Catalina Log-Dateien kann zwar gesehen werden, warum das Deployment gescheitert ist, dies ist jedoch oft schwer zu verstehen.
6.9.2) Gibt es genug Feedback, wenn keine Probleme auftreten?
Antwort:
Es gibt meines Wissens nach kein Feedback in diesem Fall.

6.10) Gibt es generell Probleme, die Sie im Deployment-Prozess wahrnehmen / erfahren?
Antwort:
Ja.

6.10.1) Wenn ja, beschreiben Sie diese.
Antwort:
Das Deployment ist sehr zeitaufwändig. Außerdem fällt der Delegs-Editor während der Durchführung des Deployment-Prozesses für Benutzer aus.

6.11) Sehen Sie Behinderungen, die den Fluss des Deployment-Prozesses aufhalten?
Antwort:
Ja.

6.11.1) Wenn ja, welche?
Antwort:
Das Kompilieren in GWT dauert teilweise sehr lange und blockiert damit einen Arbeitsrechner.

7.) Gibt es Wünsche oder Verbesserungsvorschläge für Ihr Projekt,

7.1) in Bezug auf den Deployment-Prozess?
Antwort:
Ja, ein automatischer Deployment-Prozess wäre sehr sinnvoll. Es wäre auch sinnvoll,wenn ein Deployment vorgenommen werden könnte, ohne dass währenddessen Anwender den Zugang zum delegs-Editor verlieren und nicht mehr mit der Anwendung arbeiten können.

7.2) in Bezug auf den Informationsfluss, hinsichtlich der Rückmeldungen durch den Deployment-Prozess (sowohl an die direkt beteiligten Kräfte, als auch an die nicht direkt beteiligten Kräfte)?
Antwort:
Ja, es wäre wahrscheinlich sinnvoll, dass Rückmeldungen über den E-Mail Verteiler au-

tomatisch versandt werden.

7.3) in Bezug auf das Testen?

Antwort:

Ein Build-Server wäre sinnvoll.

7.4) in Bezug auf die Verteilung von Aufgaben und Verantwortlichkeiten?

Antwort:

—

8.) Haben Sie noch weitere Anmerkungen?

Antwort:

—

Interviews mit dem Lehrteam

Die Fragen an das Lehrteam sind bewusst nicht technisch, sondern beziehen sich auf die Zusammenarbeit mit dem Tech-Team.

Interview mit Katrin Mrohs

Speziell zusammengestellte Fragen an das Lehrteam des delegs-Projekts (kein IT-Personal).
Interviewte Person: Katrin Mrohs
Kontext: delegs-Projekt / DevOps
Datum des Interviews: 14.12.2016

Folgende Fragen wurden während des Interviews gestellt:

1.) Wie findet die Kommunikation mit dem Tech-Team statt?

Antwort:

Es gibt einen wöchentlichen Austausch durch ein regelmäßig stattfindendes Team-Meeting. Dort können etwaige Probleme besprochen werden. Außerdem sitzt das Lehrteam im gleichen Büro wie das Tech-Team. Dadurch ist ein direktes Gespräch bei Problemen kurzfristig möglich.

1.1) Sehen Sie hier Verbesserungsmöglichkeiten?

Antwort:

Die Meetings sind manchmal etwas zu technisch, Nachfragen ist aber möglich. Wenn über technische Probleme gesprochen wird, sollten diese beispielhafter erklärt und direkt an der Anwendung gezeigt werden.

Außerdem ist es manchmal schwierig nachzuvollziehen, woran das Tech-Team gerade arbeitet. Absprachen darüber, was als nächstes dran ist, sollten klarer kommuniziert

werden und vom Lehrteam mit priorisiert werden. Zum Beispiel sollte zusammen mit dem Lehrteam abgestimmt werden, ob der Bedarf für das, was gerade eingeplant werden soll, überhaupt da ist.

2.) Wie gehen Sie vor, wenn Ihnen Fehler oder Probleme in einer Testversion auffallen?

Antwort:

Ich versuche den Fehler selbst zu beheben, vielleicht war es nur ein Anwendungsfehler von mir. Wenn ein Fehler während der Arbeit auftritt, zeige ich ihn direkt dem Tech-Team, sofern jemand aus dem Team im Büro ist. Wenn niemand vom Tech-Team da ist oder es gerade nicht passt, schreibe ich eine Notiz. Ich teile den Fehler dann mit, wenn ich wieder mit dem Tech-Team zusammenarbeite oder wenn das nicht passt, dann beim nächsten Montagsmeeting.

2.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

Antwort:

Nein.

3.) Wie gehen Sie vor, wenn Ihnen Fehler oder Probleme in einer Produktivversion auffallen?

Antwort:

Das Vorgehen ist das gleiche wie auch bei der Testversion. Allerdings ist das Arbeiten mit der Testversion anders als mit der Produktivversion. In der Testversion versuche ich explizit ein Dokument nachzubauen, um so die Anwendung auf Richtigkeit zu prüfen. In der Produktivversion tauchen Fehler eher durch Zufall auf, während des allgemeinen Arbeitsprozesses. Dann fotografiere ich den Fehler ab und schreibe eine Mail, wenn dieser sehr schwerwiegend ist. Ist das Tech-Team erreichbar, spreche ich direkt mit diesem. Manche kleineren nicht so gravierenden Fehler bespreche ich im Team-Meeting, montags.

3.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

Antwort:

Nein.

4.) Wie gehen Sie vor, wenn Ihnen Verbesserungsvorschläge für die Anwendung einfallen?

Antwort:

Ich notiere sie mir und schlage sie im Team-Meeting vor. Dort wird vom ganzen Team geklärt, ob die Idee umsetzbar ist. Das geht durch die Kontinuierlichkeit und Regelmäßigkeit der Meetings gut.

4.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

Antwort:

Nein, es ist gut so, da im Team-Meeting alle da sind und sich ausgetauscht werden kann.

5.) Haben Sie den Eindruck, dass Ihr Feedback aufgenommen und entsprechende Maßnahmen umgesetzt werden?

Antwort:

Ja, es wird aufgenommen und Maßnahmen ergriffen. Dadurch, dass immer viel ansteht und es viele offene Baustellen gibt, dauert die Umsetzung manchmal länger, allerdings nicht bei Sachen, die wirklich wichtig sind. Solche werden direkt umgesetzt.

6.) Haben Sie noch weitere Anmerkungen?

Antwort:

Es wäre gut, direkt die Kommunikation zu suchen, um Themen anzusprechen oder sich dafür zu verabreden. Das gilt für die Kommunikation in beide Richtungen zwischen Tech-Team und Lehrteam. Warten zu müssen, bis ein Meeting stattfindet, ist nicht wirklich gut und eher kontra-produktiv.

Interview mit Thimo Kleyboldt

Speziell zusammengestellte Fragen an das Lehrteam des delegs-Projekts (kein IT-Personal).
Interviewte Person: Thimo Kleyboldt
Kontext: delegs-Projekt / DevOps
Datum des Interviews: 14.12.2016
Anmerkung: Das Interview wurde mit Hilfe einer Dolmetscherin geführt. Der Befragte ist Muttersprachler der Deutschen Gebärdensprache.

Folgende Fragen wurden während des Interviews gestellt:

1.) Wie findet die Kommunikation mit dem Tech-Team statt?

Antwort:

Über Gebärdensprachdolmetscher, in Textform per Skype, über direkte Kommunikation mit den Kollegen die DGS beherrschen oder per Mail. Ich sprechen Probleme oft direkt im Büro an. Wenn etwas sehr speziell ist, warte ich auf die Möglichkeit einen Dolmetscher zu bekommen oder bis meine Kollegin da ist, die für mich übersetzen kann.

Wenn es im Tech-Team niemanden gäbe, der DGS könnte, wäre es weitaus schwieriger zu kommunizieren. Manchmal will ich das Tech-Team nicht unterbrechen und schreibe mir die Probleme auf, um sie im Meeting anzusprechen. Dann laufe ich aber Gefahr, diese nicht mehr reproduzieren zu können. Da es im Tech-Team über die Woche großen Wechsel in der Personalbesetzung gibt, bin ich mir manchmal nicht sicher, ob alle meine Anmerkungen mitbekommen haben.

1.1) Sehen Sie hier Verbesserungsmöglichkeiten?

Antwort:

Manchmal sind die Team-Meetings zu technisch. Für mich ist das dann teilweise Zeitverschwendung.

Es wäre gut, den Fokus im Meeting mehr darauf zu haben, was gerade sinnvoll ist für das Lehrteam und nicht direkt in die Tiefe über die technischen Möglichkeiten in der Umsetzung zu gehen. Ich fände es besser, wenn erst die Punkte des Lehrteams mit allen zusammengesammelt und besprochen würde. Das Tech-Team kann sich dann später darüber Gedanken machen, wie diese technisch zu bearbeiten sind. Danach könnten man dann wieder mit allen zusammenkommen und nochmal über die Ergebnisse sprechen und entscheiden, was umgesetzt werden soll und was nicht.

2.) Wie gehen Sie vor, wenn Ihnen Fehler oder Probleme in einer Testversion auffallen?

Antwort:

Wenn jemand vom Tech-Team da ist, sage ich direkt Bescheid. Ansonsten merke ich mir den Fehler oder das Problem und spreche darüber im Montagsmeeting.

Für mich ist es das Beste, direkt Bescheid zu sagen und den Fehler zu zeigen, es ist aber die Frage, ob das auch für das Tech-Team gut ist. Vielleicht reiße ich das Team auch aus ihrer Arbeit heraus und störe ihre Konzentration.

2.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

Antwort:

Nein.

3.) Wie gehen Sie vor, wenn Ihnen Fehler oder Probleme in einer Produktivversion auffallen?

Antwort:

Ich gehe hier genauso vor, wie bei Fehler in der Testversion.

3.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

Antwort:

—

4.) Wie gehen Sie vor, wenn Ihnen Verbesserungsvorschläge für die Anwendung einfallen?

Antwort:

Ich spreche nicht direkt darüber mit dem Tech-Team, sondern warte auf das nächste Meeting. Vorher tausche ich mich mit meiner Kollegin aus dem Lehrteam aus und wir diskutieren darüber, ob mein Vorschlag sinnvoll ist oder eventuell schon aufgenommen wurde. Danach kommuniziere

ich meine Idee mit dem Tech-Team im Team-Meeting. Ich spreche nicht direkt mit dem Tech-Team, da ich es nicht stören will.

4.1) Sehen Sie hier Potenzial zur Verbesserung, sowohl für Kommunikation als auch für das Vorgehen in einem solchen Fall? Was sollte geändert werden?

Antwort:

Nein.

5.) Haben Sie den Eindruck, dass Ihr Feedback aufgenommen und entsprechende Maßnahmen umgesetzt werden?

Antwort:

Das ist sehr unterschiedlich. Manchmal gibt es eine sehr schnelle Umsetzung. Manchmal wird das, was ich anspreche, als gering priorisiert. Dann wird kommt es teilweise sehr viel später zu einer Umsetzung.

Mir fehlt die Übersicht darüber, was gerade dran ist und vom Tech-Team umgesetzt wird.

Manche Bugs bringe ich auch gar nicht ein, weil ich denke, dass das gerade zu viel ist, weil das Tech-Team eine sehr lange Liste an offenen Aufgaben hat.

6.) Haben Sie noch weitere Anmerkungen?

Antwort:

Es war in der letzten Zeit nicht mehr so transparent, was gerade vom Tech-Team gemacht wird und was in Zukunft gemacht werden soll. Es wäre gut, wieder mehr Austausch zwischen den beiden Teams zu haben.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 16.05.2017 Fritz Oscar Stephan Berngruber