HOCHSCHULE-RHEIN-WAAL

FAKULTÄT KOMMUNIKATION UND UMWELT

BACHELORARBEIT

Entwurf und Implementierung eines Monitoringsystems für Continuous Delivery Pipelines am Beispiel der Schleupen AG

Joshua Siegemund (16960)

1. Betreuer

Prof. Dr. Thomas RICHTER

2. Betreuer

Dipl. -Inf. Sebastian JANCKE

1. Juni 2018

Abstract

The software company Schleupen AG is changing their development architecture to continuous delivery (CD). Continuous delivery describes a number of processes that enable faster and safer software deployment. At the end of the change the company will have an accumulation of automated steps.
This bachelor thesis is about the development of a system that monitors the continuous delivery process. The main focuses are the implementation of a dashboard and the use of telemetry.

Schleupen AG ändert ihre Entwicklungsarchitektur zu Continuous Delivery (CD). Continuous Delivery beschreibt eine Reihe von Vorgängen, die als Ziel eine schnellere und sicherere Softwarebereitstellung haben. Am Ende der Umstellung wird das Unternehmen eine Ansammlung von automatisierten Schritten besitzen.
In dieser Bachelorarbeit wird ein Monitoringsystem, das eine Überwachung des Continuous Delivery Prozesses ermöglicht, entwickelt. Die wesentlichen Ziele sind die Erstellung eines Dashboards und die Benutzung von Telemetrie.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AIT Automatische Integrationstests

CI Continuous Integration

CD Continuous Delivery

ERP Enterprise Resource Planning

GPT Geschäftsprozess Tests

Dev Development

RC Release Candidate

CR Certified Release

PSI Potentially Shippable Increment

TFS Team Foundation Server

SSE Server Sent Events

QS Qualität und Sicherheit

API Application Programming Interface

TFVC Team Foundation Version Control

Abbildungsverzeichnis

Teil I

Einleitung

1 Unternehmen

Das Unternehmen Schleupen AG bietet Softwarelösungen sowohl für den Bereich der Energie- und Wasserwirtschaft als auch für den Bereich Governance, Risk und Compliance. Das Unternehmen beschäftigt mehr als 440 Mitarbeiter. Die Anforderungen an die Software sind für jeden Kunden verschieden und können durch ein flexibles ERP-System variabel angepasst werden.

2 Problembeschreibung

Im Bereich Energie und Wasserwirtschaft besteht das Unternehmen aus mehreren Abteilungen, in denen Softwareentwickler an unterschiedlichen Problemen arbeiten. Da am Ende der Produktionslinie jedoch nur ein Produkt steht, müssen die einzelnen Lösungen zusammengebracht werden. Um dies zu bewerkstelligen, werden die Komponenten ausgiebig auf ihre Funktionalität und Integrität getestet. Dieser Vorgang ist zeitintensiv und erfordert teilweise manuelle Eingriffe.

2.1 Continuous Delivery

Bei Beginn dieser Arbeit findet bei Schleupen eine Umstellung des Entwicklungsprozesses in Richtung Continuous Delivery statt. Continuous Delivery beschreibt eine Reihe von Vorgängen, die den Produktionszyklus verbessern sollen. Ziel ist es, die Software kontinuierlich und automatisiert auszuliefern. Jez Humble, , schreibt dazu: „Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way “(Humble, 2010). Bei Änderungen an der Software wird eine Continuous Delivery (CD)-Pipeline durchlaufen. In dieser werden automatisierte Tests ausgeführt. Nach erfolgreichem Durchlauf der Pipeline wird eine neue Release-Version erstellt, wodurch ein kontinuierlicher Entwicklungsprozess gewährleistet wird (vgl. Humble und Farley, 2010, S.107)

Die CD-Pipeline wird parallel zur bestehenden Produktionslinie eingeführt. Als Grundlage wird zum einen der Team Foundation Server (TFS) von Microsoft verwendet. Zum

anderen wird das webbasierte System von Jenkins benutzt.

Um Informationen zu erlangen, können die API-Schnittstellen vom TFS und von Jenkins benutzt werden.

2.2 Projektziel

Im 2016 erschienen DevOps-Handbook steht: „One of the defining characteristics of a complex system is that it defies any single person's ability to see the system as a whole and understand how all the pieces fit together"(Kim et al., 2016, S.56). Diese Aussage spiegelt sich auch im Ziel dieser Arbeit wieder. Die Mitarbeiter im Unternehmen sollen ein allgemeines Verständnis für den Ablauf der CD-Pipeline entwickeln. Sowohl die Vorgänge der einzelnen Teilabschnitte als auch die Zusammensetzung des Gesamtprozesses sollen den Anwendern vermittelt werden. Zusätzlich soll das entwickelte Monitoring bei der Lösung von Auffälligkeiten und Problemen helfen.

Innerhalb der Bachelorarbeit wird die Architektur für ein Monitoringsystem erstellt und als Prototyp implementiert. Das Hauptziel ist die Entwicklung eines Dashboards. Das Dashboard soll anpassbar an die jeweiligen Team-Umgebungen des Unternehmens sein. Als Anzeigeflächen werden sowohl TV- als auch Desktopmonitore verwendet.

Teil II

Grundlagen

3 Telemetrie zur Unterstützung von Continuous Delivery

In dem Buch Continuous Delivery wird im Zusammenhang mit Telemetrie der Hawthorne Effekt erwähnt (vgl. Humble und Farley, 2010, S. 137). Dieser sagt aus, dass Menschen ihr Verhalten ändern, wenn sie wissen, dass sie beobachtet werden. Durch die Einführung von Monitoring in einem Softwareunternehmen kann eine solche Verhaltensänderung bewusst hervorgerufen werden. Die Metriken, die in den Mittelpunkt von Analysen gestellt werden, bekommen eine besondere Aufmerksamkeit. Aufgrund dessen muss vorher geklärt sein, ob die Auswirkungen durch die Darstellung der gewählten Telemetrie erwünscht sind.

Beispielhaft kann als Hauptmetrik die Anzahl der Codezeilen gemessen und aufbereitet werden. Die Entwickler bekommen dadurch ein positives Feedback, wenn sie Implementationen in wenigen Zeilen umsetzen (vgl. Humble und Farley, 2010, S. 137 f.). Andernfalls kann ein Fokus auf die Testabdeckung gelegt werden. Dieser Schwerpunkt würde wahrscheinlich in einer Quantitätssteigerung der Tests resultieren.

3.1 Auswahl von Metriken

Im Folgenden werden die für das Projekt betrachteten Metriken vorgestellt. Die Anzeige auf dem Dashboard soll dabei helfen, eine Verbesserung in den jeweiligen Themengebieten zu erzielen.

- **Durchlaufzeit:** Wie lange dauert ein Durchlauf der CD-Pipeline?
- **Broken Build Time:** Wie lange war ein Build nicht erfolgreich?
- **Prozentzahl der fehlgeschlagenen Builds**
- **Durchschnittliche Anzahl von Komponenten, die durch einen Build geändert wurden**

Ein Hauptziel der Umstellung auf Continuous Delivery ist eine Beschleunigung des Entwicklungsprozesses. Um diese Anforderung zu analysieren, wird die Durchlaufzeit benötigt. Aus diesem Grund wird für den Prototypen des Monitoringsystems die Durchlaufzeit ana-

lysiert. Eine Definition der Durchlaufzeit erfolgt in Kapitel 5. Die weiteren Metriken können in folgenden Arbeiten umgesetzt werden.

4 Continuous Delivery bei Schleupen

In diesem Projekt soll ein Dashboard implementiert werden, das die CD-Pipeline von Schleupen widerspiegeln kann. Um ein besseres Verständnis über den Entwicklungsprozess in einer CD-Pipeline zu bekommen, wird im Folgenden der geplante Vorgang bei Schleupen beschrieben. Die Architektur in Abbildung 1 bietet eine einfache Darstellung der Continuous Delivery Pipeline.

Abbildung 1: Einfache Darstellung der Continuous Delivery Pipeline von Schleupen [8]

4.1 Continuous Integration Build (CI)

Die Pipeline besteht aus mehreren Stufen, die durchlaufen werden. Den Startpunkt einer CD-Pipeline bilden Continuous Integration (CI) Builds[1]. Diese starten jedes Mal, wenn Codeänderungen in die Versionsverwaltung des TFS eingecheckt wird. Zu jedem Softwaremodul von Schleupen existiert ein CI-Build.

Ein CI-Build vollzieht alle funktionalen Tests, die keine installierte Software benötigen. Beispiele dafür sind Unit- oder Komponententests. Weiterhin wird eine neue Version der Produkteinheit gebaut und die Versionsnummer hochgezählt. Die genauen Buildschritte werden im TFS durch eine Builddefinition bestimmt.

4.2 Qualitätsstufen

Nach dem Abschluss eines CI-Builds wird die gebaute Komponente in der Qualitätsstufe **CI** gemeldet. Eine Qualitätsstufe bildet einen Informationsspeicher zwischen zwei Pipelineabschnitten. Wie in Abbildung 1 zu sehen, gibt es noch die Qualitätsstufen **Develop-**

[1] In der Abbildung als CD gekennzeichnet

ment (Dev) und **Release Candidate (RC)**, welche in den folgenden Abschnitten erreicht werden. Weiterhin existieren dieselben Elemente für die Plattformebene, da eine Trennung der Ebenen stattfindet (siehe Kapitel 4.6).

Die Qualitätsstufen sind durch ein sogenanntes Artifact Repository implementiert. Im Artifact Repository werden die Informationen der Schleupen Komponenten im JSON-Format festgehalten. Gespeichert sind Daten wie Name, Abkürzung und Versionsnummer einer Komponente. Für jede Qualitätsstufe existiert ein Ordner.

4.3 Automatische Integrationstests (AIT)

Automatische Integrationstests finden auf Teamebene statt. Als Grundlage wird der Automatisierungsserver Jenkins benutzt. Auf den Testmaschinen ist eine aktuelle Version von Schleupen installiert. Die funktionalen Änderungen werden an dieser Stelle in das vorhandene System integriert und auf Kompatibilität getestet. Die AIT-Builds enden in der Qualitätsstufe Dev.

4.4 Geschäftsprozesstests (GPT)

Geschäftsprozesstests stellen sicher, dass die Funktionalität des Gesamtprozesses bestehen bleibt. Im Gegensatz zu AITs sind Geschäftsprozesstests teamübergreifend. Am Ende eines GPT-Builds wird die Qualitätsstufe RC erreicht. Eine Version, die in RC gemeldet wurde, ist soweit getestet, dass sie für eine Freigabe geeignet wäre.

4.5 Betriebliche Tests im Bereich Qualität und Sicherung (QS)

Bevor Versionen zum Kunden ausgeliefert werden, erfolgen betriebliche Tests zur Qualitätssicherung. Diese beinhalten sowohl manuelle als auch automatische Tests. In Abbildung 2 ist eine detailliertere Darstellung der CD-Pipeline gegeben. In dieser wird nach einem RC-Stand der Begriff HIP-Integrationstests (**HIP-IT**) benutzt. HIP steht für 'Hardening - Innovation - Planning' und beschreibt einen Iterationsabschnitt im Entwicklungsmodell von Schleupen. Die HIP-ITs unterliegen ebenfalls dem Aufgabenbereich der QS. In ihnen finden die letzten manuellen Tests statt. Nach der Korrektur aller Fehler aus dem HIP-IT, endet der Entwicklungsprozess in der Qualitätsstufe **Certified Release (CR)**.

4.6 Trennung zwischen Plattform und Anwendungen

Die Anwendungssoftware ist abhängig von der Plattform. Um eine zusätzliche Absicherung der Kompatibilität zu wahren, besitzen beide Ebenen getrennte CD-Pipelines. Die

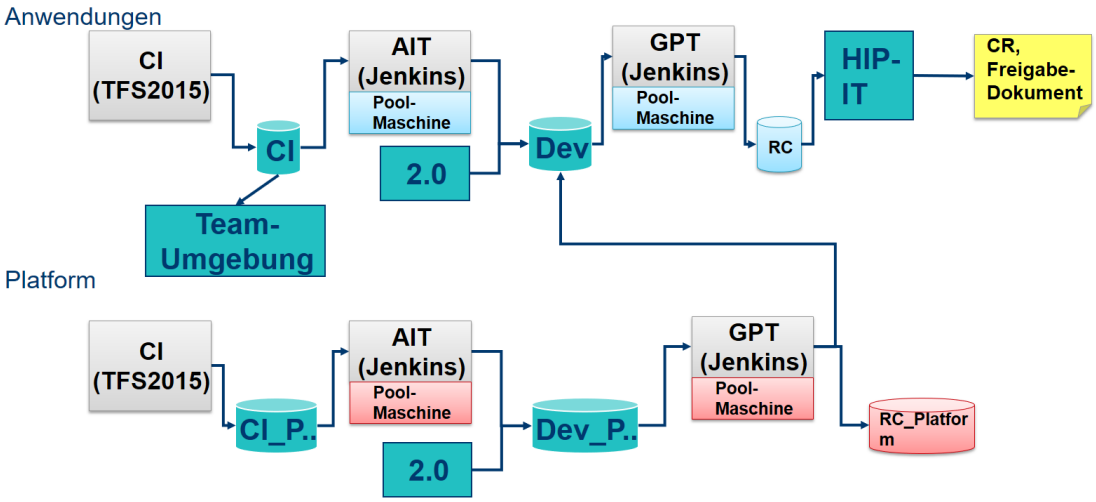

Abbildung 2: Darstellung der Continuous Delivery Pipeline von Schleupen mit Ebenen [8]

Plattformabschnitte erreichen, wie in Abbildung 2 dargestellt, eigene Qualitätsstufen. Eine Verbindung zwischen Anwendung und Plattform findet statt, wenn die Release Versionen der Plattformkomponenten in die Qualitätsstufe Dev der Anwendungsebene geschoben werden.

5 Definition der Durchlaufzeit

Im Zusammenhang von Produktionszyklen bestehen eine Vielzahl von unterschiedlichen Zeitabschnitten, die für eine Überwachung relevant sein könnten. Zum Beispiel können die Zeiten der Bearbeitung einer Kundenanfrage, der Auslieferung eines Produktes oder des Durchlaufs eines Gesamtprozesses gemessen werden. Für die Benennung dieser Zeiten gibt es mehrere Begriffe wie Produktionszeit, Durchlaufzeit oder Zykluszeit.

In dieser Arbeit wird als zeitlicher Orientierungspunkt die Bezeichnung Durchlaufzeit benutzt. Da es leicht zu Missverständnissen zwischen diesen Ausdrücken kommen kann, findet im Folgenden eine Definition der Durchlaufzeit Im Kontext von Schleupen statt.

Der aktuelle Durchlauf des gesamten Auslieferungsprozesses von Schleupen dauert ungefähr 3 Monate. Diese Zeit ist durch das Entwicklungsmodell von Schleupen vorgegeben und wird sich durch die Umstellung auf Continuous Delivery nicht verändern. Für das Monitoring soll die Durchlaufzeit der CD-Pipeline ermittelt werden, da diese Information Aussagen zur Effizienz der Continuous Delivery Architektur zulässt. Die Durchlaufzeit soll als Unterstützung für interne Entwickler dienen. Wenn ein Entwickler Funktionen implementiert oder Fehler beseitigt, soll er abschätzen können, wann seine Änderungen für eine Freigabe geeignet wären.

Die Durchlaufzeit ist definiert als die Zeit, die eine neu gebaute Komponentenversion für den **Durchlauf der CD-Pipeline** benötigt. Beginnend bei der Qualitätsstufe **CI**, endend bei der Qualitätsstufe **RC**.

Teil III

Analyse

6 Pflichtenheft für das CD-Pipeline Dashboard

In diesem Kapitel wird ein Pflichtenheft gebildet. Als Vorlage wird die Anforderungsschablone aus dem Buch ”Lehrbuch der Softwaretechnik”(Balzert, , 2009, S. 495 f.) verwendet. Zusätzliche Orientierung gaben die Fallstudien auf den Seiten 107-126.

6.1 Visionen und Ziele

[V01] Das Dashboard ermöglicht ein Monitoring der Continuous Delivery Pipeline.
[Z01] Personen aus dem Management können auf dem Dashboard eine Übersicht von Buildinformationen zur Pipeline sehen.
[Z02] Personen aus dem Management können die Durchlaufzeit der CD-Pipeline sehen.
[Z03] Entwickler können erkennen, **welche** Builds nicht erfolgreich waren, um das Auftreten von Problemen zu bemerken.
[Z04] Entwickler können erkennen, **warum** ein Build nicht erfolgreich war, um Fehler zu beheben.

6.2 Rahmenbedingungen

[R01] Das Dashboard ist eine Webanwendung.
[R02] Zielgruppe sind alle Mitarbeiter von Schleupen im Bereich der Energie und Wasserwirtschaft.
[R03] Das Dashboard soll durchgehend während der Arbeitszeit zur Verfügung stehen.
[R04] Software auf Clients: Webbrowser - Unterstützung der zwei marktführenden Webbrowser
[R05] Alle Clients und der Server befinden sich im Intranet von Schleupen.

6.3 Kontext und Überblick

[K01] Das System besitzt eine Schnittstelle zu Jenkins.
[K02] Das System besitzt eine Schnittstelle zu TFS.

6.4 Funktionale Anforderungen

[F01] Darstellung von Jenkins Builds

Auf dem Dashboard sind Jenkins Builds graphisch aufbereitet. Der Status, der zuletzt gelaufenen Builds, ist schnell und anschaulich einsehbar.

[F02] Darstellung von TFS Builds

Es muss möglich sein, CI-Builds aus dem TFS darzustellen. Diese Anforderung soll durch Konfiguration ein-/ausschaltbar sein. Der Status, der zuletzt gelaufenen Builds, ist schnell und anschaulich einsehbar.

[F03] Darstellung der CD-Pipeline

Ein Anwender kann durch das Dashboard den Ablauf und Status der Pipeline nachvollziehen. Sowohl TFS- als auch Jenkins-Builds sind funktional zusammengefasst.

[F04] Darstellung der Durchlaufzeit

Auf dem Dashboard ist eine Anzeige der Durchlaufzeit zu sehen. Es ist der aktuelle Wert sowie ein Vergleich zu vorherigen Werten gegeben. Der Anwender soll schätzen können, wie viel Zeit eine Komponente für den Durchlauf der Pipeline benötigt.

[F05] Konfiguration eines Dashboards

Ein Anwender muss ein Dashboard konfigurieren können. Die Auswahl und Position der Widgets muss bestimmbar sein.

[F06] Erstellung mehrerer Dashboards

Für die verschiedenen Teams im Unternehmen können individuelle Dashboards erstellt und gleichzeitig genutzt werden. Jedes Team sollte sein Dashboard über eine eigene URL erreichen können.

[F07] Weiterleitungen

Über des Dashboard können die Zusammenfassungen der Jenkins- oder TFS-Builds auf den jeweiligen Seiten erreicht werden.

6.5 Qualitätsanforderungen nach ISO/IEC 9126

In diesem Kapitel werden die Qualitätsanforderungen an das System bewertet. Als Vorlage wird die ISO-Norm 9126 verwendet. Definitionen der einzelnen Merkmale sind im

”Lehrbuch der Softwaretechnik”(Balzert, 2009, S.468 ff.) gegeben. Im Anhang ist eine Tabelle mit Bewertungen der Qualitätsanforderungen gegeben (Tabelle 1). Im Folgenden werden die wichtigsten Anforderungen vorgestellt.

Funktionalität

Bei den Funktionalitäten ist der wichtigste Parameter die Genauigkeit. Das Dashboard darf keine ungenauen Ergebnisse darstellen. Dies gilt besonders für die Funktion der Durchlaufzeit, da ansonsten falsche Rückschlüsse gezogen werden könnten. Das Dashboard muss mit dem Continuous Delivery System von Schleupen zusammen arbeiten. Aus diesem Grund muss eine gute Interoperabilität gegeben sein.

Zuverlässigkeit

Das Dashboard muss zuverlässig Daten darstellen. Wenn beispielsweise keine Informationen zur CD-Pipeline abgerufen werden können, darf dies nicht in ein leeres Dashboard resultieren. Es sollte immer der letzte verfügbare Stand angezeigt werden.

Benutzbarkeit

Für die Benutzbarkeit sind die wichtigsten Punkte die Attraktivität und die Verständlichkeit. Beide Merkmale verfolgen das Ziel, ein übersichtliches Dashboard zu gestalten. Anwender sollen schnell und einfach an Informationen gelangen.

Effizienz

Das Dashboard soll die Daten in regelmäßigen Abständen aktualisieren. Bei Neuladen der Webseite sollten in kurzer Zeit die letzten aktuellen Ergebnisse angezeigt werden.

Wartbarkeit

Das Endprodukt soll durch weitere Layouts erweitert werden können. Sollten Fehler beim Betrieb des Dashboards auftreten, müssen deren Ursachen diagnostizierbar sein.

7 Anforderungen an das Dashboard Design

Einer der wichtigsten Wegbereiter für das Design von Dashboards ist Stephen Few. Er untersuchte Anfang der 2000er Jahre unterschiedliche Dashboards von Unternehmen (vgl. Stephen Few, 2006, S.9 ff.). Aus diesen Forschungen erstellte er einige Grundsätze, die heute immer noch Relevanz besitzen.

Stephen Few gab folgende Definition für ein Dashboards: „A dashboard is a visual display of the most important information needed to archieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance“(Stephen Few, 2004). Aus dieser Aussage lassen sich mehrere Punkte extrahieren. Zum einen soll der Nutzer durch das Dashboard einen **schnellen Überblick** bekommen. Die relevanten Daten sollen hervorgehoben und somit leicht erkennbar gestaltet sein. Zum anderen werden nur relevante Informationen dargestellt. Alle Daten, die nicht **zielführend** sind, sollten keine Nutzung finden.

Aus diesen Beobachtungen wurden folgende Schritte aufgestellt, welche die Anforderungen eines Nutzers an das Dashboard definieren:

- **[D01] Übersicht und Identifikation:** Muss etwas getan werden?
- **[D02] Analyse:** Wo liegt das Problem? Wen kann ich ansprechen?
- **[D03] Weiterleitung:** Detailliertere Informationen / andere Umgebungen

Das Monitoringsystem soll die Anwender dabei unterstützen, den momentanen Status der Continuous Delivery Pipeline nachzuvollziehen. Dazu muss auf den ersten Blick die Anforderung der ’Übersicht und Identifikation’ [D01] erfüllt werden. Die CD-Pipeline ist ein komplexer Vorgang mit vielen Iterationen. Durch die graphische Darstellung derselben soll einerseits eine Veranschaulichung der Pipeline geschehen, andererseits sollen Probleme oder Fehler identifiziert werden. Die Anforderungen der Analyse und der Weiterleitung [D02, D03] sollen den Nutzern die Behebung der Probleme ermöglichen.

8 Auswahl der Dashboard Software

In diesem Kapitel werden die Dashboard Frameworks vorgestellt, die für eine spätere Implementation in Betracht gezogen wurden. Für die Entscheidung werden sowohl allgemeine als auch spezifische Anforderungen (s. Kapitel 6.4) berücksichtigt.

Eine Grundanforderung an die Frameworks ist, das eine Erstellung von eigenen Widgets möglich sein muss. Um die CD-Pipeline darzustellen, muss . Das Dashboard soll für mehrere Teams nutzbar sein. Deshalb ist eine weitere Anforderung, dass unterschiedliche Layouts erstellt werden können. Um eine qualifiziertere Entscheidung zu treffen, wurde zudem das Dashboard Selection Model (Staron, Niesel und Meding, 2015, S.130 ff.) verwendet.

8.1 Dashboard Selection Model

Das *Dashboard Selection Model* soll bei der Entscheidung helfen, welche Visualisierung für Messdaten und Metriken geeignet ist. Das Modell stellt sieben Parameter auf. Jeder Parameter besitzt zwei Randwerte. Im Folgenden werden die Parameter, mit ihren Entscheidungsmöglichkeiten vorgestellt. Im nächsten Schritt werden die Randwerte benannt, die auf das Dashboard zutreffen.

- **Typ des Dashboards:** Bericht oder Dashboard
- **Datenbeschaffung:** Manuell oder Automatisch
- **Stakeholders:** Einzelpersonen oder Gruppen
- **Datenzulieferung:** Abruf oder Zulieferung
- **Update:** Periodisch oder Kontinuierlich
- **Ziel:** Information oder Entscheidungshilfe
- **Datenfluss:** Rohdaten oder Indikatoren

Die Darstellung der CD-Pipeline soll aktuelle Informationen widerspiegeln. Aus diesem Grund muss eine **automatische, kontinuierliche Abfrage** der Daten erfolgen. Die Stakeholder umfassen eine **Gruppe** von Personen. Die Darstellungen der CD-Pipeline und der Durchlaufzeit sollen die Anwender zu weiteren Handlungen bewegen. Aus diesem Grund tendiert das Ziel des Dashboards zu einer **Unterstützung bei Entscheidungen**. Der Datenfluss beschreibt, ob das Dashboard die zusätzlich Daten aufbereiten muss. Im vorliegenden Projekt ist dies der Fall, sodass der Datenfluss als **Indikator** bestimmt ist.

Mithilfe der Anforderungen aus diesem Modell und den Anforderungen aus Kapitel 6.4 wurden mehrere Dashboards überprüft. Die zwei Frameworks Smashing und Mozaik besitzen die Möglichkeit alle Anforderungen abzubilden. Im Folgenden werden Beide genauer betrachtet. Mithilfe der Vor- und Nachteile soll herausgestellt werden, welches Framework, die Ziele am effizientesten erreichen kann.

8.2 Smashing

Smashing ist eine Weiterführung von Dashing, welches von Shopify entwickelt wurde. Die Grundlage bildet das Webframework Sinatra. Smashing bietet eine Bibliothek mit zusätzlichen Widgets. Das Erstellen von Widgets wird durch konfigurative Befehle erleichtert. Zur Einrichtung von Dashboards werden Layoutdateien verwendet. Jedes Layout kann über eine eigene URL aufgerufen werden. Smashing bietet ein Sparklines- und ein Chartjs-Plugin. Diese ermöglichen das Integrieren von Diagrammen.

Die Widgets sind nicht skalierbar, sodass zum Beispiel eine Nutzung auf einem Smartphone nicht möglich ist. Smashing benutzt Server Sent Events, um Daten zu aktualisieren. Da der Microsoft Internet Explorer keine Server Sent Events unterstützt, ist Smashing für diese Plattform nicht verfügbar. Momentan gilt das auch gleiche für Microsoft Edge. Durch die Rahmenbedingung [R04] ist dieser Nachteil jedoch zu ignorieren.

8.3 Mozaik

Mozaik basiert auf den Technologien nodejs, react, d3 und stylus. Das Layout des Dashboards basiert auf einem Gridsystem, welches im Gegensatz zu Smashing skalierbar ist. Mozaik benutzt Websockets, um Daten auf dem Dashboard zu aktualisieren. Damit ist eine Kompatibilität mit allen Browsern gegeben.

Als Negativpunkt ist zu erwähnen, dass die Bibliothek von zusätzlichen Widgets kleiner ist als bei Smashing. Weiterhin gibt es keine bestehenden Widgets für die Darstellung von Diagrammen. Auch ist das Hinzufügen von Dashboards, die auf weiteren URLs laufen, stark erschwert.

8.4 Entscheidung anhand der funktionalen Anforderungen

Die allgemeinen Vor- und Nachteile sind ausgeglichen und lassen noch keine aussagekräftige Entscheidung zu. Aus diesem Grund werden die funktionalen Anforderungen

aus Kapitel 6.4 herangezogen. Die Erstellung und Konfiguration von mehreren Dashboards [F05, F06] erfordert bei Mozaik zusätzliche Implementierungen. Bei Smashing ist diese Möglichkeit bereits gegeben. Zusätzlich kann die Darstellung der Durchlaufzeit [F04] bei Smashing durch bestehende Plugins realisiert werden. Alle anderen funktionalen Anforderungen können in beiden Frameworks umgesetzt werden. Deshalb wird für die Implementierung des Dashboards Smashing verwendet.

Teil IV

Architektur

9 Grundstruktur des Dashboards

In diesem Kapitel wird die Grundstruktur des Dashboards beschrieben.

9.1 Verfügbare Schnittstellen

Die Daten im Unternehmen können über verschiedene Schnittstellen erreicht werden. In Abbildung 3 wird gezeigt, welche Kommunikationsarten für das Abrufen der Informationen bestehen.

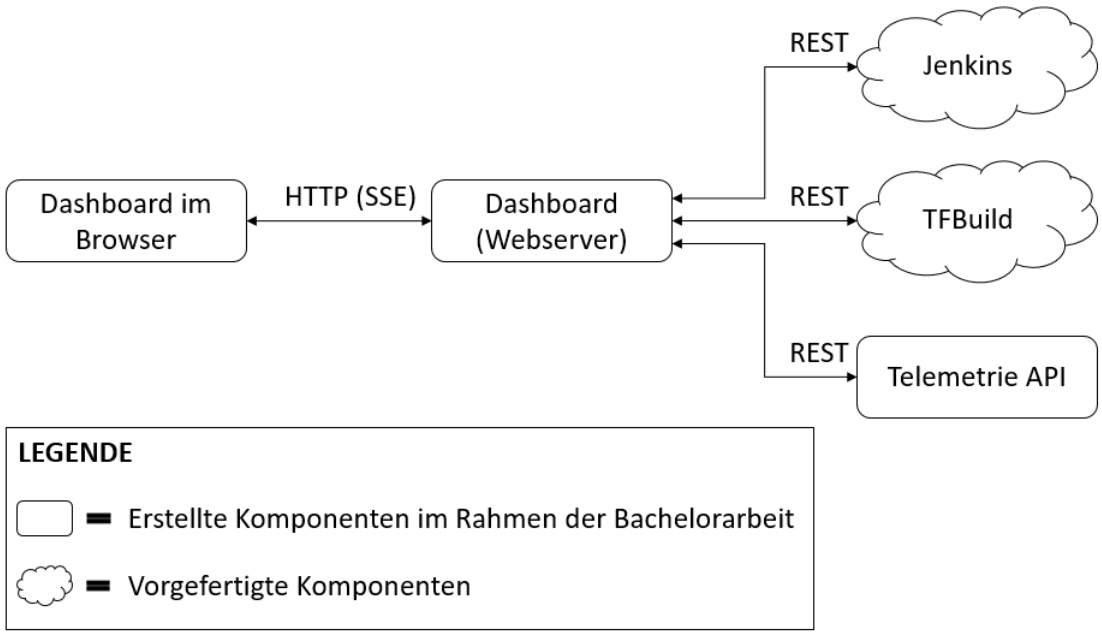

Abbildung 3: Schnittstellen des Dashboards

Jenkins und TFBuild[2] stellen beide eine REST API zur Verfügung. Es können Informationen wie der Status oder die URL eines Builds ermittelt werden. Diese Funktionalität wird von Smashing serverseitig zur Aktualisierung der Buildinformationen verwendet. Die gewonnen Informationen werden aufbereitet, an die Clients versendet und im Browser dargestellt.

Für die Anforderung [F04] soll Smashing die Durchlaufzeit darstellen. Damit dies möglich ist, wird eine Webanwendung implementiert, die eine API-Schnittstelle zur Telemetrie bildet. Die Architektur der API wird in Kapitel 10 erläutert.

[2] Buildsystem des TFS

9.2 Ausgehende Weiterleitungen des Dashboards

Ein Dashboard bietet zum einen die Zusammenfassung von aussagekräftiger Telemetrie. Zum anderen soll der Nutzer die Möglichkeit haben, zu detaillierten Informationen weitergeleitet zu werden [F07]. Um diese Anforderung umzusetzen, sollen die Vorteile einer Webanwendung ausgenutzt werden. Die Weiterleitungen auf dem Dashboard werden mithilfe von Links implementiert.

Für die Griddarstellung der Pipeline werden die URLs der Builds mitgeliefert. Mit diesen Informationen können die jeweiligen Buildseiten von Jenkins und TFS erreicht werden. Zusätzlich existieren weitere Links für Entwickler, die auf unternehmensinternen Seiten verweisen. Für fast jedes Element auf dem Dashboard gibt es eine Form der Weiterleitung. Damit wird aus dem Dashboard ein zentraler Punkt, welchen Entwickler als Start für Analysen nutzen können. In Abbildung 4 wird der Aufbau der Linkstruktur gezeigt.

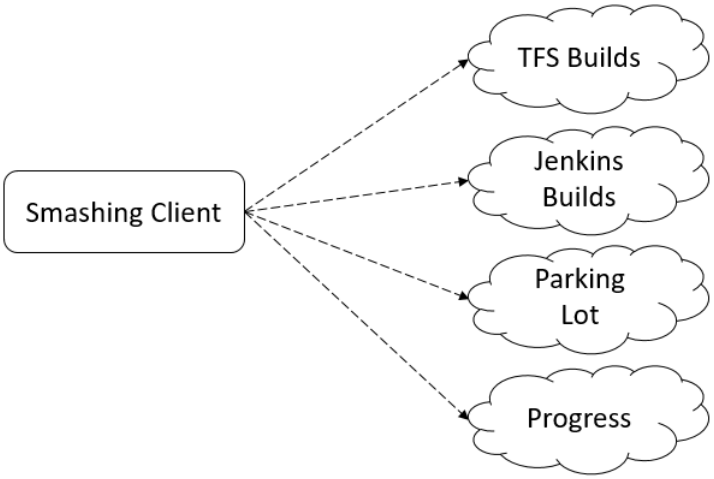

Abbildung 4: Weiterleitungen vom Dashboard

Mithilfe des Parking Lots können Entwickler die Maschinen, auf denen Pipeline-Abschnitte laufen, kontrollieren. Die Seite zeigt Informationen zum Namen und Status (active, idle ...) der Maschinen. Zusätzlich können Maschinen geparkt werden, sodass keine weiteren Builds auf ihnen gestartet werden. Diese Funktion wird zu Analysezwecken benötigt.

Die Progress-Seite gibt detaillierte Informationen zu einzelnen Komponenten. Zur Nutzung muss ein Anwender Name und Version einer Komponente angeben. Ein Abschnitt der Progress-Seite wird in Kapitel 16 vorgestellt.

Die vorgestellten Links sind nicht endgültig und können im Verlauf der Entwicklung erweitert werden. Die Implementierung, der hier aufgestellten Architektur, geschieht in Kapitel

15.

10 Architektur der Telemetrie

In diesem Kapitel erfolgt eine Einführung in die Benutzung und Darstellung von Telemetrie. Weiterhin wird die Architektur zur Speicherung und Abfrage von Metriken vorgestellt. Genauer wird die Durchlaufzeit, die in der Anzeige des Dashboards Verwendung findet, erläutert.

10.1 Speicherung von Metriken in Elasticsearch

Zur Speicherung und Abfrage der Daten wird Elasticsearch[1] benutzt. Elasticsearch ist eine Suchmaschine, die Analysen über große Datensätze ermöglicht. Zur Kategorisierung werden Indizes benutzt. Jeder Index kann eine Vielzahl von Dokumenten beinhalten. Zur Identifizierung bekommen die Dokumente eindeutige Ids zugewiesen. Die Metriken für das Dashboard sind über zwei Indizes verteilt.

Zum einen gibt es einen Index mit dem Namen *pipeline-telemetry*. In diesem werden Metriken zu allen Komponenten, welche die Pipeline durchlaufen, hinterlegt. Die Telemetrie enthält Metadaten wie Versionsnummer oder die momentane Entwicklungslinie. Zusätzlich werden die Zeitpunkte, an denen die Komponenten neue Qualitätsstufen erreichen, abgespeichert. In einer Pipelinehistorie befinden sich die Buildinformationen zu den entsprechenden Pipelineabschnitten. Der Aufbau eines Dokuments wird in Abbildung 5 noch einmal verdeutlicht.

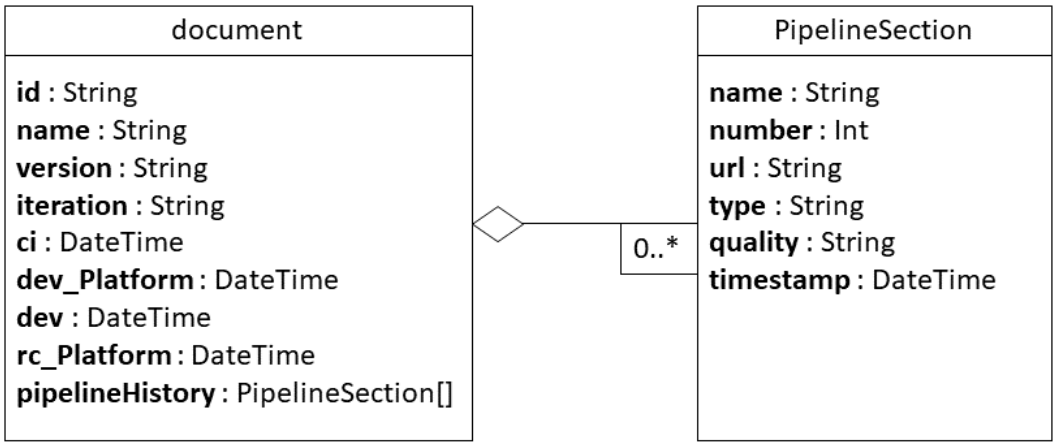

Abbildung 5: Aufbau des Telemetrie-Indexes in Elasticsearch

Die Durchlaufzeiten werden unter dem Index *pipeline-cycletime* abgelegt. Die Ids der

Dokumente sind durch die momentane PSI-Iteration definiert. Weiterhin werden die Komponenten des letzten Durchlaufs mit ihren Zeiten in einer Historie abgespeichert. Die Historie findet in der Detailseite Verwendung (Kapitel 16).

10.2 Telemetrie API zur Kommunikation mit Elasticsearch

Die Durchlaufzeit wird wie im Kapitel 10.1 erläutert in Elasticsearch gespeichert. Für die Kommunikation mit Elasticsearch wird eine dotnet Webapplikation erstellt. Über die Schnittstelle erfolgen sowohl Abfragen als auch Aktualisierungen der Daten. Elasticsearch stellt zwei offizielle dotnet Clients zur Verfügung. Zum einen den Low-Level Client Elasticsearch.Net. Zum anderen den High-Level Client NEST, welcher auf Elasticsearch.Net aufbaut. Für die Anwendung wird NEST verwendet. Die Struktur wird in Abbildung 6 verdeutlicht.

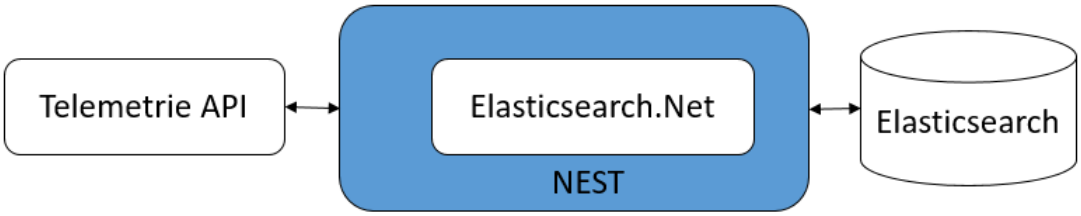

Abbildung 6: Architektur zwischen der Telemetrie API und Elasticsearch

Die Telemetrie API ist eine wichtige Schnittstelle des Projektes. Zur Orientierung werden im Folgenden werden alle Verwendungen der Schnittstelle aufgelistet:

- **Kapitel 13:** Artifact Repository Client
- **Kapitel 14:** Berechnung der Durchlaufzeit
- **Kapitel 15.1.4:** Abfrage der Durchlaufzeit vom Dashboard Server
- **Kapitel 16:** Anzeige von Detailinformationen zu Komponenten.

10.3 Zusammenfassung der Durchlaufzeiten

In Kapitel 5 wurde eine Definition für den Begriff Durchlaufzeit im Kontext von Schleupen aufgestellt. Aus dieser Definition wurde eine Anwendung entwickelt, die eine Ermittlung der Zeiten ermöglicht. Da eine Mehrzahl von Komponenten bestehen, existieren unterschiedliche Durchlaufzeiten. Für das Dashboard müssen die Informationen sinnvoll zusammengefasst werden.

Die Durchlaufzeit beschreibt den Zeitraum von der Erstellung einer neuen Komponentenversion bis zu ihrer Veröffentlichung. Da sich das Unternehmen in der Einführungsphase der CD-Pipeline befindet, kann es zu starken Ausreißern in den Werten kommen. Als Beispiel ist in Abbildung 7 die Verteilung der Durchlaufzeiten von ungefähr 50 Komponenten berechnet worden. Die Daten wurden aus Datenschutzgründen anonymisiert.

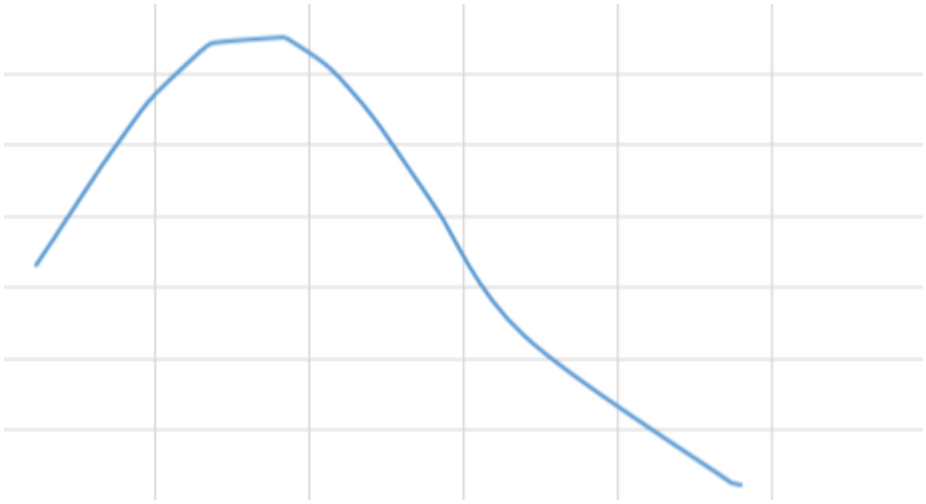

Abbildung 7: Verteilung der Durchlaufzeit bei 51 Komponenten

Die wichtige Information ist, dass das Diagramm sich deutlich von einer glockenförmigen Normalverteilung unterscheidet. Um zu beweisen, dass die Daten nicht normalverteilt sind, wurde ein zusätzliches Testverfahren, der Shapiro-Wilk-Test (Shapiro und Wilk, 1965) ausgeführt. Der Test ist besonders bei kleineren Datensätzen effektiv. Mithilfe der .Net Bibliothek Accord [11] wurde in einer einfachen Implementation der Datensatz aus Abbildung 7 überprüft.

Der Shapiro-Wilk-Test setzt als Null-Hypothese voraus, dass ein Datensatz normalverteilt ist. Die Hypothese ist dann widerlegt, wenn der p-Wert kleiner als das Signifikanzniveau ist. Als Signifikanzniveau wird ein Standardwert von 0.05 (5%) genommen.

Beim untersuchten Datensatz liegt der p-Wert deutlich unter 0.05. Somit sind die Durchlaufzeiten nicht normalverteilt.

Dieses Ergebnis beeinflusst die Entscheidung, auf welche Weise die Durchlaufzeiten zusammengefasst werden. Für einen Vergleich des Durchschnitts mit dem Median schreibt Raphael Albino: “As the data becomes skewed, the average loses its ability to provide the best central location for the data because the data will drag it away from the typical value“ [**?**]. Diese Aussage bestätigt sich bei den vorliegenden Daten, da sich Durchschnitt und

Median stark voneinander unterscheiden. Um den Einfluss der Randwerte zu minimieren und genauere Vorhersagen treffen zu können, wird deswegen als Kennzahl der Median benutzt.

In den gesammelten Metriken aus Kapitel 10.1 sind alle Zeiten gespeichert, an denen neue Qualitätsstufen erreicht werden. In Kapitel 14 wird beschrieben, wie diese Daten genutzt werden können.

Teil V

Design

11 Darstellung des Continuous Delivery Pipeline Status

Im Folgenden werden die Designansätze zur Darstellung der CD-Pipeline vorgestellt. Für die entwickelten Prototypen fanden kontinuierlich Reviews statt, sodass fortschreitend Verbesserungen erzielt werden konnten. Zur Vorstellung der Entwürfe wurden Mockups gestaltet. Wenn es zeitlich möglich war, sind auch Implementierungen entstanden.

11.1 Entwurf einer Baumdarstellung mithilfe von D3.js

Die erste Idee war die Benutzung eines *tree diagrams*. Durch die Baumstruktur werden die Abhängigkeiten zwischen den Pipelineabschnitten verdeutlicht. Mithilfe der farbigen Kreise können die Status der Builds eingesehen werden. Ein Problem bei dieser Darstellung ist, dass ein großer Teil der Fläche ungenutzt bleibt. Weiterhin steht die wichtige Information des Status nicht im Vordergrund. Die Darstellung in Abbildung 8 wurde mit der Javascript-Bibliothek D3.js programmiert. Die Kinder der einzelnen Elemente sind ein- und ausklappbar. Dies vermeidet eine Überflutung von Informationen.

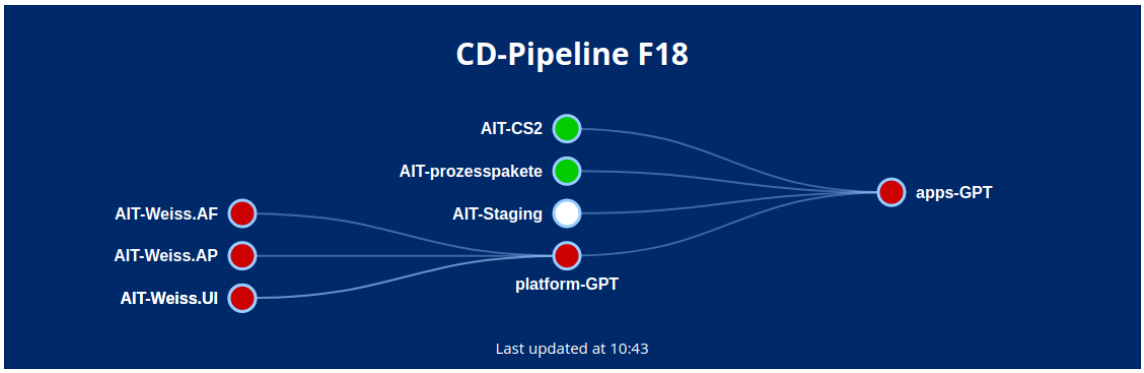

Abbildung 8: Baumdarstellung der Continuous Delivery Pipeline

11.2 Neugestaltung zu einem Gridsystem

Um die Fläche des Widgets besser zu nutzen, ist die Idee für ein Gridsystem entstanden. Der Status wird durch die Farbe des Hintergrundes der einzelnen Gridelemente gezeigt. In Abbildung 9 wird ein Mockup für die Gestaltung der Pipeline gezeigt. Die Positionen und Größen der Elemente werden aus der Baumstruktur ermittelt.

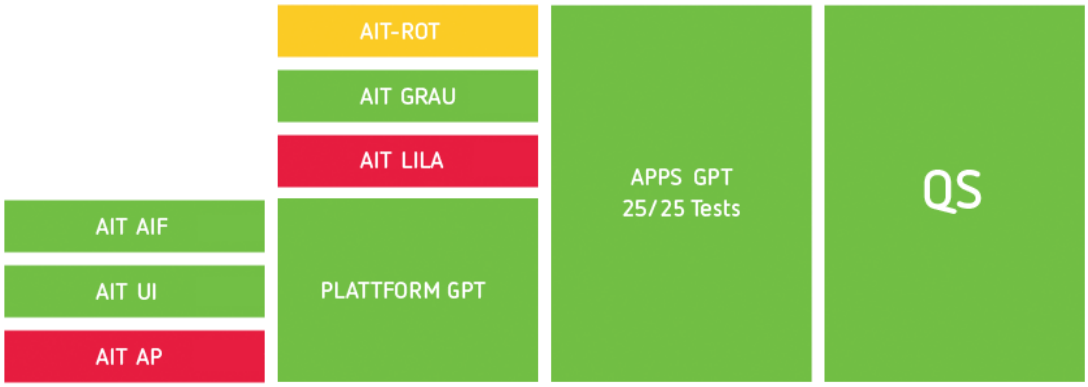

Abbildung 9: Mockup einer Griddarstellung der Continuous Delivery Pipeline

Im Folgenden wurden die Farben durch zusätzliche Icons ergänzt, sodass auch Menschen mit einer Farbschwäche das Dashboard nutzen können. Die Icons sind synchron mit denen, die Jenkins benutzt. In Anforderung [F03] steht, dass der Ablauf der Continuous Delivery Pipeline nachvollzogen werden soll. Aus diesem Grund werden die Übergänge der Pipelineabschnitte durch Pfeile dargestellt. Die Struktur ist von links nach rechts aufgebaut.

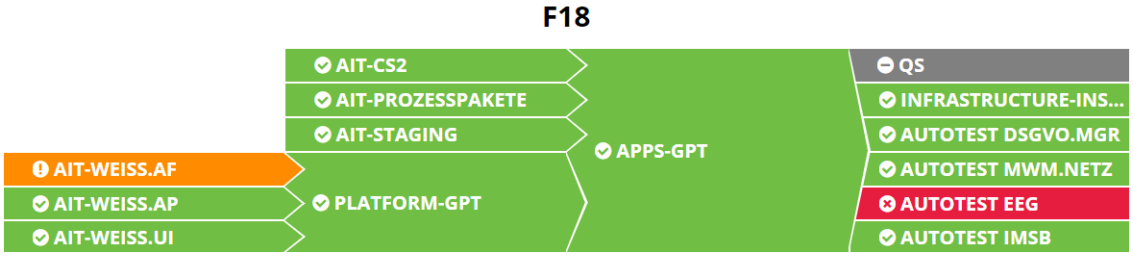

Abbildung 10: Visualisierung CD-Pipeline durch ein Gridsystem

12 Visualisierung der Durchlaufzeit

Das Widget aus Abbildung 11 zeigt die Durchlaufzeit der Pipeline in Stunden. Die Daten werden in einem bestimmten Zeitraum aktualisiert. Zusätzlich wird die Veränderung zum vorherigen Datensatz mit einer Prozentzahl wiedergegeben. Der Pfeil nach oben oder unten dient als Unterstützung zur Anzeige der Veränderung. Weiterhin wird ein Sparkline Graph benutzt. Dieser beinhaltet weder Achsen noch Beschriftungen. Dadurch wird der Fokus auf die Entwicklung der Werte gelegt. Mit der Kombination dieser Darstellungen kann ein Anwender schnelle kompakte Informationen erlangen.

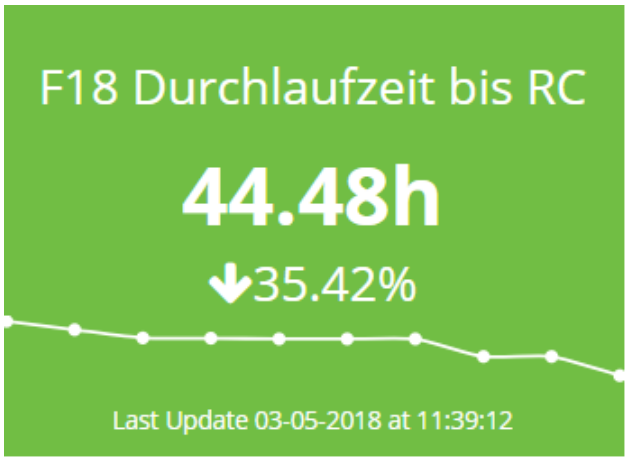

Abbildung 11: Durchlaufzeit - Als Beispiel mit Zufallsdaten

Die Darstellung der Durchlaufzeit hat zwei Ziele. Zum einen soll der Betrachter einsehen können, wie lange eine neue Komponentenversion voraussichtlich braucht, um die CD-Pipeline zu durchlaufen. Dies ermöglicht genauere Abschätzungen für die Auslieferung von Komponenten. Das zweite Ziel ist eine Verbesserung der CD-Pipeline Architektur. Mithilfe von Vergleichen zu vorherigen Durchlaufzeiten kann die Konsistenz der CD-Pipeline überprüft werden. Wenn Builds häufiger fehlschlagen, wird es zu einer negativen Veränderung der Durchlaufzeit kommen.

Teil VI

Implementierung

13 Datensammeln durch den Artifact Repository Client

In Kapitel 4.2 wurde im Zusammenhang von Qualitätsstufen das Artifact Repository erwähnt. Um Komponenten von einer Qualitätsstufe in die Nächste zu befördern, wurde vom Unternehmen eine Anwendung mit dem Namen Artifact Repository Client entwickelt. Der Client kann sowohl manuell als auch automatisch am Ende eines Builds gestartet werden. Zum Sammeln von Telemetrie findet eine Erweiterung des Artifact Repository Clients statt.

Die Daten werden über eine Klasse namens *TelemetryWriter* an die Telemetrie API (Kapitel 10.2) als POST-Request versendet. Der *TelemetryWriter* wird direkt nach erfolgreichem Verschieben einer Komponente ausgelöst, sodass die Zeitstempel möglichst genau sind.

Sollte die Übermittlung der Telemetrie fehlschlagen, muss gewährleistet sein, dass der Artifact Repository Client nicht abbricht, da eine Komponente trotzdem befördert werden soll. Diese Anforderung wird mit folgendem Unittest überprüft:

```
var mockCreator = CreateWebRequestMockWithException();

using (new HttpWebRequestWrapperSession(mockCreator.Object))
{
Assert.That(() => telemetryWriter.UpdateTelemetry(artifactInformation,
    developmentLine, toQuality), Throws.Nothing);

loggerMock.Verify(logger => logger.Error(It.IsAny<WebException>()));
}
```

In der *CreateWebRequestMockWithException* Methode wird ein Mock-Objekt für einen *HttpWebRequest* erstellt. Mithilfe des Mock-Objekts wird das Auftreten einer Web-Exception simuliert. Der *TelemetryWriter* sollte in diesem Testfall den Ausnahmefall behandeln und keinen Absturz des Programms hervorrufen.

14 Anwendung zur Berechnung der Durchlaufzeit (pipeline-cycletime)

Im Folgenden wird die Implementation zur Berechnung der Durchlaufzeit beschrieben. Die Erkenntnisse aus Kapitel 10.3 bilden die Grundlage der Umsetzung. Die Berechnung erfolgt in einer dotnet Konsolenapplikation, die einmal täglich, zeitgesteuert ausgeführt wird. Die Applikation hat den Namen *pipeline-cycletime*.

Das Artifact Repository befindet sich in einem Team Foundation Version Control (TFVC). TFVC stellt eine REST API zur Verfügung, mit der Informationen zu den verwalteten Dateien abgerufen werden können.

pipeline-cycletime ermittelt zunächst über die TFVC API die aktuellen Komponentenversionen aus dem Artifact Repository. Die Versionsinformationen erlauben eine Abfrage der gesammelten Telemetriedaten aus Elasticsearch. Die Abfrage erfolgt über die Telemetrie API (Kapitel 10.2).

In den Daten sind die Zeiten gespeichert, an denen neue Qualitätsstufen erreicht wurden. Die Durchlaufzeit kann nur für Komponenten berechnet werden, die Daten zu den Stufen CI und RC besitzen, also die Pipeline einmal durchlaufen haben. Die Differenz zwischen CI und RC wird in Sekunden zurückgegeben. Die Methode, die die Durchlaufzeiten aus der Telemetrie extrahiert, ist folgendermaßen aufgebaut:

```
var cycletimes = new Dictionary<string, int> { };

foreach (var telemetry in telemetries)
{
if (telemetry.RC != default(DateTime) && telemetry.CI != default(DateTime))
{
var cycletime = telemetry.RC.Subtract(telemetry.CI).TotalSeconds;
cycletimes.Add(telemetry.Id, Convert.ToInt32(cycletime));
logger.Info($"Component {telemetry.Id} has a cycletime of {cycletime}");
}
}
return cycletimes;
```

Als Ergebnis liefert die Methode ein Dictionary zurück. Der **Schlüssel** wird durch die Id und der **Wert** durch die Durchlaufzeit der Komponente gebildet. Für das Dashboard wird im Folgenden der Median aus den Zeiten gebildet. Der Median ist definiert als der

zentrale/mittlere Wert einer sortierten Zahlenfolge. Bei einer geraden Menge von Zahlen wird der Durchschnitt der beiden mittleren Werte ermittelt.

```
var halfIndex = cycletimesCount / 2;
var sortedNumbers = cycletimes.OrderBy(n => n);

if ((cycletimesCount % 2) == 0)
{
median = ((sortedNumbers.ElementAt(halfIndex) +
    sortedNumbers.ElementAt(halfIndex - 1)) / 2);
}
else
{
median = sortedNumbers.ElementAt(halfIndex);
}
```

Im Folgenden werden die Ergebnisse über die Telemetrie API an Elasticsearch gesendet.

15 Erstellung des Dashboard

In diesem Kapitel wird die Implementierung des Dashboards beschrieben. Zur Umsetzung wird das Framework Smashing verwendet. Smashing besitzt eine feste Grundstruktur. Für Entwickler sind die wichtigsten Informationen in den Ordnern:

- **Jobs:** Abrufen und Verarbeiten der Daten.
- **Widgets:** Visualisierung der Daten.
- **Dashboards:** Konfiguration der Layouts.

15.1 Abrufen und Verarbeiten der Daten

Das Abrufen und Verarbeiten von Daten wird in Ruby Skripten realisiert. Als Auslöser für Abfragen wird rufus-scheduler benutzt. Der Scheduler führt alle zwölf Sekunden einen neuen Job aus. Zusätzlich ist für den Scheduler die Option *overlap* deaktiviert. Das führt dazu, dass Jobs mit gleicher Funktionalität nicht parallel laufen.

Smashing benutzt Server Sent Events (SSE), um die Dashboards mit aktuellen Daten zu versorgen. Bei SSE findet fast keine bidirektionale Kommunikation zwischen Client und Server statt. Nachdem eine Verbindung zum Client errichtet wurde, schickt der Server kontinuierlich Daten in Form von event-streams. Jeder Client kann mit einem *Event-Listener* diese Nachrichten abonnieren. Die Handhabung der Databindings erfolgt durch das Framework *batman.js*. Durch die Zuordnung von Events können auch unterschiedliche Dashboards die gleichen Datensätze abbilden. Dies ist eine Grundvoraussetzung für Anforderung [F06].

15.1.1 Abfrage von Jenkins Buildinformationen

Jenkins stellt mehrere REST APIs zur Verfügung. Für die Abfrage der Buildinformationen wird die JSON API verwendet. Jenkins bietet zusätzlich den Abfrage-Parameter ”tree”. Mit diesem lassen sich Suchen durch Jenkins auf die benötigten Elemente einschränken. Die URL, um Informationen zu Jenkins-Jobs abzufragen, sieht folgendermaßen aus: `http://jenkins:8080/api/json?pretty=true\&tree=jobs[name,color,url]`. Damit werden jeweils Name, Farbe und URL von allen Jobs ausgegeben. Die Farbe repräsentiert den Status des letzten Builds.

Die TFS- und Jenkins Abfragen erfolgen zusammen in einem Ruby-Skript.

15.1.2 Abfrage von TFS Buildinformationen

Die Benutzung der TFS API benötigt eine zusätzliche Authentifizierung mittels NTLM. Aus diesem Grund wird die Ruby-Bibliothek HTTPI, die auf ruby-ntlm aufbaut, verwendet.

Der TFS wird nur für CI-Builds benötigt. Über eine Input-Datei werden die Namen der Builddefinitionen bestimmt, die in den Dashboardlayouts Verwendung finden. Den Status des letzten Builds einer Definition anzuzeigen, ist etwas umständlicher als bei Jenkins. Zunächst muss mittels der angegebenen Namen die Definitions-ID erfragt werden. Die URI dazu ist `"#{TFS_URI}/build/definitions?name=#{definitionName}\&api-version=2.0"`. Die geschweiften Klammern verweisen in Ruby auf Parameternamen. In der zurückgegebenen Json-Datei ist die Id der Definition enthalten. Durch eine weitere Abfrage wird der Wert des letzten abgeschlossenen Builds ermittelt: `"#{TFS_URI}/build/builds?definitions=#{id}\&statusFilter=completed\&\$top=1\&api-version=2.0`.

15.1.3 Verarbeitung der Buildinformationen

Nach erfolgreichem Abrufen besitzen die Buildinformationen noch keine Abhängigkeiten zueinander. Um den Aufbau der CD-Pipeline widerzuspiegeln, werden die Daten in eine Baumstruktur gebracht. Für die Ordnung der Pipelinesegmente gibt es fest verankerte Informationen. Zum Beispiel ist das Wurzelelement immer die QS-Umgebung. Weiterhin werden durch eine Konfigurationsdatei Elemente für das Grid mitgegeben.

In Abbildung 12 wird der Aufbau der Klasse für eine Buildinformation gezeigt. Der Status enthält das Resultat des letzten abgeschlossenen Builds. Die URL wird für die Weiterleitungen in Kapitel 9.2 benötigt. Durch den *parent* Parameter wird später die Baumstruktur gebildet. Manche Elemente werden bewusst herausgefiltert und setzen den Parameter *isInTree* auf *false*.

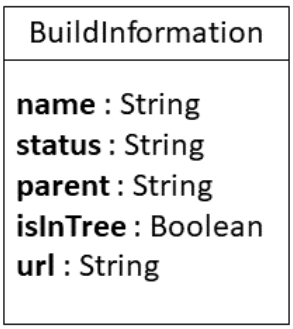

Abbildung 12: Klassendiagramm für Buildinformationen

Als letzten Schritt werden die Daten über Events an die Clients verteilt, welche die Informationen wiederum in eine Griddarstellung umwandeln.

15.1.4 Abfrage der Durchlaufzeit

Die Abfrage der Durchlaufzeit erfolgt über die in Kapitel 10.2 beschriebene Web API der Pipeline-Telemetrie. Die Anforderung ist wieder in einer Rubydatei realisiert. Über einen Scheduler werden im Abstand von 15 Sekunden neue Abfragen durchgeführt. Für jede Entwicklungslinie, die auf den Dashboards dargestellt wird, existiert eine mittlere Durchlaufzeit. Die Abfrage erfolgt über `"#{webapi_uri}/api/cycletime/#{iteration}"`.

15.2 Generierung einer Gridstruktur zur Darstellung der Continuous-Delivery-Pipeline

Zur Visualisierung der CD-Pipeline werden clientseitig dynamische Gridsysteme generiert. Als Grundlage werden die Standardfunktionen eines CSS Grid Layouts verwendet.

Zunächst bekommen die Clients vom Server Daten in Form einer Baumstruktur. Der Aufbau variiert durch die unterschiedlichen Konfigurationen der Dashboards. Um eine Generierung des Grids zu ermöglichen, werden Positionen und Größe für die Elemente benötigt. Im Folgenden wird die Ermittlung derselben beschrieben.

Alle Elemente besitzen die gleiche Breite. Die Höhe ist variabel und muss berechnet werden. Die Blätter des Baumes bilden die kleinste Einheit des Grids (s. Abbildung 13). Somit bekommen sie eine relative Höhe von 1. Alle anderen Elemente bilden die Summe der Höhen ihrer Kinderelemente. Eine Ausnahme bildet die QS-Umgebung, da sie im späteren Verlauf nochmals in mehrere Builds aufgeilt wird.

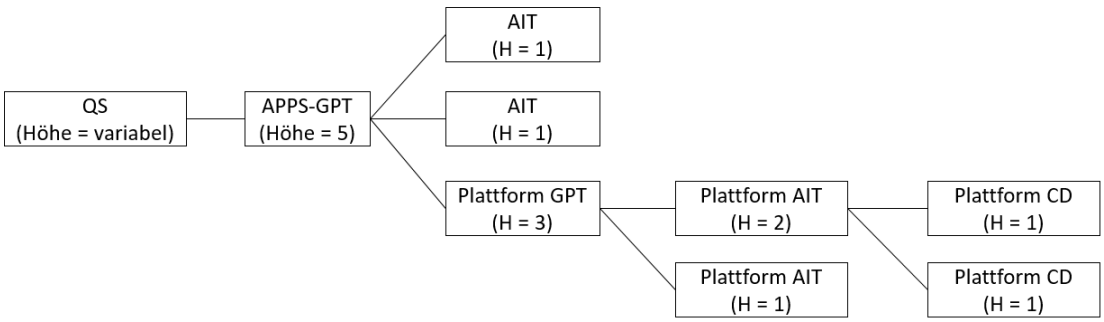

Abbildung 13: Baumstruktur der Pipeline-Elemente

Die Position der Elemente wird durch deren Baumtiefe bestimmt. Damit das Grid von

links nach rechts aufgebaut wird, ist die Position definiert durch: Höhe des Baums - Tiefe des Knotens. Als Beispiel besitzen in Abbildung 13 die QS-Elemente die Position 5 und die Plattform AIT-Segmente die Position 2.

15.3 Konfigurationsmöglichkeiten

In der Anforderung [F05] wird die Konfiguration eines Dashboards gefordert. Diese Möglichkeit ist durch die Layout Dateien im Ordner **Dashboards** gegeben. Es können unterschiedliche Widgets eingerichtet werden, die über IDs ihre Datensätze bekommen. Um genauere Einstellungen an der CD-Pipeline Darstellung vornehmen zu können, müssen in einer zusätzlichen Datei Informationen zu Builds mitgeliefert werden. Zur Einstellung der Widgetpositionen kann auch das Drag-and-Drop-System von Smashing benutzt werden.

```
<% content_for :title do %>Pipeline Dashboard<% end %>
<div class="gridster">
  <ul>
    <li data-row="1" data-col="1" data-sizex="3" data-sizey="1">
      <div data-id="pipelinePlatformS18" data-iteration="S18" data-view="Cdpipeline" data-title="S18" data-url=""></div>
    </li>

    <li data-row="1" data-col="4" data-sizex="1" data-sizey="1">
      <div data-id="cycletimeS18" data-view="Cycletime" data-title="S18 Durchlaufzeit bis RC" data-graphtype="lineplot" data-suffix="h"></div>
    </li>

    <li data-row="2" data-col="1" data-sizex="3" data-sizey="1">
      <div data-id="pipelinePlatformF18" data-iteration="F18" data-view="Cdpipeline" data-title="F18"></div>
    </li>

    <li data-row="2" data-col="4" data-sizex="1" data-sizey="1">
      <div data-id="cycletimeF18" data-view="Cycletime" data-title="F18 Durchlaufzeit bis RC" data-graphtype="lineplot" data-suffix="h"></div>
    </li>

    <li data-row="3" data-col="1" data-sizex="3" data-sizey="1">
      <div data-id="pipelinePlatformW18" data-iteration="W18" data-view="Cdpipeline" data-title="W18" data-url=""></div>
    </li>
  </ul>
</div>
```

Abbildung 14: Beispiel für eine Layout Datei von Smashing

Die Abbildung 14 zeigt den Aufbau einer Layout Datei in Smashing. Die Datei hat die Endung '.erb', da sie HTML-Text mit Ruby Code beinhaltet.

Alle Elemente sind einem *div* der Klasse *gridster* untergeordnet. Gridster wird von Smashing zur Positionierung der Widgets verwendet. Jedes Listenelement bildet ein Widget ab. Durch Parameter wie *data-col* oder *data-sizex* werden Position und Größe bestimmt. Mit der *data-id* wird festgelegt, welcher Datensatz für das Widget benutzt wird. Die Daten müssen dabei der jeweiligen Vorlage entsprechen. Die *data-view* bestimmt die Art des Widgets.

16 Integration von Detailinformationen in eine Webanwendung (Progress)

In Kapitel 3 wurde vorgestellt, wie und welche Telemetrie in Elasticsearch gespeichert werden. Genauer wurde die Berechnung der Durchlaufzeit beschrieben. Die Informationen zu den einzelnen Pipelineabschnitten sind damit jedoch noch ungenutzt (s. Abbildung 5).

Im Unternehmen existiert eine Webanwendung mit dem Namen ”Progress”. Mit der Anwendung lässt sich der Änderungsverlauf von einzelnen Komponenten nachvollziehen lässt. Das bedeutet, dass alle User Stories, die seit dem Veröffentlichen der letzten Version entstanden sind, aufgelistet werden. Mit den neu gesammelten Daten, kann diese Applikation um eine Pipeline Historie erweitert werden. Die Historie verdeutlicht den Durchlauf einer Komponenten in der CD-Pipeline. Dadurch kann nachvollzogen werden, welche Builds abgeschlossen und welche Qualitätsstufen erreicht worden sind. Das Abrufen der Daten erfolgt über die Telemtrie API (10.2).

Abbildung 15 zeigt eine Visualisierung der vorhandenen Daten. Die Blau hervorgehobenen Stellen sind Qualitätsstufen. Zusätzlich wird bei Komponenten, die die CD-Pipeline erfolgreich abgeschlossen haben, die Durchlaufzeit angezeigt.

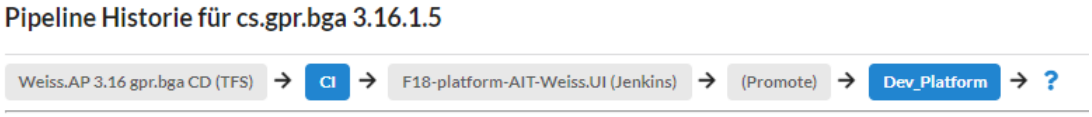

Abbildung 15: Visualisierung der Pipeline Historie einer Komponenten

Teil VII

Ergebnisse und Fazit

Im Folgenden werden die Ergebnisse der Arbeit zusammengefasst, ein Fazit gebildet und mögliche Erweiterungen vorgestellt.

17 Zusammenfassung

In dieser Arbeit wurde die Entwicklung eines Monitoringsystem beschrieben, welches die Vorgänge einer Continuous Delivery Pipeline veranschaulicht und unterstützt.

In den ersten Kapiteln fand eine Beschreibung von Continuous Delivery statt, sowie eine Definition der Durchlaufzeit. Als nächstes wurde ein Pflichtenheft gebildete, um die Anforderungen des Projektes aufzuschreiben. Mithilfe des Pflichtenheftes wurde eine Auswahl für die Dashboard Software getroffen. Die Entscheidung lag beim Framework Smashing.

Im Kapitel Architektur wurde vorgestellt, wie Smashing in die Umgebung von Schleupen eingebaut werden kann. Es wurden sowohl Schnittstellen als auch Weiterleitungen bestimmt. Es folgte eine Architektur der Telemetrie, für die Elasticsearch benutzt wurde. Im Folgenden wurde begründet warum für die Zusammenfassung der Durchlaufzeiten der Median benutzt wurde.

Das Kapitel Design stellt den Verlauf der unterschiedlichen Designansätze dar. Weiterhin wird das Widget der Durchlaufzeit beschrieben. Als nächstes wurden die Implementierungen erläutert. Diese beinhalteten zum einen die Dashboard Entwicklung. Zum anderen eine Integration in eine Webanwendung mit Detailinformationen.

18 Fazit

Das entwickelte Dashboard wird im Unternehmen funktionsfähig eingesetzt. Es kann sowohl auf Fernseh- als auch auf Computermonitoren benutzt werden.

19 Erweiterungsmöglichkeiten

Im Unternehmen wurde das Monitoringsystem parallel zur Einführung von Continuous Delivery entwickelt. Die Umstellung zur Continuous Delivery Pipeline fand teamweise statt. Dies führte dazu, dass für hinzukommende Umgebungen neue Dashboards erstellt wurden. Am Ende der Umstellung sollte jedes Team von Schleupen ein eigenes Dashboard besitzen.

Eine Möglichkeit zur Weiterentwicklung besteht durch das Hinzufügen von Metriken. In dieser Arbeit wurde die Durchlaufzeit detailliert besprochen. In zukünftigen Projekten könnte die Bedeutung von anderen Telemetriedaten analysiert werden. Wichtig dabei ist, herauszuarbeiten, welches Ziele mit der Visualisierung erreicht werden.

Literatur

[1] Elasticsearch. https://www.elastic.co/products/elasticsearch.

[2] Raphael Albino. Power of the metrics: Don't use average to forecast deadlines. http://blog.plataformatec.com.br/2016/01/power-of-the-metrics-dont-use-average-to-forecast-deadlines/. Retrieved 27.04.2018.

[3] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum, Akad. Verl., 2009.

[4] Stephen Few. Dashboard confusion, perceptual edge. 2004.

[5] Stephen Few. Information dashboard design. 2006.

[6] Jez Humble. Continuous delivery. https://continuousdelivery.com, 2010. Retrieved 29.01.2018.

[7] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.

[8] Sebastian Jancke. Continuous delivery konzept. Schleupen Intranet Wiki, 2018.

[9] Gene Kim, Patrick Debois, John Willis, and Jez Humble. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[10] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591, 1965.

[11] César Souza, Andrew Kirillov, Marcos Diego Catalano, and Accord.NET contributors. The accord.net framework. http://accord-framework.net, 2014.

[12] Miroslaw Staron, Kent Niesel, and Wilhelm Meding. Selecting the right visualization of indicators and measures – dashboard selection model. In *Software Measurement*, pages 130–143, Cham, 2015. Springer International Publishing.

Anhang

Systemqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Funktionalität				
Angemessenheit			X	
Genauigkeit	X			
Interoperabilität		X		
Sicherheit			X	
Konformität			X	
Zuverlässigkeit				
Reife	X			
Fehlertoleranz		X		
Wiederherstellbarkeit			X	
Konformität			X	
Benutzbarkeit				
Verständlichkeit	X			
Erlernbarkeit			X	
Bedienbarkeit		X		
Attraktivität	X			
Konformität			X	
Effizienz				
Zeitverhalten	X			
Verbrauchsverhalten			X	
Konformität			X	
Wartbarkeit				
Analysierbarkeit		X		
Änderbarkeit		X		
Stabilität		X		
Testbarkeit			X	
Konformität			X	
Portabilität				X

Tabelle 1: Bewertung von Qualitätsanforderungen nach ISO 9126

Selbstständigkeitserklärung

Hiermit erkläre ich, Joshua Siegemund, dass ich die hier vorliegende Arbeit selbstständig und ohne unerlaubte Hilfsmittel angefertigt habe. Informationen, die anderen Werken oder Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich kenntlich gemacht und mit exakter Quellenangabe versehen. Sätze oder Satzteile, die wörtlich übernommen wurden, wurden als Zitate gekennzeichnet. Die hier vorliegende Arbeit wurde noch an keiner anderen Stelle zur Prüfung vorgelegt und weder ganz noch in Auszügen veröffentlicht. Bis zur Veröffentlichung der Ergebnisse durch den Prüfungsausschuss werde ich eine Kopie dieser Studienarbeit aufbewahren und wenn nötig zugänglich machen.

<Ort, Datum><Unterschrift>