[Achtung: Verwenden Sie einen Sperrvermerk nur in sehr gut begründeten Fällen!]

[evtl. Sperrvermerk]

Auf Wunsch der Firma [FIRMA] ist die vorliegende Arbeit bis zum [DATUM] für die öffentliche Nutzung zu sperren.

Veröffentlichung, Vervielfältigung und Einsichtnahme sind ohne ausdrückliche Genehmigung der oben genannten Firma und der/dem Verfasser/in nicht gestattet. Der Titel der Arbeit sowie das Kurzreferat/Abstract dürfen jedoch veröffentlicht werden.

Dornbirn, Unterschrift der Verfasserin/des Verfassers

Firmenstempel

FH Vorarlberg
University of Applied Sciences

Qualitätsmanagement in Scrum-Teams

Untertitel

Masterarbeit
zur Erlangung des akademischen Grades

Master of Science (MSc)

Fachhochschule Vorarlberg
Informatik

Betreut von
Prof. Dr. Michael Felderer

Vorgelegt von
Daniel Grießer
Dornbirn, Juli 2018

[evtl. Widmung]

[Text der Widmung]

Kurzreferat

[Deutscher Titel Ihrer Arbeit]

[Text des Kurzreferats]

Abstract

[English Title of your thesis]

[text of the abstract]

[evtl. Vorwort]

[Text des Vorworts]

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

LOC Lines of Code
CLOC Changed Lines of Code
VCS Version Control System
PTS Project Tracking System
DoD Definition of Done
CI Continuous Integration
CD Continuous Delivery
APM Application Performance Monitoring
BI Business Intelligence
IEEE Institute of Electrical and Electronics Engineers
MTTF Mean Time to Failure
MTTR Mean Time to Release
GQM Goal Question Metric

1 Einleitung

(Mit Qualitäts-Analysetools, wie z.B. SonarQube[1], können ganze Softwaresysteme kontinuierlichen Qualitätstests unterzogen werden. Durch die ermittelten Kennzahlen können Aussagen zur Qualität des gesamten Systems, über einzelne Komponenten, bis hin zu einer einzelnen Quellcode-Datei getroffen werden.

Auch Scrum[2] wird als agiles Vorgehensmodell in der Softwareentwicklung immer beliebter. Dabei wird bei mehreren Teams, die auf vielen Systeme arbeiten, auf 2 Arten von Scrum Teams zurückgegriffen: Feature- oder Komponenten-Teams.

Diese Arbeit beschäftigt sich mit dem Qualitätsmanagement in Scrum-Teams. Das bedeutet, dass Kennzahlen zu Qualitätsmerkmalen nicht auf System-, sondern auf Komponentenebene gesammelt und aggregiert werden, um für jedes Team eine individuelle Sicht auf das Qualitätsmanagement bereitzustellen. Um das zu ermöglichen, wird erst eine Vorgehensweise zur Ermittlung von relevanten Kennzahlen entwickelt und diese an einer Beispiel-Organisation angewendet. Zur Sammlung, Auswertung und Darstellung dieser Kennzahlen wird eine Software entwickelt, die in eine bestehende Umgebung integriert werden kann.)

. . . schreibe ich ganz am Schluss neu

1.1 Problemstellung

[1] *Continuous Code Quality | SonarQube.* URL: `https://www.sonarqube.org/` (besucht am 05. 01. 2018).

[2] *Scrum.* URL: `http://www.scrum.org` (besucht am 05. 01. 2018).

2 Situationsanalyse

2.1 Agile Softwareentwicklung

Diese Arbeit dreht sich um agile Teams, deshalb ist es essentiell, zu verstehen, was der Gedanke hinter dem agilen Entwicklungsansatz ist. Seinen Ursprung hat das Ganze, als sich 2001 ein paar schlaue Köpfe zusammengeschlossen haben und das sogenannte agile Manifest, sowie die agilen Prinzipien aufgestellt haben. Ziel war es, eine Alternative zu den bisherigen, schwergewichtigen und von Dokumentation getriebenen Softwareentwicklungs-Methodologien zu finden.

2.1.1 Agiles Manifest

Das agile Manifest ist er Grundbaustein aller agilen Vorgehensmodelle:

> Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
>
> Individuen und Interaktionen mehr als Prozesse und Werkzeuge
> Funktionierende Software mehr als umfassende Dokumentation
> Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlungen
> Reagieren auf Veränderung mehr als das Befolgen eines Plans
>
> Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.

Manifest für Agile Softwareentwicklung. URL: `http://agilemanifesto.org/iso/de/manifesto.html` (besucht am 16. 03. 2018)

2.1.2 Agile Prinzipien

Die agile Softwareentwicklung folgt diesen zwölf Prinzipien:

> Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
>
> Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.

Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.

Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.

Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.

Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.

Funktionierende Software ist das wichtigste Fortschrittsmaß.

Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.

Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.

Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.

Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Prinzipien hinter dem Agilen Manifest. URL: `http://agilemanifesto.org/iso/de/principles.html` (besucht am 16. 03. 2018)

2.2 Scrum (Quellenangabe!)

Das Scrum Framework ist eine solche agile Softwareentwicklungs-Methodologie. Scrum basiert auf Empirismus, also der Theorie, dass Wissen aus Erfahrung erlangt wird und Entscheidungen auf Basis dieses Wissens getroffen werden. Die drei Grundsäulen einer solchen empirischen Prozesskontrolle sind:

Transparenz
: Signifikante Aspekte des Prozesses müssen für alle sichtbar sein.

Inspektion
: Artefakte müssen regelmäßig inspiziert werden, aber dieser Vorgang darf der Arbeit selbst nicht im Weg stehen.

Adaption
: Weicht ein oder mehrere Aspekte eines Prozesses von seinen akzeptablen Limits ab, muss dieser so früh wie möglich angepasst werden.

Das Scrum Framework (Abbildung 2.1) besteht aus drei Rollen, fünf Ereignissen und drei Artefakten.

- **Rollen**
  - **Development Team**: Selbstorganisiertes Team, das am Produkt arbeitet.
  - **Scrum Mater**: Verantwortlich dafür, sicherzustellen, dass Scrum verstanden und gelebt wird.
  - **Product Owner**: Verantwortlich den Wert des Produktes und die Arbeit des Development Teams zu maximieren.
- **Ereignisse**
  - **Sprint**: Ist das Herz von Scrum: eine Timebox von 2 bis 4 Wochen, in dem ein fertiges, verwendbares und potentiell releasebares Produkt-Inkrement entwickelt wird.
  - **Sprint Planning**: Planung eines Sprints. Hier commited sich das Scrum Team, eine gewisse Anzahl an Aufgaben im kommenden Sprint abzuarbeiten.
  - **Daily Scrum**: Tägliches, zeitlich begrenztes Meeting, bei dem von jedem Teammitglied folgende drei Fragen beantwortet werden:
    1. Was habe ich gemacht?
    2. Was werde ich machen?
    3. Was behindert mich bei meiner Arbeit?
  - **Sprint Review**: Abschluss eines Sprints. Hier präsentiert das Team dem Product Owner die Ergebnisse des letzten Sprints.
  - **Sprint Retrospective**: Das Team reflektiert den Sprint-Ablauf und ergreift Maßnahmen, um den Prozess weiter zu verbessern.
- **Artefakte**
  - **Product Backlog**: Ist eine Sammlung von möglichen Aufgaben für das Team am Produkt. Sollte einen Ausblick auf die zukünftige Entwicklung des Produktes geben. Oben im Product Backlog befinden sich die bereits fein geplanten Aufgaben, weiter unten die groben.
  - **Sprint Backlog**: Entspricht den Aufgaben, die vom Team in den Sprint genommen und dem Product Owner zugesagt wurden.
  - **Increment**: Entsteht am Ende eines jeden Sprints und ist eine lauffähige Version des Produkts, die releasefähig ist.

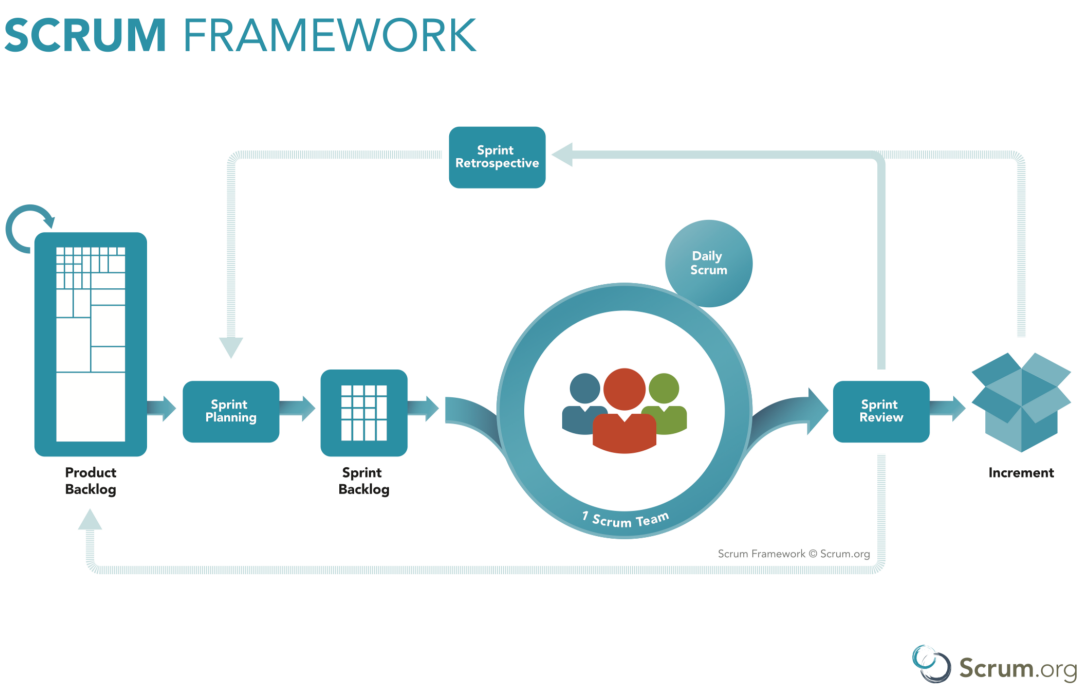


Abbildung 2.1: Scrum Framework[3]

2.2.1 Scrum in mehreren Teams[4]

Scrum beschreibt eine agile Vorgehensweise für ein Team (ein Team entwickelt ein Produkt). In der Realität existieren aber oft mehrere Teams und/oder mehrere Produkte. Dahingehend muss die Organisation der unterschiedlichen Scrum Teams individuell angepasst werden. Für die Trennung der Teams gibt es unterschiedliche Ansätze:

Trennung nach Organisationseinheiten
: Die Teams werden entlang der Abteilungsstruktur einer Organisation getrennt. Aus Scrum-Sicht macht das nicht immer Sinn, da bei der Umsetzung eines Features Abhängigkeiten zu anderen Teams bestehen (keine cross-funktionalen Teams).

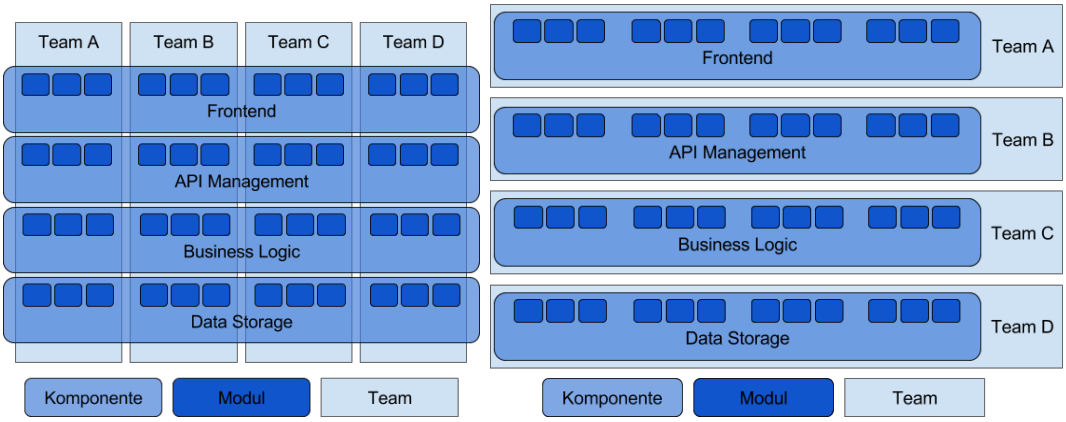
Trennung nach Komponenten (Komponenten-Teams)
: Die technischen Komponenten werden den Teams zugeteilt, was ebenfalls zu Abhängigkeiten zu anderen Teams führt und eine gute Abstimmung zwischen den Teams voraussetzt.

Trennung nach fachlichen Themen (Feature-Teams)
: Jedes Team entwickelt, unabhängig von den anderen Teams, eine fachliche Komponente. Diese Variante erfüllt die Forderung des Scrum Frameworks nach cross-

[3] *The Scrum Framework Poster | Scrum.org*. URL: `https://www.scrum.org/resources/scrum-framework-poster` (besucht am 01. 04. 2018).

[4] vgl. Rolf Dräther, Holger Koschek und Carsten Sahling. *Scrum: kurz & gut*. 1. Auflage. O'Reillys Taschenbibliothek. Beijing Cambridge Farnham Köln Sebastopol, Tokyo: O'Reilly, 2013. ISBN: 978-3-86899-833-7, S.172ff.

funktionalen Teams, weshalb bei dieser Form die Abstimmung zwischen den Teams am geringsten ist.



(a) Feature-Teams (b) Komponenten-Teams

Abbildung 2.2: Scrum Teams

In allen Varianten existieren aber pro Team unterschiedliche Software-Module und (agile) Prozesse, die unabhängig voneinander die Team-Qualität als gesamtes bestimmen.

2.3 Software-Qualität[5]

Eine mögliche Definition von Software-Qualität findet sich in der DIN-ISO-Norm 9126:

> "Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen."

Wie aus dieser Definition schon erkennbar ist, gibt es viele unterschiedliche Kriterien, um die Qualität von Software zu bewerten. Einige wesentliche Merkmale, um die Qualität von Software bewerten zu können, lassen sich in kunden- und herstellerorientierte Merkmale unterteilen:

Kundenorientierte Merkmale

Nach außen hin sichtbare Merkmale, die sich auf den kurzfristigen Erfolg der Software auswirken, da sie die Kaufentscheidung möglicher Kunden beeinflussen.

Funktionalität (Functionality, Capability)

Beschreibt die Umsetzung der funktionalen Anforderungen. Fehler sind hier häufig Implementierungsfehler (sogenannte Bugs), welche durch Qualitätssicherung bereits in der Entwicklung entdeckt oder vermieden werden können.

Laufzeit (Performance)

Beschreibt die Umsetzung der Laufzeitanforderungen. Besonderes Augenmerk muss in Echtzeitsystemen auf dieses Merkmal gelegt werden.

Zuverlässigkeit (Reliability)

Eine hohe Zuverlässigkeit ist in kritischen Bereichen, wie z.B. Medizintechnik oder Luftfahrt, unabdingbar. Erreicht werden kann diese aber nur durch die Optimierung einer Reihe anderer Kriterien.

Benutzbarkeit (Usability)

Betrifft alle Eigenschaften eines Systems, die mit der Benutzer-Interaktion in Berührung kommen.

Herstellerorientierte Merkmale

Sind die inneren Merkmale, die sich auf den langfristigen Erfolg der Software auswirken und somit als Investition in die Zukunft gesehen werden sollten.

Wartbarkeit (Maintainability)

Die Fähigkeit auch nach der Inbetriebnahme noch Änderungen an der Software vorzunehmen. Wird oft vernachlässigt, ist aber essentiell für langlebige Software und ein großer Vorteil gegenüber der Konkurrenz.

Transparenz (Transparency)

Beschreibt, wie die nach außen hin sichtbare Funktionalität intern umgesetzt wurde. Gerade bei alternder Software, kann es zu einer Unordnung kommen, welche auch Software-Entropie (Grad der Unordnung) genannt wird.

[5] vgl. Dirk W. Hoffmann. *Software-Qualität*. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-35699-5 978-3-642-35700-8, Kapitel 1.2.

Übertragbarkeit
: Wird auch Portierbarkeit genannt und beschreibt die Eigenschaft einer Software, in andere Umgebungen übertragen werden zu können (z.B. 32-Bit zu 64-Bit oder Desktop zu Mobile).

Testbarkeit (Testability)
: Testen stellt eine große Herausforderung dar, da oft auf interne Zustände zugegriffen werden muss oder die Komplexität die möglichen Eingangskombinationen vervielfacht. Aber gerade durch Tests können Fehler frühzeitig entdeckt und behoben werden.

Je nach Anwendungsgebiet und den Anforderungen der Software haben die Merkmale unterschiedliche Relevanz und einige können sich auch gegenseitig beeinflussen, wie aus der Korrelationsmatrix in Abbildung 2.1 ersichtlich. Dabei sind die positiv korrelierenden Merkmale mit "+" und die negativ korrelierenden mit "-" gekennzeichnet.

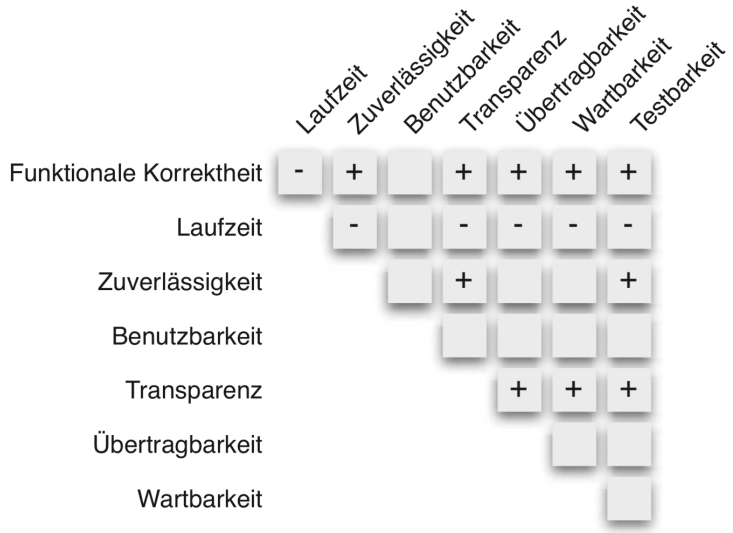


Abbildung 2.3: Korrelationsmatrix Qualitätskriterien[6]

[6]Dirk W. Hoffmann. *Software-Qualität*. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-35699-5 978-3-642-35700-8, S. 11, Abb. 1.3.

2.4 Kennzahlen

Software-Metriken helfen uns dabei, bestimmte (Qualitäts-) Merkmale beziehungsweise Kenngrößen eines Software-Systems systematisch und quantitativ zu erfassen. Ziel ist es dabei, diese oft versteckten Merkmale sichtbar und vergleichbar zu machen. Ein einfaches Beispiel ist die Lines of Code (LOC)-Metrik, die die gesamte Anzahl an Zeilen Code darstellt und als grobes Maß für die Komplexität verwendet werden kann.

Abbildung 2.4: Softwareentwicklungsprozess

Im Entwicklungsprozess werden in den unterschiedlichen Systemen und Prozessschritten Daten erzeugt, die als Kennzahlen oder direkt als Metriken genutzt werden können. Abbildung 2.4 zeigt die einzelnen Schritte und Systeme im Entwicklungsprozess.

2.4.1 Versionsverwaltung[7]

Das VCS befindet sich nah an der Arbeit der Entwickler, da hier der Quellcode des Produkts verwaltet wird. Daher können hier Daten darüber gesammelt werden, wie viel gearbeitet und auch wie viel zusammengearbeitet wird. Um bestmögliche Daten zu bekommen, sollten verteilte Versionskontrollsysteme wie Git verwendet und mit Pull Requests gearbeitet werden.

Changed Lines of Code (CLOC)
Anzahl der geänderten Code Zeilen.

CLOC pro Entwickler
Anzahl der geänderten Zeilen im Quellcode pro Entwickler.

Commits
Gesamtzahl an Commits in einem bestimmten Zeitraum.

Commits pro Entwickler
Gesamtzahl an Commits in einem bestimmten Zeitraum pro Entwickler.

Kommentare pro Commit
Anzahl der Kommentare pro Commit.

CLOC pro Commit
Anzahl der geänderten Zeilen im Quellcode pro Commit.

Pull Requests
Gesamtzahl an Pull Requests in einembestimmten Zeitraum.

Gemergte Pull Requests
Anzahl erfolgreicher Pull Requests ineinem bestimmten Zeitraum.

Abgelehnte Pull Requests
Anzahl abgelehnter Pull Requests in einem bestimmten Zeitraum.

Kommentare pro Pull Request
Anzahl der Kommentare pro Pull Request.

[7] vgl. Christopher W. H. Davis. *Agile Metrics in Action: Measuring and Enhancing the Performance of Agile Teams*. 1st. Greenwich, CT, USA: Manning Publications Co., 2015. ISBN: 978-1-61729-248-4, S.62ff.

2.4.2 Projektmanagement[8]

In einem PTS werden Aufgaben definiert und zugewiesen, Bugs verwaltet und Arbeitszeit mit Aufgaben verknüpft. Hier können Daten über das Projektverständnis des Teams, die Geschwindigkeit und vor allem die Konsistenz der Arbeit gesammelt werden. Um bestmögliche Daten erhalten zu können, gibt es folgende Empfehlungen:

- PTS wird von allen genutzt
- Aufgaben mit möglichst vielen Tags versehen
  - Aufgaben kategorisieren (nach "gut", "ok" und "schlecht")
- Aufgaben schätzen
- gemeinsam eine Definition of Done (DoD) festlegen

Jede Arbeit, die am PTS vorbei geht, fällt später bei der Auswertung der Daten durch das Raster. Durch das Taggen der Aufgaben können später Korrelationen ausgewertet werden, vor allem auch durch das Taggen, wie gut die Aufgabe abgelaufen ist. Nur wenn die Aufgabe geschätzt ist, kann festgestellt werden, ob richtig geschätzt wurde oder wie viele Ausreißer es gibt. Dazu muss auch die Arbeitszeit auf der Aufgabe gespeichert werden. Die DoD hilft allgemein den Prozess zu verbessern und Rückläufe im Arbeitsablauf zu minimieren.

Dadurch ergeben sich folgende Kennzahlen aus einem PTS:

Burn Down
: Die Anzahl erledigte Arbeit über die Zeit. Liefert einen Richtwert, wo man sich gerade im Sprint befindet, verglichen zum Commitment.

Velocity
: Eine relative Messung der Konsistenz erledigter Arbeit über die Sprints.

Cummulative Flow
: Zeigt wie viel Aufgaben nach Status dem Team zugewiesen sind über die Zeit.

Lead Time
: Zeit zwischen Start und Abschluss einer Aufgabe, vor allem interessant bei Kanban.

Bug Counts
: Die Anzahl an Bugs über die Zeit.

  Bug-Erzeugungsrate
  : Anzahl Bugs nach Erstellungsdatum.

  Bug-Fertigstellungsrate
  : Anzahl Bugs nach Erledigungsdatum.

Aufgaben-Volumen
: Die Anzahl der Aufgaben und kann der Schätzung gegenübergestellt werden, um die Größe der Aufgaben oder ungeplante Arbeit aufzuzeigen.

Aufgaben-Rückfälligkeit
: Zeigt auf, wie oft Aufgaben im Arbeitsablauf rückwärts gehen.

[8] vgl. Davis, *Agile Metrics in Action*, S.37ff.

2.4.3 Kontinuierliche Integration und Auslieferung[9]

CI- und CD-Systeme stellen sicher, dass die erstellte Software zu jedem Zeitpunkt auslieferbar ist, in dem sie zu definierten Zeitpunkten automatisch neu gebaut und ausgeliefert wird. In einer solchen Build-Pipeline können sehr viel nützliche Daten erzeugt werden, vor allem mit Tools für statische Analysen (wie zum Beispiel SonarQube[10]). Diese Systeme sind aber auch jene Elemente im Softwareentwicklungsprozess, die von Team zu Team am meisten variieren können. Daher hängen die erzeugten Daten auch stark vom jeweiligen Setup ab. Grundsätzlich können aber folgende Kennzahlen aus diesen Systemen ermittelt werden:

Build-Dauer
: Geschätzte und tatsächliche Dauer der Builds.

Build-Status
: Es können die Anzahl der erfolgreichen und fehlerhaften Builds gegenüber gestellt werden.

Build-Frequenz
: Wie oft wird ein Build ausgelöst.

Test Reports
: Anzahl erfolgreicher und fehlerhafter Tests, Gesamtdauer der Tests.

Code Coverage
: Wie viel Prozent des Quellcodes ist mit Tests abgedeckt.

Stresstests oder Benchmarking
: Wird oft im Build Prozess mit getestet mit Tools wie JMeter[11] oder Gatling[12].

[9] vgl. Davis, *Agile Metrics in Action*, S.84ff.
[10] *Continuous Code Quality | SonarQube*.
[11] *Apache JMeter - Apache JMeter™*. URL: `https://jmeter.apache.org/` (besucht am 29. 03. 2018).
[12] *Gatling Load and Performance testing - Open-source load and performance testing*. en-US. URL: `https://gatling.io/` (besucht am 29. 03. 2018).

2.4.4 Produktionssystem[13]

Daten aus den Produktionssystemen können gesammelte APM- oder auch BI-Kennzahlen sein. Diese Kennzahlen ermöglichen Aussagen, ob die Kunden zufrieden sind und wie das System arbeitet. Die BI-Kennzahlen sollten möglichst nahe am Entwicklungsteam gehalten werden, damit es verstehen kann, wie die Kunden die Applikation nutzen. Dazu können Frameworks wie StatsD[14] und Atlas[15] verwendet werden. Im Produktionssystem können folgende Kennzahlen ermittelt werden:

CPU Nutzung
: Auslastung der Prozessoren über die Zeit.

Heap Size
: Auslastung des Heap über die Zeit.

Fehlerraten
: Anzahl Fehler über die Zeit (kann aus dem Logging kommen).

Antwortzeiten
: Dauer der Verarbeitung bestimmter Anfragen.

Benutzeranzahl
: Anzahl gleichzeitiger Benutzer in der Applikation über die Zeit.

Aufenthaltsdauer
: Verweildauer der Benutzer auf bestimmten Seiten.

Conversion Rate
: Anzahl Benutzer die zu Kunden wurden.

Semantisches Logging
: Ermöglicht es, beim Logging strukturierte Daten auszugeben, zum Beispiel: was suchen Benutzer auf bestimmten Seiten.

Verfügbarkeit
: Verfügbarkeit der Applikation über die Zeit.

2.4.5 Übersicht Kennzahlen im Entwicklungsprozess

Die Metriken finden sich nochmal als Tabelle dargestellt und mit den dazugehötigen Fragen, die sie jeweils beantworten, im Anhang A.1.

[13] vgl. Davis, *Agile Metrics in Action*, S.107ff.

[14] *statsd: Daemon for easy but powerful stats aggregation.* original-date: 2010-12-30T00:09:50Z. März 2018. URL: `https://github.com/etsy/statsd` (besucht am 29. 03. 2018).

[15] *atlas: In-memory dimensional time series database.* original-date: 2014-08-05T05:23:04Z. März 2018. URL: `https://github.com/Netflix/atlas` (besucht am 29. 03. 2018).

2.5 Metriken

Eine Softwaremetrik wird vom Institute of Electrical and Electronics Engineers (IEEE) Standard 1061 von 1998 folgendermaßen definiert:

> "Eine Softwarequalitätsmetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet, welcher als Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit interpretierbar ist."[16]

Vereinfacht gesagt, ist eine Metrik eine oder mehrere Kennzahlen, die mithilfe einer Funktion ein Qualitätsmerkmal in einen Zahlenwert abbilden. Eine Kennzahl kann daher auch schon direkt eine Metrik sein, wenn sie in der Lage ist, ein gewünschtes Qualitätsmerkmal abzubilden.

2.5.1 Veröffentlichung von Metriken[17]

Metriken können auf verschiedene Art und Weise veröffentlicht werden. Zwei mögliche Beispiele sind Dashboards oder Emails. Grundsätzlich sollte beachtet werden, dass man sich bei der Veröffentlichung von Metriken innerhalb der Grenzen und Gewohnheiten des Unternehmens bewegen sollte. Außerdem sollte auf folgende Punkte geachtet werden:

Dashboards

- den Zugriff innerhalb der Firma nicht einschränken
  - aber als intern ansehen
- muss nach den Bedürfnissen der Teams anpassbar sein
- Metriken werden als Werkzeug gesehen, nicht als Waffe (gegen andere Teams oder Personen)
- Page Tracking verwenden, um das Nutzungsverhalten zu verstehen

Emails

- aus dem Dashboard optional machen (sonst landen sie schnell automatisch im Spam-Ordner)
- minimal erforderliche Daten, den Rest verlinken zum Dashboard
- den Richtigen Rhytmus finden (zwischen oft genug informieren und nerven)

Arbeitet ein Unternehmen beispielsweise viel mit Reports via Email, dann kann ein reines Dashboard weniger Anerkennung finden. Hier könnte beispielsweise eine Übersicht per Mail versendet und mit Links zum Dashboard versehen werden.

[16] vgl. „IEEE Standard for a Software Quality Metrics Methodology". In: *IEEE Std. 1061-1998* (1998), S.3.

[17] vgl. Davis, *Agile Metrics in Action*, S.177ff.

2.5.2 Allgemeine Metriken

Es gibt einige allgemeine Metriken, die für jedes Scrum Team von Bedeutung sind. Wie stark, kann jedes Team selbst entscheiden, aber sie sollten nicht aus den Augen verloren werden.

- Metrik 1

2.5.3 Eigene Metriken erstellen[18]

Um eigene Metriken erstellen zu können sind 2 Dinge notwendig:

- Daten
- eine Funktion, um die Metrik zu berechnen

Dabei sollte darauf geachtet werden,

- dass man auf die Metrik reagieren kann (Dinge, die einen stören und die man nicht ändern kann, frustrieren oder demotivieren)
- dass sich die Metrik nach den Team-Grundsätzen und Kerngeschäften ausrichtet
- dass die Metrik für sich alleine stehen kann

2.5.4 Agile Prinzipien messen[19]

Um die agilen Prinzipien messen zu können, muss zuerst herausgefunden werden, was die Kernaussagen dieser Prinzipien sind. Dies kann zum Beispiel grafisch, durch die Erstellung einer Wortwolke, wie in Abbildung 2.5 ersichtlich, erreicht werden.

[18] vgl. Davis, *Agile Metrics in Action*, S.127ff.
[19] vgl. Davis, *Agile Metrics in Action*, S.201ff.

Abbildung 2.5: Agile Prinzipien als Wortwolke

Aus dieser Wortwolke heben sich neben den Begriffen “development” und “software” vor allem auch die Begriffe “team”, “processes”, “effective” und “requirements” hervor. Mithilfe dieser Begriffe lassen sich folgende vier Punkte ableiten:

- Effektive Software
- Effektiver Prozess
- Effektives Team
- Effektive Anforderungen

Für jeden dieser vier Punkte sind Metriken aus den unterschiedlichsten Systemen anwendbar[20]:

Effektive Software

- erfolgreiche / fehlerhafte Builds
- Business-Metriken
- Status der Applikation
  - Fehlerraten
  - CPU/Speicher Auslastung
  - Antwort- / Transaktionszeiten

[20] vgl. Davis, *Agile Metrics in Action*, S.219ff.

– Heapgröße / Garbage Collection / Anzahl Threads

Effektiver Prozess

- Velocity
- PTS und VCS Kommentare
- erfolgeiche Releases

Effektives Team

- Lead Time
- Mean Time to Release (MTTR)
- Deploy-Frequenz
- fehlerhafte Builds

Effektive Anforderungen

- Rückläufigkeit
- Lead Time
- MTTR
- Velocity

3 Zielsetzung

Agile Methoden, im speziellen Scrum, sind heutzutage in der Softwareentwicklung sehr weit verbreitet. Ein wichtiges Werkzeug dieser Methoden ist der evolutionäre Ansatz, der in Form von Retrospektiven (bei Scrum) zur kontinuierlichen Verbesserung des agilen Prozesses beitragen soll. In diesen Retrospektiven werden dann auch Maßnahmen getroffen, um solche Verbesserungen umzusetzen. In dieser Arbeit soll ein Vorgehensmodell entwickelt werden, wie solche Verbesserungen oder auch Defizite messbar und somit sichtbar gemacht werden können. Weiters soll eine Software entwickelt werden, um die notwendigen Daten zu sammeln und darstellen zu können.

3.1 Vorgehensmodell

Entwicklung eines Vorgehensmodells zur Bestimmung von relevanten Qualitätsmetriken von agilen Teams. Dabei müssen folgende Punkte beachtet werden:

- Ebene für die die Metriken bestimmt sind (agiles Team, mittleres Management, Geschäftsleitung)
- allgemeine Metriken für diese Ebene
- spezielle Probleme erkennen und Metriken dazu erstellen

3.2 Software

Entwicklung einer Software zum Sammeln von Kennzahlen zur Erstellung von Qualitätsmetriken. Dabei müssen folgende Kriterien beachtet werden:

- Umsetzung in Java
- einzubindende Systeme: BitBucket Server[21], JIRA[22], Jenkins[23], SonarQube[24], Icinga[25]
- Speicherung und Darstellung der Metriken erfolgt in einem Elastic Stack[26]

[21] Atlassian. *Bitbucket Server*. en. URL: `https://www.atlassian.com/software/bitbucket/server` (besucht am 31. 03. 2018).
[22] Atlassian. *Jira | Software zur Vorgangs- und Projektverfolgung*. de-DE. URL: `https://de.atlassian.com/software/jira` (besucht am 31. 03. 2018).
[23] *Jenkins*. URL: `https://jenkins.io/index.html` (besucht am 31. 03. 2018).
[24] *Continuous Code Quality | SonarQube*.
[25] *Icinga*. en-US. URL: `https://www.icinga.com/` (besucht am 31. 03. 2018).
[26] *Elastic Stack*. de-de. URL: `https://www.elastic.co/de/products` (besucht am 31. 03. 2018).

4 Methodik

4.1 Vorgehensmodell

4.1.1 GQM

Befragung Team blabla

4.1.1.1 Vorauswahl

Um eine Vorauswahl an Metriken treffen zu können, wurden alle bisherigen Retrospektiven (es waren genau 15) analysiert und eine Topliste von Schlagwörtern der folgenden Fragestellungen aus den Retrospektiven erstellt:

1. Welche guten Entscheidungen haben wir getroffen?
2. Was haben wir gelernt?
3. Was können wir besser machen?
4. Was nervt uns noch immer?

Dazu wurden die Ergebnisse in eine ElasticSearch Datenbank gespeichert und über eine sogenannte Terms Aggregation die wichtigsten Schlagwörter analysiert. Bei der Indizierung werden die Wörter normalisiert, deshalb die teilweise andere Schreibweise (zum Beispiel wird aus Issue der Term issu).
Aus den Ergebnissen in Anhang A.2 ist ersichtlich, dass

4.2 Software

Technologien, Plattform, etc.

4.2.1 Architektur

Abbildung 4.1 zeigt die Position und Abbildung 4.2 die grobe Architektur der Software (Agile Metrics). Die Software bildet eine Schnittstelle zwischen den einzelnen Systemen des Entwicklungsprozesses und dem System zur Darstellung der Metriken (in diesem Fall ElasticSearch und Kibana).

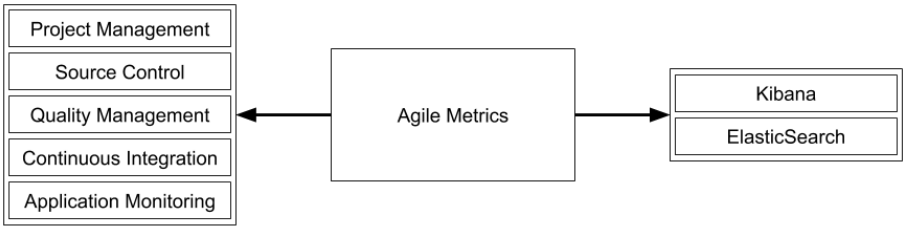


Abbildung 4.1: Position der Software

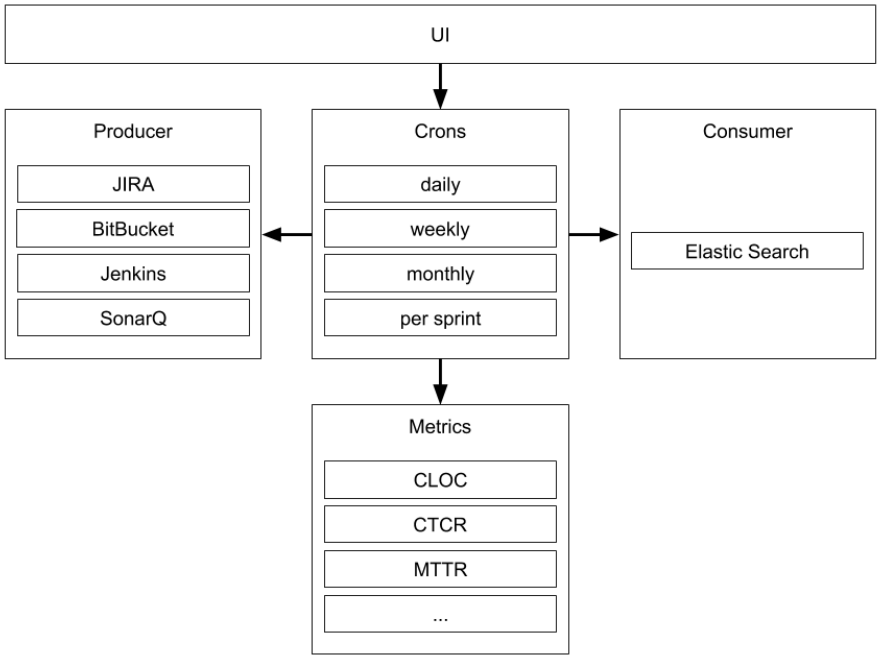


Abbildung 4.2: Übersicht der Software-Architektur

UI
: Bietet eine grafische Benutzeroberfläche zur Konfiguration.

Producer
: Sind Schnittstellen zu allen Systemen, die Messdaten erzeugen.

Crons
: Zeitsteuerung der Messdaten-Abfrage (z.B. täglich oder pro Sprint).

Metrics
: Hier können aus Messdaten direkt Metriken erstellt werden.

Consumer
: Sind Schnittstellen zu allen Systemen, die Messdaten und Metriken konsumieren.

5 Ergebnisse

5.1 Vorgehensmodell

... Ergebnis der Ausarbeitung.

5.2 Einführung des Vorgehensmodells

... bei Gebrüder Weiss.

5.3 Inbetriebnahme der Software

... allgemein, Beschreibung der Connectoren, Darstellungsarten, etc.

5.4 Evaluierung des Vorgehensmodells

... wie wurde es angenommen? Welche Auswirkungen hatte es?

6 Schlussfolgerungen

Effektivität des Modells, Erkenntnisse aus der Einführung bei Gebrüder Weiss

7 Zusammenfassung

Erkenntnisse und Ausblick

Literatur

Apache JMeter - Apache JMeter™. URL: https://jmeter.apache.org/ (besucht am 29. 03. 2018).

atlas: In-memory dimensional time series database. original-date: 2014-08-05T05:23:04Z. März 2018. URL: https://github.com/Netflix/atlas (besucht am 29. 03. 2018).

Atlassian. *Bitbucket Server*. en. URL: https://www.atlassian.com/software/bitbucket/server (besucht am 31. 03. 2018).

– *Jira | Software zur Vorgangs- und Projektverfolgung*. de-DE. URL: https://de.atlassian.com/software/jira (besucht am 31. 03. 2018).

Continuous Code Quality | SonarQube. URL: https://www.sonarqube.org/ (besucht am 05. 01. 2018).

Davis, Christopher W. H. *Agile Metrics in Action: Measuring and Enhancing the Performance of Agile Teams*. 1st. Greenwich, CT, USA: Manning Publications Co., 2015. ISBN: 978-1-61729-248-4.

Dräther, Rolf, Holger Koschek und Carsten Sahling. *Scrum: kurz & gut*. 1. Auflage. O'Reillys Taschenbibliothek. Beijing Cambridge Farnham Köln Sebastopol, Tokyo: O'Reilly, 2013. ISBN: 978-3-86899-833-7.

Elastic Stack. de-de. URL: https://www.elastic.co/de/products (besucht am 31. 03. 2018).

Gatling Load and Performance testing - Open-source load and performance testing. en-US. URL: https://gatling.io/ (besucht am 29. 03. 2018).

Hoffmann, Dirk W. *Software-Qualität*. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-35699-5 978-3-642-35700-8.

Icinga. en-US. URL: https://www.icinga.com/ (besucht am 31. 03. 2018).

„IEEE Standard for a Software Quality Metrics Methodology". In: *IEEE Std. 1061-1998* (1998).

Jenkins. URL: https://jenkins.io/index.html (besucht am 31. 03. 2018).

Manifest für Agile Softwareentwicklung. URL: http://agilemanifesto.org/iso/de/manifesto.html (besucht am 16. 03. 2018).

Prinzipien hinter dem Agilen Manifest. URL: http://agilemanifesto.org/iso/de/principles.html (besucht am 16. 03. 2018).

Scrum. URL: http://www.scrum.org (besucht am 05. 01. 2018).

statsd: Daemon for easy but powerful stats aggregation. original-date: 2010-12-30T00:09:50Z. März 2018. URL: https://github.com/etsy/statsd (besucht am 29. 03. 2018).

The Scrum Framework Poster | Scrum.org. URL: https://www.scrum.org/resources/scrum-framework-poster (besucht am 01. 04. 2018).

Anhang

A.1 Metriken aus dem Entwicklungsprozess

CLOC	Anzahl der geänderten Zeilen im Quellcode.
Wie viele Änderungen passieren in der Codebasis? *Wo finden die meisten Änderungen statt?*	
CLOC pro Entwickler	Anzahl der geänderten Zeilen im Quellcode pro Entwickler.
Wie viel Code ändert jeder im Team? *Wer ist wie oft in welchem Modul?*	
CLOC pro Commit	Anzahl der geänderten Zeilen im Quellcode pro Commit.
Wie groß sind die Commits?	
Commits	Gesamtzahl an Commits in einem bestimmten Zeitraum.
Wie viel Änderungen wurden im Quellcode vorgenommen?	
Commits pro Entwickler	Gesamtzahl an Commits in einem bestimmten Zeitraum pro Entwickler.
Wie viel Änderungen wurden im Quellcode von einem Entwickler vorgenommen?	
Kommentare pro Commit	Anzahl der Kommentare pro Commit.
Wer arbeitet zusammen? *Wie viel wird zusammengearbeitet?*	
Pull Requests	Gesamtzahl an Pull Requests in einem bestimmten Zeitraum.
Wird mit Pull Requests gearbeitet? *Werden Reviews gemacht?*	
Gemergte Pull Requests	Anzahl erfolgreicher Pull Requests in einem bestimmten Zeitraum.
Wie oft werden erfolgreiche Änderungen in die Codebasis übernommen?	
Abgelehnte Pull Requests	Anzahl abgelehnter Pull Requests in einem bestimmten Zeitraum.
Wie oft werden Änderungen an der Codebasis abgelehnt? *Wie klar sind die Erwartungen des Teams an eine abgeschlossene Änderung (DoD)?*	
Kommentare pro Pull Request	Anzahl der Kommentare pro Pull Request.
Wer arbeitet zusammen? *Wie viel wird zusammengearbeitet?*	

Tabelle A.1: Kennzahlen aus dem VCS

Burn Down	Die Anzahl erledigte Arbeit über die Zeit. Liefert einen Richtwert, wo man sich gerade im Sprint befindet, verglichen zum Commitment.
Erfüllt das Team seine Commitments? *Plant das Team seine Arbeit realistisch?*	
Velocity	Eine relative Messung der Konsistenz erledigter Arbeit über die Sprints.
Wie konsistent arbeitet das Team?	
Cumulative Flow	Zeigt wie viel Aufgaben nach Status dem Team zugewiesen sind über die Zeit.
Gibt es Engpässe oder Schwachstellen im Prozess? *Müssen gewisse Abläufe im Prozess optimiert werden?*	
Lead Time	Zeit zwischen Start und Abschluss einer Aufgabe, vor allem interessant bei Kanban.
Wie schnell können Aufgaben vom Team erledigt werden? *Wie lange dauert die Umsetzung eines neuen Features?*	
Bug Counts	Die Anzahl an Bugs über die Zeit.
Wie viele Fehler werden vom Team im Entwicklungsprozess übersehen? *Wie viel ungeplante Arbeit kam zum Sprint dazu?*	
Bug-Erzeugungsrate	Anzahl Bugs nach Erstellungsdatum.
Wie viele Fehler wurden zu einem bestimmten Zeitpunkt erzeugt?	
Bug-Fertigstellungsrate	Anzahl Bugs nach Erledigungsdatum.
Wie viele Fehler wurden zu einem bestimmten Zeitpunkt beseitgt?	
Aufgaben-Volumen	Ist die Anzahl der Aufgaben und kann der Schätzung gegenübergestellt werden, um die Größe der Aufgaben oder ungeplante Arbeit aufzuzeigen.
Wie viel ungeplante Arbeit kam zum Sprint dazu? *Wie groß ist die durchschnittliche Aufgabe? Gibt es Ausreißer?*	
Aufgaben-Rückfälligkeit	Zeigt auf, wie oft Aufgaben im Arbeitsablauf rückwärts gehen.
Wie viele Aufgaben werden wieder in einen vorhergehenden Status gesetzt? *Gibt es Probleme beim Verständnis der Aufgaben?* *Wie klar sind die Erwartungen des Teams an eine abgeschlossene Änderung (DoD)?*	

Tabelle A.2: Kennzahlen aus dem PTS

Build-Dauer	Geschätzte und tatsächliche Dauer der Builds.
Wie lange dauert es ein Softwareartefakt zu erstellen? *Wie verändert sich die Dauer der Erstellung eines Softwareartefakts über die Zeit?*	
Build-Status	Es können die Anzahl der erfolgreichen und fehlerhaften Builds gegenüber gestellt werden.
Gibt es ein Problem im Freigabeprozess?	
Build-Frequenz	Wie oft wird ein Build ausgelöst.
Wrid oft genug ein neues Softwareartefakt erstellt?	
Test Reports	Anzahl erfolgreicher und fehlerhafter Tests, Gesamtdauer der Tests.
Wie lange dauert ein kompletter Testdurchlauf? *Gibt es Tests, die optimiert werden müssen?* *Wie oft werden fehlerhafte Tests in die Codebasis aufgenommen?*	
Code Coverage	Wie viel Prozent des Quellcodes ist mit Tests abgedeckt.
Gibt es Module, die nicht oder schlecht getestet sind? *Wie sieht die Entwicklung der Testabdeckung über die Zeit aus?*	
Stresstests oder Benchmarking	Hier kann das Ergebnisse die unterschiedliche Reports sein.
Ist das Produkt auch noch unter Last verwendbar? *Wie verändert sich die Leistung über die Zeit?*	

Tabelle A.3: Kennzahlen aus den CI- und CD -Systemen

CPU Nutzung	Auslastung der Prozessoren über die Zeit.
Heap Size	Auslastung des Heap über die Zeit.
Arbeitet die Software technisch effizient?	
Ist die Hardware ausreichend?	
Gibt es eine erhöhte Auslastung nach einer Änderung?	
Fehlerraten	Anzahl Fehler über die Zeit (kann aus dem Logging kommen).
Werden seit einer Änderung mehr Fehler produziert?	
Wie entwickelt sich die Fehlerrate über die Zeit?	
Antwortzeiten	Dauer der Verarbeitung bestimmter Anfragen.
Reagiert und arbeitet das Produkt noch schnell genug?	
Gibt es Geschwindigkeitsprobleme seit der letzen Änderung?	
Wie entwickeln sich die Antwortzeiten über die Zeit?	
Benutzeranzahl	Anzahl gleichzeitiger Benutzer in der Applikation über die Zeit.
Wie entwickeln sich die Benutzerzahlen mit der Zeit?	
Geht das Produkt in die richtige Richtung?	
Ist mit höheren Lasten zu rechnen?	
Aufenthaltsdauer	Verweildauer der Benutzer auf bestimmten Seiten.
Welche Features werden besonders oft / selten genutzt?	
Hat das neue Feature den gewünschten Effekt? Wird es genutzt?	
Conversion Rate	Anzahl Benutzer die zu Kunden wurden.
Wie entwicklt sich die Zahl der zahlenden Neukunden?	
Semantisches Logging	Strukturierte Daten aus dem Logging.
Hier können Daten zu anderen Fragen gesammelt werden, die für den Prozess wichtig sind.	
Verfügbarkeit	Verfügbarkeit der Applikation über die Zeit.
Wie hoch ist die Ausfallsicherheit?	
Wie lange war die Applikation nicht verfügbar?	

Tabelle A.4: Kennzahlen aus den APM- und BI -Systemen

A.2 Ergebnisse Analyse Retrospektiven

Welche guten Entscheidungen haben wir getroffen?

1. sprint (4)
2. einblick (3)
3. onboarding (3)
4. pair (3)
5. programming (3)
6. system (3)
7. arbeit (2)
8. daily (2)
9. erledig (2)
10. information (2)
11. issu (2)
12. po (2)
13. review (2)
14. reviewing (2)
15. schnell (2)
16. stori (2)
17. urlaub (2)
18. angenehm (1)
19. annehm (1)
20. cloud (1)
21. dailys (1)
22. diskussion (1)
23. dor (1)
24. durchgefuhrt (1)
25. einfach (1)

Was haben wir gelernt?

1. sprint (7)
2. onboarding (4)
3. team (4)
4. arbeit (3)
5. besprech (3)
6. board (3)
7. datenfluss (3)
8. issus (3)
9. planungswoch (3)
10. retro (3)
11. richtlini (3)
12. system (3)

13. uberblick (3)
14. umgestellt (3)
15. altlast (2)
16. analogboard (2)
17. approved (2)
18. backlog (2)
19. daily (2)
20. digital (2)
21. direkt (2)
22. genau (2)
23. impediment (2)
24. infrastruktur (2)
25. iso (2)

Was können wir besser machen?

1. sprint (10)
2. review (7)
3. checklist (4)
4. daily (4)
5. display (4)
6. doku (4)
7. issu (4)
8. po (4)
9. einarbeitung (3)
10. einkalkuli (3)
11. gross (3)
12. https (3)
13. java (3)
14. stori (3)
15. ablauf (2)
16. anderung (2)
17. arbeitspaket (2)
18. aufnehm (2)
19. aufteil (2)
20. backlog (2)
21. blocked (2)
22. dokumenti (2)
23. erledig (2)
24. geplant (2)
25. geschatzt (2)

Was nervt uns noch immer?

1. problem (11)
2. updat (11)
3. apis (10)
4. archiv (10)
5. erreichbar (10)
6. infrastruktur (10)
7. jenkin (10)
8. test (9)
9. umgebung (9)
10. dba (8)
11. eingerichtet (8)
12. jndi (8)
13. laut (8)
14. verwendbar (8)
15. arbeit (7)
16. impediment (7)
17. iso (7)
18. lang (7)
19. mitarbeit (6)
20. mehr (5)
21. wichtig (5)
22. anderung (4)
23. aufteilbar (4)
24. gross (4)
25. klar (4)

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am [Tag. Monat Jahr anführen] [Vor- und Nachname Verfasser/in]