

# **Agile Metriken**

## **Qualitätssicherung in agilen Teams**

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

# Präambel

Der Verfasser der vorliegenden Arbeit bekennt sich zu einer geschlechtergerechten Sprachverwendung. Aufgrund der häufig vorkommenden Nennung einiger Personengruppen werden zugunsten der flüssigeren Lesbarkeit für diese sowohl männliche als auch weibliche "Rollen" vergeben; diese sind ausgewogen und stereotypen Rollenbildern möglichst entgegenwirkend vergeben worden:

- Die Entwicklerin
- Der Kunde
- Die Managerin
- Der Teilnehmer
- Die Botschafterin
- Der Stakeholder
- Die Benutzerin

Folgende Fachbegriffe werden direkt aus dem Englischen geschlechtsneutral übernommen und daher keine Rollen dafür vergeben:

- Scrum Master
- Product Owner
- Meta Scrum

# Kurzreferat

## Agile Metriken - Qualitätssicherung in agilen Teams

Agile Vorgehensmodelle, insbesondere Scrum, sind bei modernen Entwicklungsteams sehr beliebt und weit verbreitet. Ein Grundsatz von Scrum bildet der Empirismus. Empirismus ist die Theorie, dass Wissen aus Erfahrung erlangt wird. Um ein solches Wissen, im Speziellen über Qualität, aufbauen zu können und basierend auf diesem Wissen Entscheidungen treffen zu können, ist es hilfreich, bestimmte Qualitätsmerkmale in Form von Metriken bereitzustellen. Qualität in agilen Teams lässt sich unterteilen in Produktqualität und Prozessqualität, deren Merkmale sich durch Metriken als Zahlenwerte darstellen lassen. Solche Metriken sind pro Team individuell und müssen daher mit dafür geeigneten Methoden ermittelt werden. Bei komplexeren Problemstellungen sind Standard-Metriken oft nicht mehr ausreichend und es müssen eigene, speziell auf das identifizierte Problem zugeschnittene Metriken erstellt werden. Die Metriken müssen dann in geeigneter Form dargestellt und dem Team zur Verfügung gestellt werden. Die Ermittlung der Metriken erfolgte in dieser Arbeit über die Goal Question Metric (GQM)-Methodik, einer Methodik zur Bestimmung von Metriken, für deren Grundlage die Daten der vorangegangenen Retrospektiven ausgewertet wurden. Zusätzlich wurden verschiedene Metriken in einer Umfrage vorgestellt und von den Teammitgliedern einzeln bewertet. Dabei konnte gezeigt werden, dass die vom Team gewählten Metriken, bis auf wenige Ausnahmen dem Ergebnis aus der GQM-Methodik entsprachen und dieses somit nicht angepasst werden musste. Die zugrundeliegenden Daten für diese ermittelten Metriken werden täglich in den einzelnen Systemen entlang des Entwicklungsprozesses erzeugt und über Schnittstellen zur Verfügung gestellt. Die Metriken werden mit einer selbst erstellten und quelloffen zur Verfügung gestellten Software erzeugt, in einem bereits vorhandenen Elastic Stack gespeichert und in einem Dashboard dargestellt. Auf dem Dashboard sind die Metriken visuell gruppiert, um allen Teammitgliedern eine möglichst effiziente Sicht bereitzustellen. Zur Evaluierung wurde das System in einem relativ fortgeschrittenen Scrum Team über den Zeitraum von nicht ganz drei Sprints eingesetzt. Dadurch konnte gezeigt werden, dass das zur Verfügung gestellte Werkzeug bereits nach kurzer Zeit von den Teammitgliedern genutzt wurde. Auch wenn in dieser kurzen Zeit noch kein eindeutiger Qualitätsgewinn nachgewiesen werden konnte, wurde in den Interviews zur Evaluierung klar, dass der Zweck erkannt und das Dashboard bereits wie vorgesehen genutzt wird. Es wurden sogar bereits Ideen geäußert, gewisse Metriken zu kombinieren, um Korrelationen nachzuweisen und Qualitätseinbußen besser verstehen und vermeiden zu können, was genau dem Ziel dieser Arbeit entsprach.

# Abstract

## Agile Metrics - Quality Assurance In Agile Teams

Agile approaches, especially Scrum, are very popular among modern software development teams. A fundamental principle of Scrum is empiricism, which is the theory of gaining new knowledge through experience. To acquire such a knowledge, especially in regard to quality, and make decisions in the future based on that knowledge, it is useful to present certain quality characteristics to the team. Quality in agile teams can be subdivided into product and process quality, which characteristics can be presented as numeric values through metrics. Such metrics are individual for each team and therefore need to be determined choosing appropriate methods. For complex problems, standard metrics are no more sufficient and custom metrics are needed, adapted for this kind of problem. Then these metrics need to be published and presented to the team in a proper way. The determination of the metrics was done with the GQM-method, which is a method for examining metrics. As a starting point for the GQM-method the data of past retrospectives was analyzed. Additionally metrics were presented to the team in a survey and each team member had to rate those metrics. It was shown that the metrics chosen by the team matched the result determined with the GQM-method with only a few exceptions, so that nothing needed to be changed. The data needed for those metrics is produced daily by the systems involved in the development process and provided by interfaces. The metrics are generated by a self-constructed open-source software, saved in an Elastic Stack and presented on a dashboard. The dashboard is grouped visually to provide an optimal view on these metrics for every team member. For evaluation purposes the system was tested in a matured scrum team for almost three sprints. These tests show that the tool provided to the team has been used by the members for the benefit of the whole team. Even if there could have not been demonstrated a measurable quality improvement, the interviews showed that the purpose had become clear and the dashboard was used by the team members as planned. There had even been some ideas about combining metrics to prove correlations between them and to better understand and avoid loss in quality, which has exactly been the purpose of this work.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>7</b>
<b>Tabellenverzeichnis</b>	<b>8</b>
<b>Abkürzungsverzeichnis</b>	<b>9</b>
<b>1 Einleitung</b>	<b>10</b>
1.1 Zielsetzung . . . . .	10
1.2 Aufbau der Arbeit . . . . .	11
<b>2 Stand der Technik</b>	<b>12</b>
2.1 Agile Softwareentwicklung . . . . .	12
2.1.1 Agiles Manifest . . . . .	12
2.1.2 Agile Prinzipien . . . . .	13
2.2 Scrum . . . . .	14
2.2.1 Scrum in mehreren Teams . . . . .	15
2.2.1.1 Scrum of Scrums (SoS) . . . . .	17
2.2.1.2 Large Scale Scrum (LeSS) . . . . .	17
2.2.1.3 Scrum at Scale . . . . .	19
2.3 Software-Qualität . . . . .	20
2.3.1 Produktqualität . . . . .	22
2.3.2 Prozessqualität . . . . .	23
2.4 Metriken . . . . .	25
2.4.1 Versionsverwaltung . . . . .	26
2.4.2 Projektmanagement . . . . .	26
2.4.3 Kontinuierliche Integration und Auslieferung . . . . .	27
2.4.4 Produktionssystem . . . . .	28
2.4.5 Übersicht Kennzahlen im Entwicklungsprozess . . . . .	29
2.4.6 Eigene Metriken erstellen . . . . .	29
2.4.7 Veröffentlichung von Metriken . . . . .	29
2.4.8 Agile Prinzipien messen . . . . .	30
2.5 Qualitätsmodelle . . . . .	32
2.5.1 Factor Criteria Metrics (FCM)-Modell . . . . .	32
2.6 GQM . . . . .	34
<b>3 Vorgehensweise</b>	<b>36</b>
3.1 Metriken bestimmen . . . . .	36

3.2 Software . . . . .	37
3.2.1 Anforderungen . . . . .	37
3.3 Evaluierung . . . . .	38
<b>4 Umsetzung</b>	<b>39</b>
4.1 Gegebenheiten . . . . .	39
4.2 Metriken Identifizieren . . . . .	39
4.2.1 GQM . . . . .	39
4.2.2 Umfrage im Team . . . . .	42
4.3 Software . . . . .	43
4.3.1 Architektur . . . . .	43
4.3.2 Lizenz . . . . .	44
4.3.3 Version Control System (VCS), Continuous Integration (CI) und Qualitätssicherung (QS) . . . . .	45
4.4 Inbetriebnahme . . . . .	48
4.4.1 Visualisierung der Metriken . . . . .	49
4.4.2 Status-Schnittstelle . . . . .	52
<b>5 Evaluierung</b>	<b>53</b>
5.1 Quantitative Evaluierung . . . . .	54
5.2 Qualitative Evaluierung . . . . .	66
5.2.1 Interview-Fragen . . . . .	66
5.2.2 Interview-Antworten . . . . .	67
<b>6 Schlussfolgerungen</b>	<b>68</b>
<b>7 Zusammenfassung</b>	<b>69</b>
<b>Literaturverzeichnis</b>	<b>71</b>
<b>Anhang</b>	<b>73</b>
A.1 Metriken aus dem Entwicklungsprozess . . . . .	74
A.2 Ergebnisse Analyse Retrospektiven . . . . .	78
A.3 Umfrage Scrum Team . . . . .	81
A.3.1 Fragebogen . . . . .	81
A.3.2 Ergebnisse . . . . .	91
A.4 Transkripte Interviews . . . . .	93
A.4.1 Product Owner . . . . .	93
A.4.2 Scrum Master . . . . .	93
A.4.3 Entwicklerin . . . . .	95
<b>Eidesstattliche Erklärung</b>	<b>96</b>

# Abbildungsverzeichnis

2.1	Scrum Framework . . . . .	14
2.2	Scrum Teams . . . . .	16
2.3	SoS auf 3 Ebenen . . . . .	17
2.4	LeSS Framework . . . . .	18
2.5	Scrum at Scale Framework - SMPO Zyklus . . . . .	19
2.6	Korrelationsmatrix Qualitätskriterien . . . . .	21
2.7	Software-Qualität . . . . .	22
2.8	Systeme im Softwareentwicklungsprozess . . . . .	25
2.9	Agile Prinzipien als Wortwolke . . . . .	30
2.10	Struktur des FCM Modells . . . . .	33
2.11	Struktur eines Ergebnisses der GQM-Methodik . . . . .	34
4.1	Position der Software . . . . .	43
4.2	Übersicht der Software-Architektur . . . . .	44
4.3	GitHub Pages - Agile Metrics . . . . .	45
4.4	Travis CI - Agile Metrics . . . . .	46
4.5	SonarCloud - Agile Metrics . . . . .	47
4.6	Logausgaben bei erfolgreichem Start . . . . .	48
4.7	Teil des Dashboards mit den kurzfristigen Metriken . . . . .	49
4.8	Teil des Dashboards mit den langfristigen Metriken . . . . .	51
4.9	Ergebnis der Abfrage der Status-Schnittstelle . . . . .	52
5.1	Quantitative Evaluierung - Acceptance Criteria Volatility . . . . .	54
5.2	Quantitative Evaluierung - Bug Count . . . . .	55
5.3	Quantitative Evaluierung - Bug Rate . . . . .	55
5.4	Quantitative Evaluierung - Burndown . . . . .	56
5.5	Quantitative Evaluierung - Coverage der kritischen Komponenten . . . . .	57
5.6	Quantitative Evaluierung - Coverage aller Komponenten . . . . .	57
5.7	Quantitative Evaluierung - Cumulative Flow . . . . .	58
5.8	Quantitative Evaluierung - Labels . . . . .	59
5.9	Quantitative Evaluierung - Lead Time . . . . .	60
5.10	Quantitative Evaluierung - Recidivism . . . . .	61
5.11	Quantitative Evaluierung - Test Execution Time . . . . .	62
5.12	Quantitative Evaluierung - Velocity . . . . .	63
5.13	Quantitative Evaluierung - Velocity Durchschnitt . . . . .	63
5.14	Quantitative Evaluierung - Issue Volume . . . . .	64
5.15	Quantitative Evaluierung - Issue Volume Total . . . . .	64

# Tabellenverzeichnis

2.1	Beispiel GQM-Methodik . . . . .	35
4.1	GQM-Ergebnis - Ablenkung der Entwicklerinnen . . . . .	41
4.2	GQM-Ergebnis - Schwachstellen im Prozess . . . . .	41
4.3	GQM-Ergebnis - Aufgabengröße . . . . .	41
A.1	Kennzahlen aus dem VCS . . . . .	74
A.2	Kennzahlen aus dem Project Tracking System (PTS) . . . . .	75
A.3	Kennzahlen aus den CI- und Continuous Delivery (CD) . . . . .	76
A.4	Kennzahlen aus den Application-Performance-Monitoring (APM)- und Business-Intelligence (BI) . . . . .	77

# Abkürzungsverzeichnis

- APM** Application-Performance-Monitoring  
**BI** Business-Intelligence  
**CD** Continuous Delivery  
**CI** Continuous Integration  
**CLOC** Changed Lines of Code  
**DoD** Definition of Done  
**EAT** Executive Action Team  
**EMS** Executive Meta Scrum  
**FCM** Factor Criteria Metrics  
**GQM** Goal Question Metric  
**IEEE** Institute of Electrical and Electronics Engineers  
**LeSS** Large Scale Scrum  
**LOC** Lines of Code  
**MTTF** Mean Time to Failure  
**MTTR** Mean Time to Release  
**NASA** National Aeronautics and Space Administration  
**NoSQL** Not-only-SQL  
**PBR** Product Backlog Refinement  
**PTS** Project Tracking System  
**QS** Qualitätssicherung  
**SoS** Scrum of Scrums  
**VCS** Version Control System

# 1 Einleitung

Dieses Kapitel gibt eine kurze Übersicht über den Aufbau und die Motivation hinter dieser Arbeit.

## 1.1 Zielsetzung

Agile Prozesse, insbesondere Scrum, basieren auf empirischen, also durch fortlaufende Beobachtung erhobenen und ausgewerteten, Daten (siehe Abschnitt 2.2). Das bedeutet, dass sich der Prozess durch Reflexion verbessert und auch Veränderung zulässt. Um diese Reflexion zu vereinfachen, ist es hilfreich, gewisse Kennzahlen des Prozesses und des Produktes grafisch darzustellen. Im Speziellen Metriken können eine gute qualitative Auskunft über den aktuellen Status geben. Ziel dieser Arbeit soll es sein, Metriken zu ermitteln, die Qualitätsprobleme im Entwicklungsprozess oder im Softwareprodukt quantitativ abbilden können. Weiters sollen diese Metriken gesammelt und grafisch dargestellt werden. Sie können aus Daten generiert werden, die bei der täglichen Arbeit in den jeweiligen Systemen erzeugt werden. Solche Systeme reichen von Version Control Systems (VCS), über Project Tracking Systems (PTS) und Continuous Integration (CI) / Continuous Delivery (CD), bis hin zu Application-Performance-Monitoring (APM). Diese Daten können meist über Schnittstellen abgefragt und anschließend aggregiert abgelegt werden. Umgesetzt wird das Ganze in einem relativ jungen, aber im Scrum Prozess bereits weit fortgeschrittenen Scrum-Team.

## 1.2 Aufbau der Arbeit

Nach der Einleitung gibt das Kapitel “Stand der Technik” einen Einblick in die unterschiedlichen Themen und die theoretischen Hintergründe dieser Arbeit. Zuerst werden die Grundsäulen der agilen Softwareentwicklung, das agile Manifest und die agilen Prinzipien, genauer beschrieben. Darauf folgend wird auf Scrum eingegangen, zusätzlich werden Ansätze für Scrum in mehreren Teams aufgezeigt. Anschließend wird zu Qualität übergegangen, im Speziellen Software- und Prozessqualität. Basierend auf den beiden vorherigen Themen werden Metriken genauer erörtert, was auch der Hauptteil dieses Kapitels darstellt. Im ersten Teil werden Metriken aus den unterschiedlichsten Systemen vorgestellt und die Erstellung eigener Metriken erläutert. Danach folgen Hinweise zur Veröffentlichung von Metriken und der Messung von Agilen Prinzipien. Den Schluss bildet noch ein Überblick über Qualitätsmodelle und das FCM-Modell im Detail, sowie die GQM-Methodik.

Im Kapitel “Vorgehensweise” wird beschrieben, wie bei der Bestimmung der relevanten Metriken für das Team, bei der Erstellung der Software und bei der Evaluierung der Ergebnisse vorgegangen wird.

Das Kapitel “Umsetzung” soll in weiterer Folge, wie der Titel schon sagt, die Umsetzung der Lösung zeigen. Anfangs werden die Gegebenheiten erläutert, in denen die erstellte Software eingesetzt wird. Weiters wird mit der Identifizierung der Metriken gestartet und diese anschließend gesammelt. Zuletzt wird die Darstellung detaillierter aufgezeigt. Es folgt das Kapitel “Evaluierung”, in dem die Ergebnisse qualitativ in Form von Interviews und quantitativ in Form der Metriken evaluiert werden. In den Kapiteln “Schlussfolgerungen” und “Zusammenfassung” werden die Ergebnisse nochmal reflektiert, zusammengefasst und ein Ausblick für mögliche weitere Arbeiten gegeben.

# 2 Stand der Technik

Um eine Übersicht über die unterschiedlichen Themen zu bieten, werden im Folgenden die theoretischen Hintergründe dieser Arbeit näher erörtert. Eine Übersicht am Anfang soll einen Einblick in die agile Softwareentwicklung geben. Darauf folgend wird etwas genauer Scrum und die möglichen Varianten bei mehreren Teams erläutert. Als Übergang dient eine kurze Aufarbeitung des Themas Software-Qualität, bevor die unterschiedliche Metriken im Detail erklärt werden. Am Ende folgt noch eine Beschreibung von Qualitätsmodellen, im Speziellen das FCM-Modell und zuletzt als Ergänzung noch die GQM-Methodik.

## 2.1 Agile Softwareentwicklung

Diese Arbeit beschäftigt sich mit agilen Teams, deshalb ist es essentiell, zu verstehen, was der Gedanke hinter dem agilen Entwicklungsansatz ist. Seinen Ursprung hat das Ganze im Jahr 2001, als sich 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 der 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<sup>1</sup>

Das Scrum Framework ist eine solche agile Softwareentwicklungs-Methodologie. Scrum basiert auf Empirismus, also der Theorie, dass Wissen aus Erfahrung gewonnen 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.

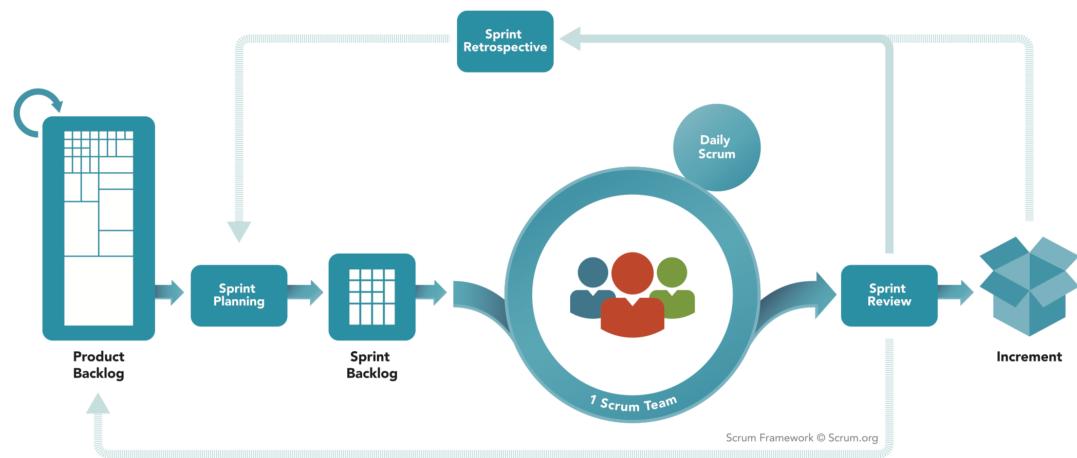
### Inspektion

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

### Adaption

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

## SCRUM FRAMEWORK



 Scrum.org

Abbildung 2.1: Scrum Framework<sup>2</sup>

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

<sup>1</sup>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.13ff.

<sup>2</sup>*The Scrum Framework Poster / Scrum.org*. URL: <https://www.scrum.org/resources/scrum-framework-poster> (besucht am 01.04.2018).

- **Rollen**
  - **Development Team:** Selbstorganisiertes Team, das am Produkt arbeitet.
  - **Scrum Master:** 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 zwei bis vier Wochen, in dem ein fertiges, verwendbares und potentiell auslieferbares Produkt-Inkrement entwickelt wird.
  - **Sprint Planning:** Planung eines Sprints. Hier verpflichtet 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.
  - **Inkrement:** Entsteht am Ende eines jeden Sprints und ist eine lauffähige Version des Produktes, die releasefähig ist.

### 2.2.1 Scrum in mehreren Teams<sup>3</sup>

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:

---

<sup>3</sup>vgl. Dräther, Koschek und Sahling, *Scrum: kurz & gut*, S.172ff.

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

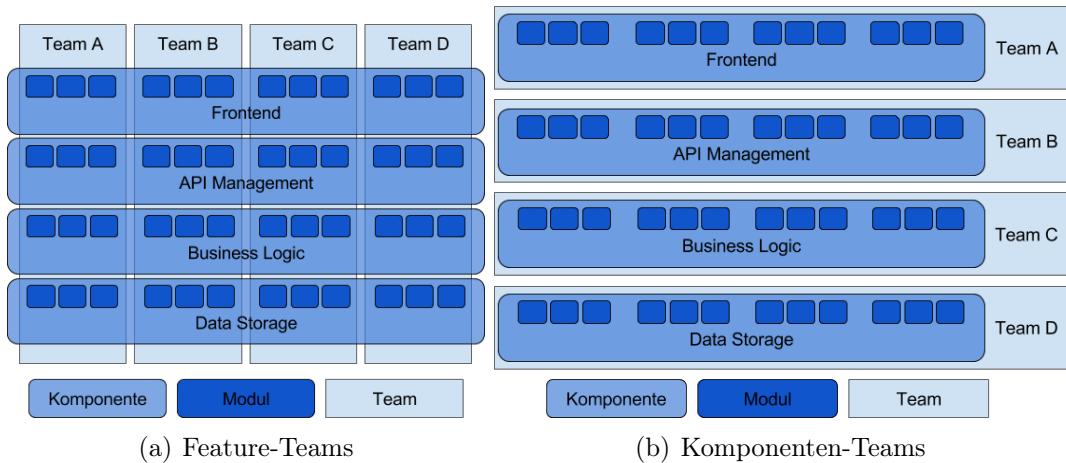


Abbildung 2.2: Scrum Teams

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

Um Scrum auf mehrere Teams skalieren zu können, existieren bereits unterschiedlichste Frameworks. Drei davon sind SoS, LeSS und Scrum at Scale, die im Folgenden kurz vorgestellt werden.

### 2.2.1.1 Scrum of Scrums (SoS)<sup>4</sup>

SoS ist eine Technik, um Scrum auf große Gruppen zu skalieren. Dabei werden die Gruppen in agile Teams von fünf bis zehn Personen geteilt. Nach jedem Daily Scrum wird pro Team eine Botschafterin bestimmt, um an einem täglichen Meeting mit anderen Botschafterinnen teilzunehmen, das sogenannte Scrum of Scrums. Je nach Kontext sind die Botschafterinnen technische Teilnehmer, Scrum Master oder sogar Team-Managerinnen. Das SoS hat sein eigenes Backlog, bei dem es Fertigstellungen, nächste Schritte und Hindernisse im Auge behält.

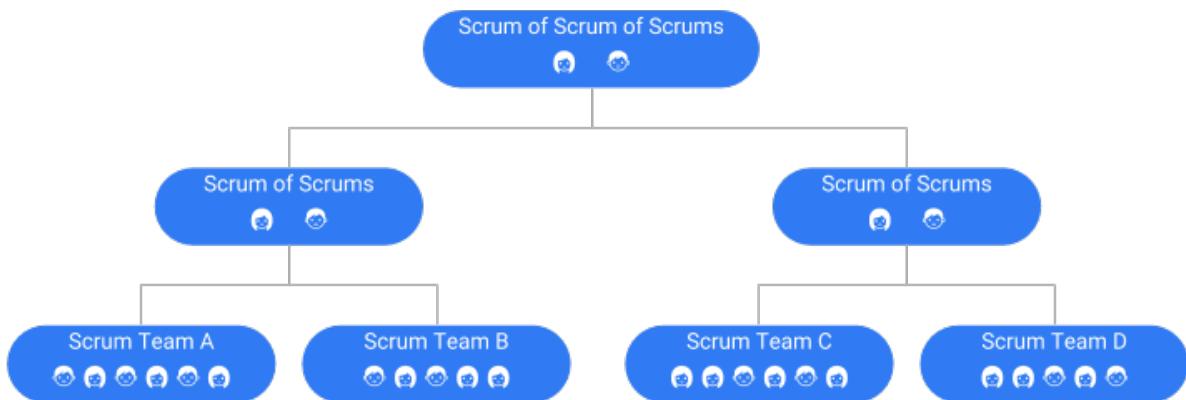


Abbildung 2.3: SoS auf 3 Ebenen

SoS ist beliebig skalierbar, das ermöglicht auch mehrere Scrum of Scrums Ebenen, wie Abbildung 2.3 zeigt.

### 2.2.1.2 Large Scale Scrum (LeSS)<sup>5</sup>

LeSS adaptiert die Grundsätze von Scrum in einen größeren Rahmen. Daher muss auch zuerst Scrum für ein Team verstanden werden, bevor mit LeSS begonnen werden kann. Es unterscheidet dabei zwei Varianten:

**LeSS** für bis zu 8 Teams mit jeweils acht Mitgliedern

**LeSS Huge** bis zu mehrere tausend Personen an einem Produkt

<sup>4</sup>vgl. *Scrum of Scrums / Agile Alliance*. URL: <https://www.agilealliance.org/glossary/scrum-of-scrums/> (besucht am 06.06.2018).

<sup>5</sup>vgl. *Overview - Large Scale Scrum (LeSS)*. URL: <https://less.works/less/framework/index.html> (besucht am 06.06.2018).

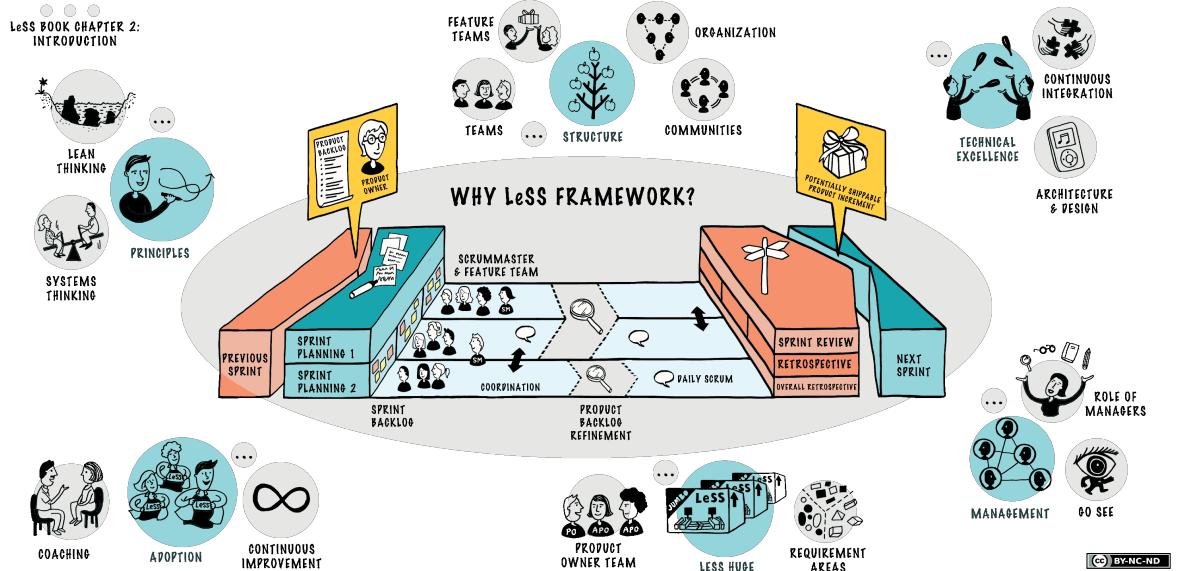


Abbildung 2.4: LeSS Framework<sup>6</sup>

Es gibt viele Dinge, die gleich bleiben, wie bei einem Team: Product Backlog, Definition of Done (DoD), nach jedem Sprint ein lauffähiges Inkrement, Product Owner, cross-funktionale Teams und ein Sprint. Darüber hinaus gibt es aber auch einige Unterschiede:

**Sprint Planning 1** kann Personen von allen Teams beinhalten, die sich ihre Arbeit selbstständig aufteilen.

**Sprint Planning 2** wird von jedem Team selbst abgehalten, es können aber mehrere Teams in einem Raum sein.

**Daily Scrum** wird auch unabhängig in jedem Team abgehalten, aber eine Person aus Team A kann bei Team B dabei sein, um den Informationsfluss zu erhöhen.

**Overall Product Backlog Refinement (PBR)** ist kurz und informativ, um abzuklären, welches Team vermutlich welche Aufgabe übernimmt, für das spätere PBR.

**PBR** können auch mehrere Teams gemeinsam machen, um voneinander zu lernen und besser koordinieren zu können.

**Sprint Review** kann zusätzlich Personen aus anderen Teams oder andere Stakeholder beinhalten.

**Overall Retrospective** ist ein ganz neues Meeting mit Scrum Mastern, Product Owners und rotierenden Personen der Teams, um den Gesamtprozess zu verbessern.

<sup>6</sup>LeSS Overview Diagram. URL: <https://less.works/img/LeSS-overview-diagram.png> (besucht am 06.06.2018).

### 2.2.1.3 Scrum at Scale<sup>7</sup>

Scrum at Scale versucht Scrum skalierbar zu machen, durch ein Minimum an Bürokratie und eine frei skalierbare Architektur. Es besteht aus Scrum Teams, die durch SoS und Meta Scrums koordiniert werden.

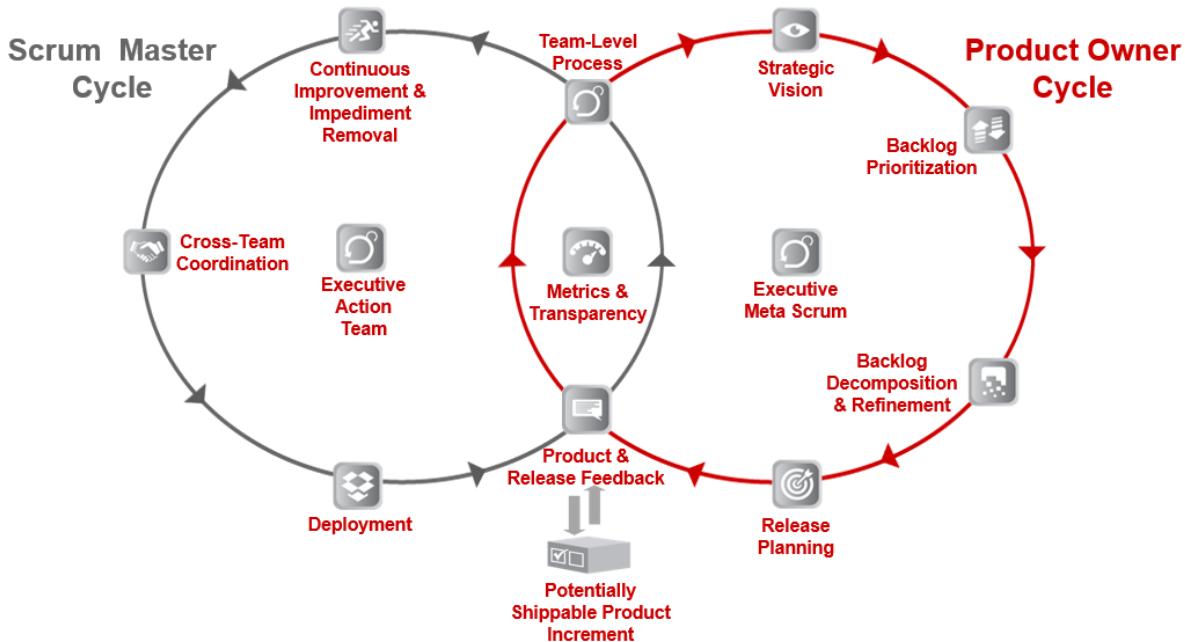


Abbildung 2.5: Scrum at Scale Framework - SMPO Zyklus<sup>8</sup>

Das Was und das Wie werden bei Scrum at Scale durch zwei Zyklen getrennt, die sich nur an zwei Punkten schneiden (Abbildung 2.5).

#### Scrum Master Zyklus

Wird über SoS gesteuert und hat an der Spitze ein Executive Action Team (EAT), welches die einzelnen SoS koordiniert.

#### Product Owner Zyklus

Ein Team von Product Owners, die ein SoS Backlog verantworten, nennt man Meta Scrum. Meta Scrums haben einen Chief Product Owner, der als Scrum Master agiert und verantwortlich für die Koordination der Product Backlogs ist. An der Spitze steht ein Executive Meta Scrum (EMS), der Visionen und strategische Ziele verfolgt.

Metriken und Transparenz im Allgemeinen werden bei Scrum at Scale groß geschrieben, da Scrum nur so optimal funktionieren kann.

<sup>7</sup>vgl. *The Scrum At Scale® Guide*. en-US. URL: <https://www.scrumatscale.com/scrum-at-scale-guide-read-online/> (besucht am 06.06.2018).

<sup>8</sup>*Scrum at Scale Framework / SMPO Cycle*. URL: <https://www.scrumatscale.com/wp-content/uploads/SMPO-Cycle.png> (besucht am 06.06.2018).

## 2.3 Software-Qualität<sup>9</sup>

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-Produktes, 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 Qualitäts- sicherung bereits in der Entwicklung entdeckt oder vermieden werden können.

#### Laufzeit (Performance)

Beschreibt die Umsetzung der Laufzeit-Anforderungen. 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 Benutzerinnen-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 vornehmen zu können. 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.

<sup>9</sup>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 (zum Beispiel 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 durch die möglichen Eingangskombinationen vervielfacht wird. Aber gerade durch Tests können Fehler frühzeitig entdeckt und behoben werden.

Je nach Anwendungsgebiet und Anforderungen an die Software haben die Merkmale unterschiedliche Relevanz und einige können sich auch gegenseitig beeinflussen, wie aus der Korrelationsmatrix in Abbildung 2.6 ersichtlich. Dabei sind die positiv korrelierenden Merkmale mit einem Plus und die negativ korrelierenden mit einem Minus gekennzeichnet.

	Laufzeit	Zuverlässigkeit	Benutzbarkeit	Transparenz	Übertragbarkeit	Wartbarkeit	Testbarkeit
Funktionale Korrektheit	-	+		+	+	+	+
Laufzeit	-		-	-	-	-	
Zuverlässigkeit		+			+		
Benutzbarkeit							
Transparenz				+	+	+	
Übertragbarkeit							
Wartbarkeit							

Abbildung 2.6: Korrelationsmatrix Qualitätskriterien<sup>10</sup>

<sup>10</sup>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.

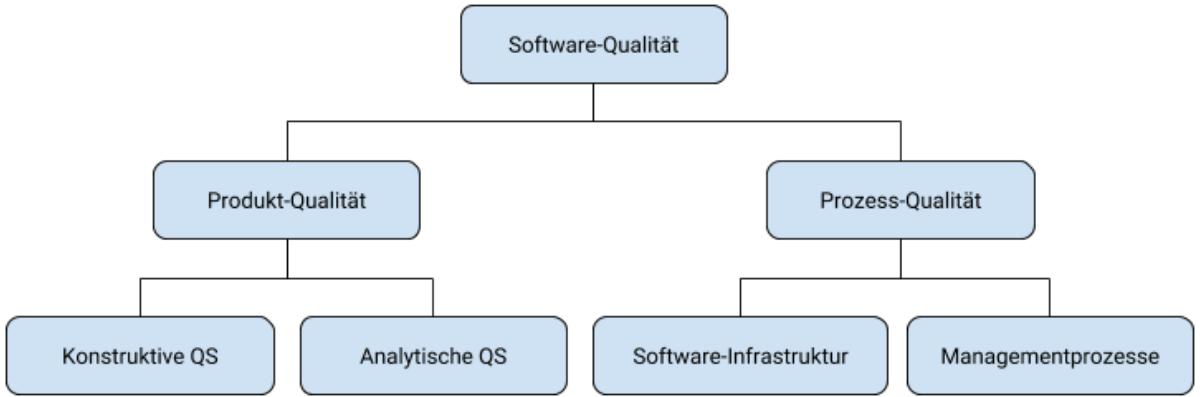


Abbildung 2.7: Software-Qualität

Um die oben genannten Merkmale verbessern zu können, lassen sich die Techniken und Methoden zur Qualitätssicherung, wie in Abbildung 2.7 ersichtlich, in die Bereiche Produkt- und Prozessqualität einteilen, welche im Folgenden noch genauer beschrieben werden.

### 2.3.1 Produktqualität<sup>11</sup>

Die Produktqualität lässt sich in die zwei Bereiche konstruktive und analytische Qualitätssicherung unterteilen. Während die konstruktive Qualitätssicherung sich mit den Vorgaben beschäftigt, die ein Softwareprodukt im Vorhinein (während der Entwicklung) erfüllen muss, dient die analytische Qualitätssicherung dem Analysieren und Bewerten im Nachhinein (des fertigen Produktes).

#### Konstruktive Qualitätssicherung

##### Software-Richtlinien

Eine solche Richtlinie gibt Regeln vor, wie Konstrukte einer Sprache zu verwenden sind. Können unternehmensspezifisch sein, aber auch allgemeinen Standards folgen.

##### Typisierung

Typisierte Sprachen können Inkonsistenzen bereits frühzeitig erkennbar machen.

##### Vertragsbasierte Programmierung

Basiert auf einer gezielten Angabe von Bedingungen (Vor-, Nachbedingungen, Invarianten, Zusicherungen) für Funktionen und Prozeduren, sogenannten Verträgen.

##### Fehlertolerante Programmierung

Fehler in der Software können nie ganz vermieden werden, aber es kann intelligent auf mögliche Fehler reagiert werden.

<sup>11</sup>vgl. Hoffmann, *Software-Qualität*, Kapitel 1.4.1.

### **Portabilität**

Portabler Programmcode ist oft allgemeiner gehalten und kann auch die Transparenz (und somit die Fehlerdichte) positiv beeinflussen.

### **Dokumentation**

Enthält die Erstellung von Standards und das Dokumentieren selbst.

## **Analytische Qualitätssicherung**

### **Software-Test**

Testen des Software-Produktes mit vordefinierten Tests. Die Auswahl dieser Tests beeinflusst dabei direkt die Qualität. Es wird zwischen Black-Box- und White-Box-Tests unterschieden und es können Testmetriken verwendet werden, um die Qualität der Tests zu bestimmen.

### **Statische Analyse**

Hier wird im Gegensatz zu den Software-Tests das Programm nicht ausgeführt, sondern es wird direkt der Quelltext analysiert. Es gibt folgende Möglichkeiten der statischen Analyse:

#### **Software-Metriken**

#### **Konformitätsanalyse**

#### **Exploit-Analyse**

#### **Anomalienanalyse**

#### **Manuelle Software-Prüfung**

### **Software-Verifikation**

Es wird versucht, gewisse Eigenschaften des Programms formal auf mathematischem Wege zu beweisen.

## **2.3.2 Prozessqualität<sup>12</sup>**

Die Prozessqualität beschäftigt sich im Gegensatz zur Produktqualität mit der Entstehung des Software-Produktes, genauer genommen mit dem Prozess dahinter. Dieser Prozess kann in eine Entwicklerinnensicht, mit der Software-Infrastruktur, und eine Managementsicht, mit den Managementprozessen, aufgeteilt werden.

### **Software Infrastruktur**

Unterstützt die Entwicklerin bei der täglichen Arbeit.

### **Konfigurationsmanagement**

Hilft bei der Verwaltung der entstehenden Artefakte, zum Beispiel durch den Einsatz von VCS.

### **Build-Automatisierung**

Erzeugt die Artefakte voll automatisch.

---

<sup>12</sup>vgl. Hoffmann, *Software-Qualität*, Kapitel 1.4.2.

### **Test-Automatisierung**

Genauso, wie das Erstellen der Artefakte, wird auch das Testen vollautomatisch durchgeführt.

### **Defektmanagement**

Um Defekte zentral erfassen und verwalten zu können.

### **Managementprozesse**

Steuern den Projektablauf und werden in Vorgehensmodelle und Reifegradmodelle unterteilt.

#### **Vorgehensmodelle**

Regeln die grundlegenden Arbeitsabläufe in einem Projekt, zum Beispiel über das V-Modell oder Scrum.

#### **Reifegradmodelle**

Haben das Ziel, Prozesse zu analysieren und zu optimieren.

## 2.4 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.”<sup>13</sup>

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

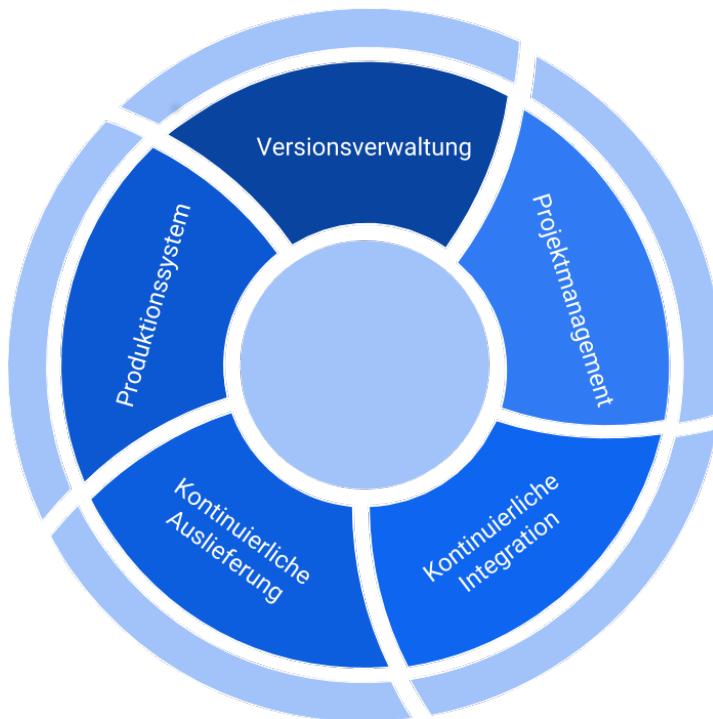


Abbildung 2.8: Systeme im Softwareentwicklungsprozess

Im Entwicklungsprozess werden in den unterschiedlichen Systemen und Prozessschritten Daten erzeugt, die als Kennzahlen oder direkt als Metriken genutzt werden können. Abbildung 2.8 zeigt die einzelnen Schritte und Systeme im Entwicklungsprozess.<sup>14</sup>

<sup>13</sup>vgl. „IEEE Standard for a Software Quality Metrics Methodology“. In: *IEEE Std. 1061-1998* (1998), S.3.

<sup>14</sup>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.8.

## 2.4.1 Versionsverwaltung<sup>15</sup>

Das Version Control System (VCS) befindet sich nah an der Arbeit der Entwicklerinnen, da hier der Quellcode des Produktes 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 Codezeilen.

### **CLOC pro Entwicklerin**

Anzahl der geänderten Zeilen im Quellcode pro Entwicklerin.

### **Commits**

Gesamtzahl an Commits in einem bestimmten Zeitraum.

### **Commits pro Entwicklerin**

Gesamtzahl an Commits in einem bestimmten Zeitraum pro Entwicklerin.

### **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 einem bestimmten Zeitraum.

### **Gemergte Pull Requests**

Anzahl erfolgreicher Pull Requests in einem bestimmten Zeitraum.

### **Abgelehnte Pull Requests**

Anzahl abgelehnter Pull Requests in einem bestimmten Zeitraum.

### **Kommentare pro Pull Request**

Anzahl der Kommentare pro Pull Request.

## 2.4.2 Projektmanagement<sup>16</sup>

In einem Project Tracking System (PTS), oft auch Issue Tracker oder Issue Tracking System genannt, 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”)

<sup>15</sup>vgl. Davis, *Agile Metrics in Action*, S.62ff.

<sup>16</sup>vgl. Davis, *Agile Metrics in Action*, S.37ff.

- Aufgaben schätzen
- gemeinsam eine DoD festlegen

Jede Arbeit, die am PTS vorbei geht, fällt später bei der Auswertung der Daten durch das Raster. Durch das Kategorisieren der Aufgaben können später Korrelationen ausgewertet werden, vor allem auch durch das Zuweisen von Tags wird sichtbar, wie gut die Aufgabe abgelaufen ist. Zusätzlich kann durch die geschätzte und tatsächliche Arbeitszeit eine qualitative Aussage über die Schätzungen des Teams getroffen werden. Eine 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 erledigter Arbeit über die Zeit. Liefert einen Richtwert, wo man sich gerade im Sprint befindet, verglichen zum optimalen Verlauf.

### **Velocity**

Eine relative Messung der Konsistenz erledigter Arbeit über die Sprints.

### **Cumulative 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. 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.

## **2.4.3 Kontinuierliche Integration und Auslieferung<sup>17</sup>**

Continuous Integration (CI)- und Continuous Delivery (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<sup>18</sup>). Diese Systeme sind aber auch jene Elemente im Softwareentwicklungsprozess, die von Team zu Team am meisten variieren können.

<sup>17</sup>vgl. Davis, *Agile Metrics in Action*, S.84ff.

<sup>18</sup>*Continuous Code Quality / SonarQube*. URL: <https://www.sonarqube.org/> (besucht am 05.01.2018).

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 gegenübergestellt 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 getestet mit Tools wie JMeter<sup>19</sup> oder Gatling<sup>20</sup>.

### **2.4.4 Produktionssystem<sup>21</sup>**

Daten aus den Produktionssystemen können gesammelte Application-Performance-Monitoring (APM)- oder auch Business-Intelligence (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<sup>22</sup> und Atlas<sup>23</sup> 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.

#### **Benutzerinnenanzahl**

Anzahl gleichzeitiger Benutzerinnen in der Applikation über die Zeit.

<sup>19</sup> Apache JMeter - Apache JMeter™. URL: <https://jmeter.apache.org/> (besucht am 29.03.2018).

<sup>20</sup> Gatling Load and Performance testing - Open-source load and performance testing. en-US. URL: <https://gatling.io/> (besucht am 29.03.2018).

<sup>21</sup>vgl. Davis, Agile Metrics in Action, S.107ff.

<sup>22</sup>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).

<sup>23</sup>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).

### **Aufenthaltsdauer**

Verweildauer der Benutzerinnen auf bestimmten Seiten.

### **Conversion Rate**

Anzahl Benutzerinnen, die zu Kunden wurden.

### **Semantisches Logging**

Ermöglicht es, beim Logging strukturierte Daten auszugeben, zum Beispiel: was suchen Benutzerinnen 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 dazugehörigen Fragen, die sie jeweils beantworten, im Anhang A.1.

## **2.4.6 Eigene Metriken erstellen<sup>24</sup>**

Um eigene Metriken erstellen zu können, sind zwei 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, also nur Metriken, die einen bestimmten Zweck erfüllen und einem Ziel folgen
- dass die Metrik für sich alleine stehen kann

## **2.4.7 Veröffentlichung von Metriken<sup>25</sup>**

Metriken können auf verschiedene Art und Weise veröffentlicht werden. Zwei mögliche Beispiele sind Dashboards oder E-Mails. 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

---

<sup>24</sup>vgl. Davis, *Agile Metrics in Action*, S.127ff.

<sup>25</sup>vgl. Davis, *Agile Metrics in Action*, S.177ff.

- Metriken werden als Werkzeug gesehen, nicht als Waffe (gegen andere Teams oder Personen)
- Page Tracking verwenden, um das Nutzungsverhalten zu verstehen

## E-Mails

- aus dem Dashboard als Option wählbar machen (sonst landen sie schnell automatisch im Spam-Ordner)
- minimal erforderliche Daten, den Rest verlinken zum Dashboard
- den richtigen Rhythmus finden (zwischen oft genug informieren und nerven)

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

## 2.4.8 Agile Prinzipien messen<sup>26</sup>

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 2.9 ersichtlich, erreicht werden.



Abbildung 2.9: Agile Prinzipien als Wortwolke

<sup>26</sup>vgl. Davis, *Agile Metrics in Action*, S.201ff.

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<sup>27</sup>:

### **Effektive Software**

- erfolgreiche / fehlerhafte Builds
- Business-Metriken
- Status der Applikation
  - Fehlerraten
  - CPU- / Speicherauslastung
  - Antwort- / Transaktionszeiten
  - Heap Größe / Garbage Collection / Anzahl Threads

### **Effektiver Prozess**

- Velocity
- PTS und VCS Kommentare
- erfolgreiche Releases

### **Effektives Team**

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

### **Effektive Anforderungen**

- Rückläufigkeit
- Lead Time
- MTTR
- Velocity

---

<sup>27</sup>vgl. Davis, *Agile Metrics in Action*, S.219ff.

## 2.5 Qualitätsmodelle<sup>28</sup>

Qualitätsmodelle sind das Fundament für Software-Produkt-Qualitätskontrolle und werden genutzt, um Qualität zu beschreiben, zu schätzen und/oder vorherzusagen. Es wurden in den letzten Jahrzehnten unzählige Modelle vorgestellt. Diese lassen sich in drei Kategorien einteilen: hierarchische, metamodellbasierte und implizite Modelle.

### Hierarchische Qualitätsmodelle

Entstanden bereits in den 1970er Jahren. Qualität wird hierarchisch in Qualitätsfaktoren zerlegt, wie Wartbarkeit oder Zuverlässigkeit. Die Idee dahinter ist, Qualität hierarchisch herunterzubrechen, damit sie messbar wird. Verschiedenste Kritiken heben hervor, dass die Prinzipien zur Dekomposition von Qualitäts-Charakteristiken oft mehrdeutig sind. Außerdem sind die resultierenden Qualitäts-Charakteristiken nicht spezifisch genug, um direkt gemessen werden zu können.

### Metamodellbasierte Qualitätsmodelle

Entstanden in den 1990er Jahren, als Forscher ausgereiftere Wege aufzeigten, um Qualitäts-Charakteristiken zu zerlegen. Dadurch entstanden auch ausführlichere Metamodelle. Ein Metamodell beschreibt, wie gültig Qualitätsmodelle strukturiert sind. Die vielen Metamodelle zeigen, dass Qualität mehr Struktur in Qualitätsmodellen, als in abstrakten Qualitäts-Charakteristiken und Metriken braucht. Es wurde kein generelles Basis-Qualitätsmodell eingeführt, das man herunterladen und anwenden kann.

### Statistische und implizite Qualitätsmodelle

Für unterschiedlichste Qualitätsfaktoren wurden statistische Modelle vorgestellt, die Eigenschaften erfassen und Qualitätsfaktoren schätzen oder voraussagen. Auch Qualitäts-Analysetools nutzen eine Art Qualitätsmodell und auch Checklisten in der Entwicklung oder in Reviews sind eine Art Qualitätsmodell.

### 2.5.1 Factor Criteria Metrics (FCM)-Modell<sup>29</sup>

Beim FCM-Qualitätsmodell, welches zu den hierarchischen Qualitätsmodellen gehört, werden zunächst die erforderlichen Qualitätsmerkmale über die Anforderungen an die Software ermittelt. In einem weiteren Schritt werden die Qualitätsmerkmale in kleinere Teilmerkmale aufgeteilt. Dieser Schritt kann über mehrere Ebenen weitergeführt werden, bis es möglich ist, das Teilmerkmal direkt über eine oder mehrere Metriken abzubilden.

<sup>28</sup>vgl. Stefan Wagner. *Software Product Quality Control*. Englisch. 2013. Aufl. New York: Springer, Aug. 2013. ISBN: 978-3-642-38570-4, S.29ff.

<sup>29</sup>vgl. Dietmar Abts und Wilhelm Mülder. *Masterkurs Wirtschaftsinformatik: Kompakt, praxisnah, verständlich - 12 Lern- und Arbeitsmodule*. de. Springer-Verlag, Okt. 2009. ISBN: 978-3-8348-0002-2, S.668ff.

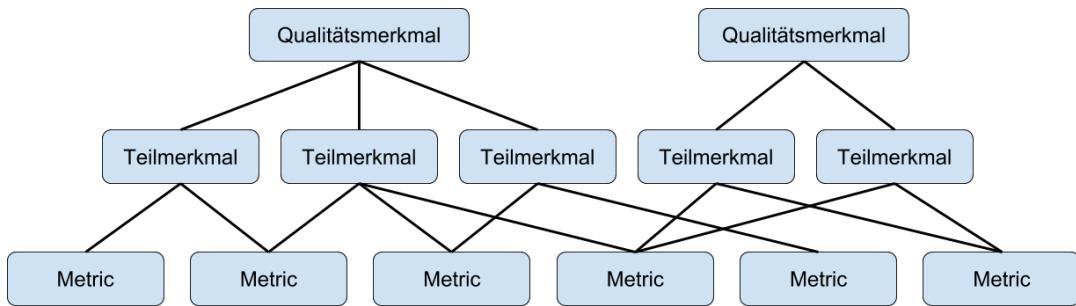


Abbildung 2.10: Struktur des FCM Modells

Abbildung 2.10 zeigt einen solchen Aufbau eines FCM-Modells.

## 2.6 Goal Question Metric (GQM)<sup>30</sup>

GQM ist eine Methodik, die dabei hilft, geeignete Metriken für ein Softwareprojekt finden zu können und wurde ursprünglich von der National Aeronautics and Space Administration (NASA) entwickelt, um Fehler in bestimmten Projekten zu erkennen. Der grundlegende Gedanke dahinter ist, dass Metriken “top-down” (von oben nach unten) definiert werden müssen. Dieser Ansatz wird dadurch begründet, dass Metriken sehr viele Charakteristiken abbilden können, aber erst durch Modelle beziehungsweise Ziele richtig genutzt und interpretiert werden können.

Das Ergebnis dieses Modells hat drei Level:

### GOAL - konzeptuelles Level

Es werden Ziele für bestimmte Objekte definiert. Objekte können Produkte, Prozesse oder Ressourcen sein.

### QUESTION - operatives Level

Für jedes Ziel werden Fragen formuliert, die zur Beurteilung oder Erreichung beitragen. Sie versuchen den Grund für die Messung zu charakterisieren.

### METRIC - quantitatives Level

Jeder Frage werden Metriken zugeordnet, die dabei helfen sollen, sie quantitativ zu beantworten.

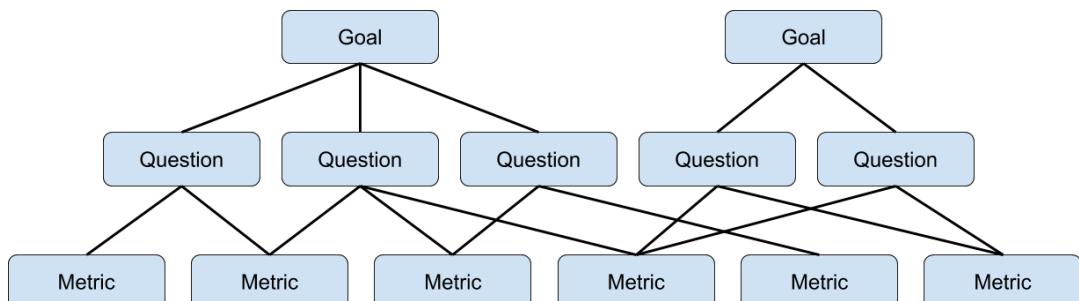


Abbildung 2.11: Struktur eines Ergebnisses der GQM-Methodik

Abbildung 2.11 zeigt die hierarchische Struktur eines Ergebnisses der GQM-Methodik.

### Beispiel<sup>31</sup>

Anhand des Beispiels in Tabelle 2.1 soll veranschaulicht werden, wie so ein Ergebnis der GQM-Methodik in der Praxis aussehen kann. Angenommen wird, das Team will seine Zusammenarbeit verbessern. Dazu muss das Ziel folgende Punkte spezifizieren: Absicht, Prozess / Produkt / Ressource, Sichtweise und Problem. Dieses Ziel kann anschließend durch Fragen verfeinert werden. Aussagen über Zusammenarbeit geben zum Beispiel

<sup>30</sup>Victor R Basili, Gianluigi Caldiera und H Dieter Rombach. *THE GOAL QUESTION METRIC APPROACH*. en. URL: <http://www.cs.umd.edu/~mvz/handouts/gqm.pdf>.

<sup>31</sup>Basili, Caldiera und Rombach, *THE GOAL QUESTION METRIC APPROACH*.

Pull Requests. Diese Fragen wiederum können mit Metriken beantwortet werden. Im Falle der Pull Requests zum Beispiel der Durchschnitt und die Standardabweichung der Kommentare pro Pull Request in jedem Sprint.

	Absicht	Verbessern
GOAL	Problem	der Zusammenarbeit innerhalb des Teams
	Prozess	im Entwicklungsprozess
	Sichtweise	aus Sicht der Entwicklerinnen.
QUESTION	Wie arbeitet das Team mit Pull Requests?	
	Anzahl Pull Requests	
METRIC	Anzahl Kommentare pro Pull Request *	
	Anzahl Entwicklerinnen pro Pull Request *	
QUESTION	Wie viele Entwicklerinnen arbeiten an den einzelnen Modulen?	
METRIC	Anzahl Commits pro Entwicklerin pro Modul	
	Anzahl Entwicklerinnen pro Pull Request pro Modul *	

\* Durchschnitt und Standardabweichung pro Sprint

Tabelle 2.1: Beispiel GQM-Methodik

# 3 Vorgehensweise

In diesem Kapitel wird erklärt, wie bei der Erarbeitung der Lösung vorgegangen wird. Zuerst werden die benötigten Metriken ermittelt. Dazu werden die Daten der letzten Retrospektiven analysiert und eine Umfrage im Team gemacht. Diese Metriken werden dann in einer Software regelmäßig erzeugt, gespeichert und angezeigt. Zuletzt wird das Ergebnis der Arbeit qualitativ und quantitativ evaluiert.

## 3.1 Metriken bestimmen

Das Unternehmen, in dem die Tests durchgeführt werden, arbeitet seit rund einem Jahr nach dem Scrum Framework. Als Basis zur Bestimmung der Metriken können daher die vorhandenen Retrospektiven genutzt werden, da diese eine Richtung vorgeben, in die sich das Team bewegen will. Es können die meistgenutzten Schlagwörter aus den Fragestellungen der Retrospektive ermittelt werden und mithilfe der GQM-Methodik in Metriken abgebildet werden. Das FCM-Modell ist hier weniger geeignet, da die Teams in dem Unternehmen, in dem die Software getestet wird, nicht an einem Produkt arbeiten, sondern an Geschäftsprozessen. Außerdem wollen sie den Scrum-Prozess verbessern, nicht nur ein Produkt.

Als Ergänzung zur GQM-Methodik wird zusätzlich das Team in einer Umfrage zu möglichen Metriken befragt. Dabei werden die in dieser Arbeit erarbeiteten Metriken genauer vorgestellt und von den Teammitgliedern bewertet. Dadurch soll zusätzlich die Möglichkeit gegeben werden, Metriken oder Probleme, die zuvor nicht genannt wurden, angeben zu können. Außerdem können die Ergebnisse der GQM-Methodik mit denen der Umfrage verglichen werden.

## 3.2 Software

Im Rahmen dieser Arbeit wird eine Software entwickelt, die Qualitätsmetriken aus unterschiedlichen Systemen ermitteln und bereitstellen kann. Dabei sollen die zuvor ermittelten Metriken automatisch erzeugt und gespeichert werden.

### 3.2.1 Anforderungen

Die Anforderungen an die Software entstanden zum einen aus der gewünschten Funktionsweise und zum anderen aus den Gegebenheiten des Umfelds der Software (und dem Unternehmen, in dem sie getestet werden soll).

#### Erweiterbarkeit

Um einfach neue Systeme und Metriken bereitstellen zu können, muss bei der Architektur auf eine einfache Erweiterbarkeit geachtet werden.

#### Fehlertoleranz

Ein Fehler in einem einzelnen System, das Daten bereitstellt, darf nicht zum Absturz der Software führen.

#### Qualitätssicherung

Die Qualitätssicherung muss nach aktuellen Standards erfolgen, das bedeutet: Einzelne Komponenten der Software sollten eine möglichst hohe Testabdeckung aufweisen. CI mit statischer Codeanalyse muss eingerichtet werden.

#### Dokumentation

Es muss auf eine möglichst verständliche und vollständige Dokumentation geachtet werden, um die Möglichkeit der Weiterentwicklung offen zu lassen.

#### Umsetzung in Java

Java ist in vielen Unternehmen verbreitet und stößt daher auf eine hohe Akzeptanz.

#### einzbindende Systeme

Metriken können in BitBucket Server<sup>32</sup>, JIRA<sup>33</sup>, Jenkins<sup>34</sup>, SonarQube<sup>35</sup> oder Icinga<sup>36</sup> vorkommen. Systeme, in denen relevante Metriken ermittelt wurden, müssen unterstützt werden.

#### Speicherung und Darstellung

Speicherung und Darstellung der Metriken erfolgt in einem Elastic Stack<sup>37</sup>.

---

<sup>32</sup>Atlassian. *Bitbucket Server*. en. URL: <https://www.atlassian.com/software/bitbucket/server> (besucht am 31.03.2018).

<sup>33</sup>Atlassian. *Jira / Software zur Vorgangs- und Projektverfolgung*. de-DE. URL: <https://de.atlassian.com/software/jira> (besucht am 31.03.2018).

<sup>34</sup>*Jenkins*. URL: <https://jenkins.io/index.html> (besucht am 31.03.2018).

<sup>35</sup>*Continuous Code Quality / SonarQube*.

<sup>36</sup>*Icinga*. en-US. URL: <https://www.icinga.com/> (besucht am 31.03.2018).

<sup>37</sup>*Elastic Stack*. de-de. URL: <https://www.elastic.co/de/products> (besucht am 31.03.2018).

### **3.3 Evaluierung**

Nach der Fertigstellung der Software wird diese für Testzwecke in einem Unternehmen für mehrere Sprints eingesetzt, um über einen möglichst langen Zeitraum Metriken zu sammeln. Parallel dazu wird den betroffenen Teammitgliedern der Umgang mit dem Dashboard näher gebracht.

Die Evaluierung wird zum Einen quantitativ durchgeführt, da sich Metriken sehr gut dafür eignen. Allerdings kann es sein, dass die Zeit, die für den Test zur Verfügung steht, einfach zu kurz ist, um eine Tendenz in den Metriken erkennen zu können.

Daher wird zum Anderen noch eine qualitative Evaluierung durchgeführt, in Form von Interviews mit ausgewählten Teammitgliedern. Dies dient der Ermittlung der Effektivität der eingesetzten Methoden und der Erfragung des subjektiven Empfindens gegenüber der Nützlichkeit von Metriken im agilen Prozess.

# 4 Umsetzung

Dieses Kapitel zeigt die erarbeitete Lösung im Detail. Zu Beginn werden die relevanten Metriken mithilfe der GQM-Methodik und durch eine Umfrage im Team, identifiziert. Dann wird genauer auf die Architektur der Software und deren Entwicklung eingegangen. Abgeschlossen wird das Kapitel mit der Inbetriebnahme der fertigen Software.

## 4.1 Gegebenheiten

Die Umsetzung findet in einem Unternehmen mit weltweit rund 150 Standorten und 6700 Mitarbeitern (Stand Mai 2018) statt. Es gibt eine zentrale Informatik mit einer eigenen Softwareentwicklungsabteilung, in der vier Scrum-Teams angesiedelt sind. Diese vier Scrum Teams koordinieren sich über SoS. Es wurde gezielt ein bestimmtes Scrum-Team zur Umsetzung gewählt, weil dieses aus Sicht des Autors den Scrum-Prozess bereits am weitesten entwickelt hat und dadurch Werkzeuge zur Qualitätssicherung dort am meisten Sinn machen. Aus sicherheitstechnischen Gründen wird auf das Unternehmen in dieser Arbeit nicht weiter eingegangen.

## 4.2 Metriken Identifizieren

### 4.2.1 GQM

Um eine Vorauswahl an Metriken treffen zu können, wurden alle bisherigen Retrospektiven analysiert und eine Topliste von Schlagwörtern der folgenden Fragestellungen erstellt:

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

Dazu wurden die Ergebnisse in einer ElasticSearch Datenbank gespeichert und über eine sogenannte Terms Aggregation die wichtigsten Schlagwörter analysiert. Bei der Indexierung werden die Wörter normalisiert, deshalb die teilweise andere Schreibweise (zum Beispiel wird aus Issue der Term issu). Die Ergebnisse in Anhang A.2 sind für die einzelnen Punkte wie folgt interpretierbar:

**Welche guten Entscheidungen haben wir getroffen?**

- Die ersten fünf Wörter lassen darauf schließen, dass das Team gut zusammenarbeitet, speziell bei der Wissensverteilung: Transparenz, Onboarding von neuen Themen und Pair-Programming.
- Ebenfalls lässt sich aus den weiteren Begriffen schließen, dass viel Wert auf Reviews, Daily Scrum und die Arbeitsweise an sich gelegt wird.

### **Was haben wir gelernt?**

- Auch hier spiegelt sich der Daten- und Kommunikationsfluss im Team wider.
- Die vielen Scrum-Schlagwörter zeigen, dass die Retrospektiven richtig genutzt wurden, um den Prozess zu verbessern.

### **Was können wir besser machen?**

- Auch hier zeugen die Scrum-Schlagwörter wieder von einer Prozessverbesserung und einem selbstreflektiven Verhalten.
- Die Dokumentation scheint teilweise noch ein Problem zu sein, diese kommt gleich zweimal vor.
- Issues scheinen teilweise nicht optimal zu sein. Da könnte das Schlagwort „groß“ dazu passen.
- “backlog”, “blocked” und “ablauf” lassen auf Probleme im Arbeitsablauf schließen.

### **Was beschäftigt uns noch immer?**

- Hier lassen die Schlagwörter “updat”, “erreichbar”, “infrastruktur”, “jenkin”, “test” und “umgebung” auf ein Infrastrukturproblem schließen, welches das Team womöglich ausbremst.
- Die Schlagwörter “lang”, “groß” und “klar” lassen auf Probleme mit Anforderungen beziehungsweise Stories schließen.
- “apis”, “dba”, “laut” und “iso” sind wahrscheinlich äußere Einflüsse, die bei der täglichen Arbeit stören.

Aus diesen Ergebnissen lassen sich die GQM-Ergebnisse in Tabelle 4.1, 4.2 und 4.3 ableiten.

<b>GOAL</b>	Absicht Problem Ressource Sichtweise	Verringerung der Ablenkung von Entwicklerinnen aus Sicht des Scrum-Masters.
<b>QUESTION METRIC</b>	Wie viele Aufgaben erledigt das Team pro Sprint? Aufgaben-Volumen pro Sprint	
<b>QUESTION METRIC</b>	Wie viele Aufgaben werden jeden Tag erledigt? erledigte Aufgaben pro Tag Burn-down pro Tag	
<b>QUESTION METRIC</b>	Welche Tags haben als "schlecht" bewertete Aufgaben? Tags der Aufgaben, um "schlecht" bewertete besser analysieren zu können	

Tabelle 4.1: GQM-Ergebnis - Ablenkung der Entwicklerinnen

<b>GOAL</b>	Absicht Problem Prozess Sichtweise	Optimierung des Durchlaufes im Entwicklungsprozess aus Sicht des Scrum Teams.
<b>QUESTION METRIC</b>	Gibt es irgendwelche Engpässe im Prozess? Cumulative Flow	
<b>QUESTION METRIC</b>	Wie lange dauert der Durchlauf einer Aufgabe? Lead Time	

Tabelle 4.2: GQM-Ergebnis - Schwachstellen im Prozess

<b>GOAL</b>	Absicht Problem Ressource Sichtweise	Optimierung der Aufgabengröße im Backlog aus Sicht des Product Owners.
<b>QUESTION METRIC</b>	Wie groß ist die Aufgabengröße im Durchschnitt? erledigte Story-Points / Aufgaben-Volumen	
<b>QUESTION METRIC</b>	Wie lange dauert der Durchlauf einer Aufgabe? Lead-Time	
<b>QUESTION METRIC</b>	Wie viele Aufgaben gehen im Entwicklungsprozess rückwärts? Aufgaben-Rückfälligkeit	

Tabelle 4.3: GQM-Ergebnis - Aufgabengröße

#### 4.2.2 Umfrage im Team

Zusätzlich zu der Analyse der Retrospektiven wurden dem Team Metriken und die Fragen, die damit beantwortet werden können, vorgestellt und deren Meinung in Form einer Befragung erhoben. Die einzelnen Metriken wurden von den Teammitgliedern nach Wichtigkeit mit einer Skala von eins bis zehn bewertet. Anhang A.3.1 zeigt die Umfrage, wie sie den Teammitgliedern vorgelegt wurde und Anhang A.3.2 die dazugehörigen Antworten. Die Ergebnisse wurden nach ihrem Durchschnittswert sortiert und die zehn als am wichtigsten bewerteten Metriken sind:

- **Burn Down (9,00)** - Erfüllt das Team seine Commitments? Plant das Team seine Arbeit realistisch?
- **Velocity (9,00)** - Wie konsistent arbeitet das Team?
- **Aufgaben-Volumen (8,57)** - Wie viel ungeplante Arbeit kam zum Sprint dazu? Wie groß ist die durchschnittliche Aufgabe? Gibt es Ausreißer?
- **Cumulative Flow (8,29)** - Gibt es Engpässe oder Schwachstellen im Prozess? Müssen gewisse Abläufe im Prozess optimiert werden?
- **Lead Time (8,14)** - Wie schnell können Aufgaben vom Team erledigt werden? Wie lange dauert die Umsetzung eines neuen Features?
- **Stresstests oder Benchmarking (7,86)** - Ist das Produkt auch noch unter Last verwendbar? Wie verändert sich die Leistung über die Zeit?
- **Code Coverage (7,71)** - Gibt es Module, die nicht oder schlecht getestet sind? Wie sieht die Entwicklung der Testabdeckung über die Zeit aus?
- **Bug Counts (7,57)** - Wie viele Fehler werden vom Team im Entwicklungsprozess übersehen? Wie viel ungeplante Arbeit kam zum Sprint dazu?
- **Aufgaben-Rückfälligkeit (7,57)** - 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)?
- **Bug-Erzeugungsrate (7,43)** - Wie viele Fehler wurden zu einem bestimmten Zeitpunkt erzeugt?

Außerdem wurde in den offenen Fragen am Ende zweimal gefordert, dass die Flüchtigkeit von Anforderungen sichtbar werden soll. Dies kann zum Einen durch die oben genannte Aufgaben-Rückfälligkeit und andererseits durch folgende Metrik abgebildet werden:

- **Anforderungs-Flüchtigkeit** - Wie oft wurde die Anforderung der Aufgabe angepasst?

## 4.3 Software

Das für die Umsetzung gewählte Unternehmen setzt bereits auf Java als bevorzugte Programmiersprache und hat die in Kapitel 3.2.1 genannten Systeme im Einsatz. Dadurch sind die Plattform und die zu unterstützenden Systeme vorgegeben. Abbildung 4.1 zeigt die Position in einer möglichen Systemlandschaft.

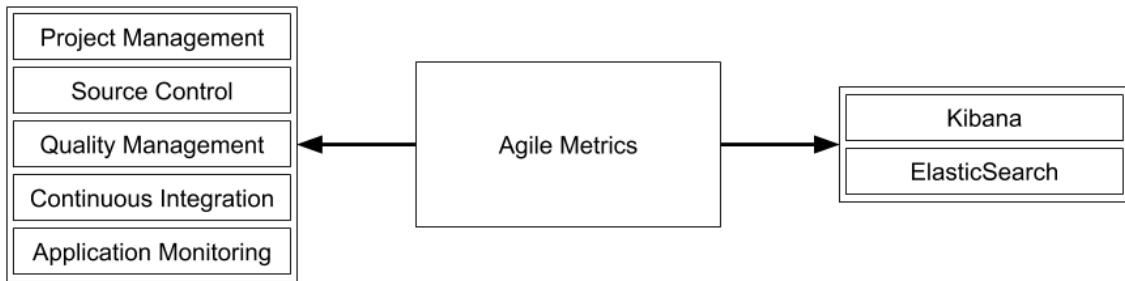


Abbildung 4.1: Position der Software

### 4.3.1 Architektur

Abbildung 4.2 zeigt das Architekturkonzept der Software (Agile Metrics). Diese bildet eine Schnittstelle zwischen den einzelnen Systemen des Entwicklungsprozesses und dem System zur Darstellung der Metriken (in diesem Fall der sogenannte Elastic Stack<sup>38</sup> mit Elasticsearch und Kibana). Elasticsearch ist dabei die Datenbank für die Metriken, genaugenommen eine volltext-indizierte Not-only-SQL (NoSQL)-Datenbank. Kibana dient zur Darstellung der in der Elasticsearch-Datenbank gespeicherten Metriken.

---

<sup>38</sup>Elastic Stack.

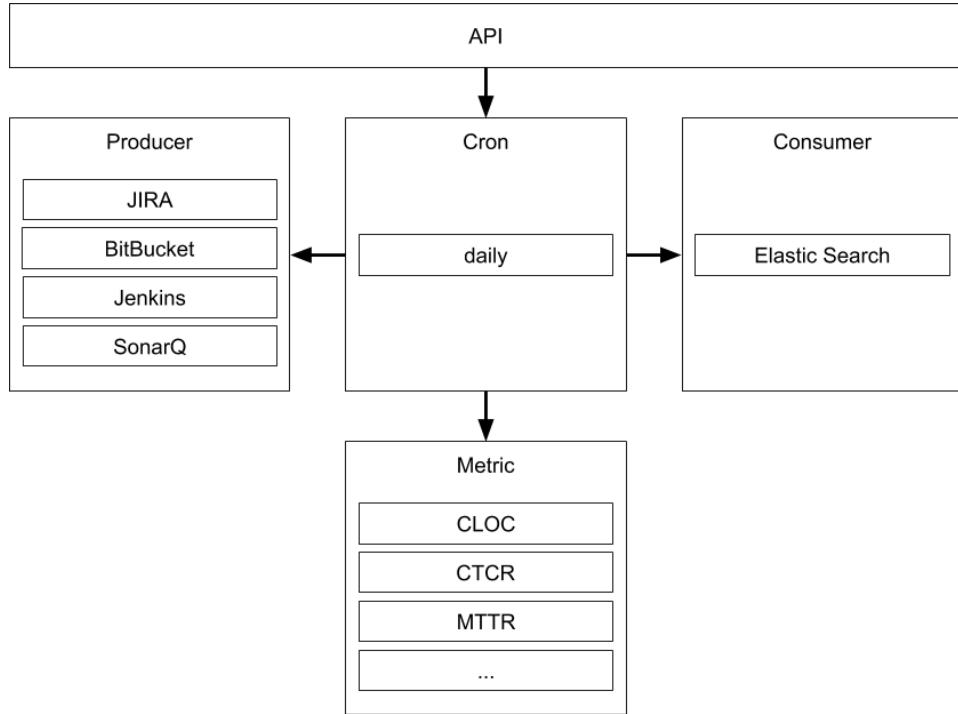


Abbildung 4.2: Übersicht der Software-Architektur

### API

Bietet eine RESTful Schnittstelle zum Abfragen des aktuellen Status.

### Producer

Sind Schnittstellen zu allen Systemen, die Messdaten erzeugen.

### Cron

Zeitsteuerung der Messdaten-Abfrage und Koordination der Erzeugung und Konsumierung von Metriken.

### Metric

Metriken, welche aus den Messdaten erzeugt und konsumiert werden.

### Consumer

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

### 4.3.2 Lizenz

Bei der Recherche wurde keine Software gefunden, welche die oben genannten Funktionalitäten anbietet. Daher wird die im Zuge dieser Arbeit erstellte Software quelloffen angeboten. Als Lizenz wurde dabei die MIT-Lizenz<sup>39</sup> gewählt, da sie eine einfache und großzügige Lizenz ist, die lediglich die Erhaltung des Urheberrechts und eine Lizenzangabe erfordert.

<sup>39</sup>MIT Lizenz. URL: <https://choosealicense.com/licenses/mit/> (besucht am 31.05.2018).

### 4.3.3 VCS, CI und QS

Da es sich um eine quelloffene Software handelt, wird als VCS ein öffentliches Repository<sup>40</sup> in GitHub verwendet. In diesem sind der Quellcode und die Veröffentlichungen der Software frei zugänglich. Zusätzlich wird über GitHub Pages eine statische Seite<sup>41</sup> für das Projekt bereitgestellt, wie in Abbildung 4.3 ersichtlich.



#### Agile Metrics



#### Overview

Agile Metrics is a collector for software development process KPI data. It collects measurements from [Producers](#), creates metrics and sends them to [Consumers](#).

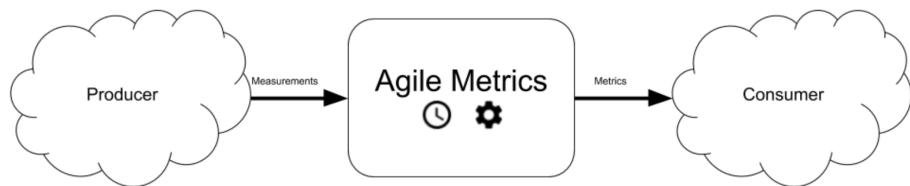


Abbildung 4.3: GitHub Pages - Agile Metrics

Travis CI<sup>42</sup> bietet einen kostenlosen CI-Service und SonarCloud<sup>43</sup> einen kostenlosen QS-Service für quelloffene Projekte.

<sup>40</sup> GitHub Repository - Agile Metrics. URL: <https://github.com/DaGrisa/agile-metrics> (besucht am 31.05.2018).

<sup>41</sup> GitHub Pages - Agile Metrics. URL: <https://dagrisa.github.io/agile-metrics/> (besucht am 31.05.2018).

<sup>42</sup> Travis CI - Agile Metrics. URL: <https://travis-ci.org/DaGrisa/agile-metrics> (besucht am 31.05.2018).

<sup>43</sup> SonarCloud - Agile Metrics. URL: <https://sonarcloud.io/dashboard?id=at.grisa.agile-metrics%3Aagile-metrics> (besucht am 31.05.2018).



Abbildung 4.4: Travis CI - Agile Metrics

Nach jedem Commit in das GitHub Repository wird ein Build gestartet, um kontinuierliche Integration zu gewährleisten. Abbildung 4.4 zeigt die Liste der zu diesem Zeitpunkt zuletzt durchgeföhrten Builds mit einigen zusätzlichen Informationen.

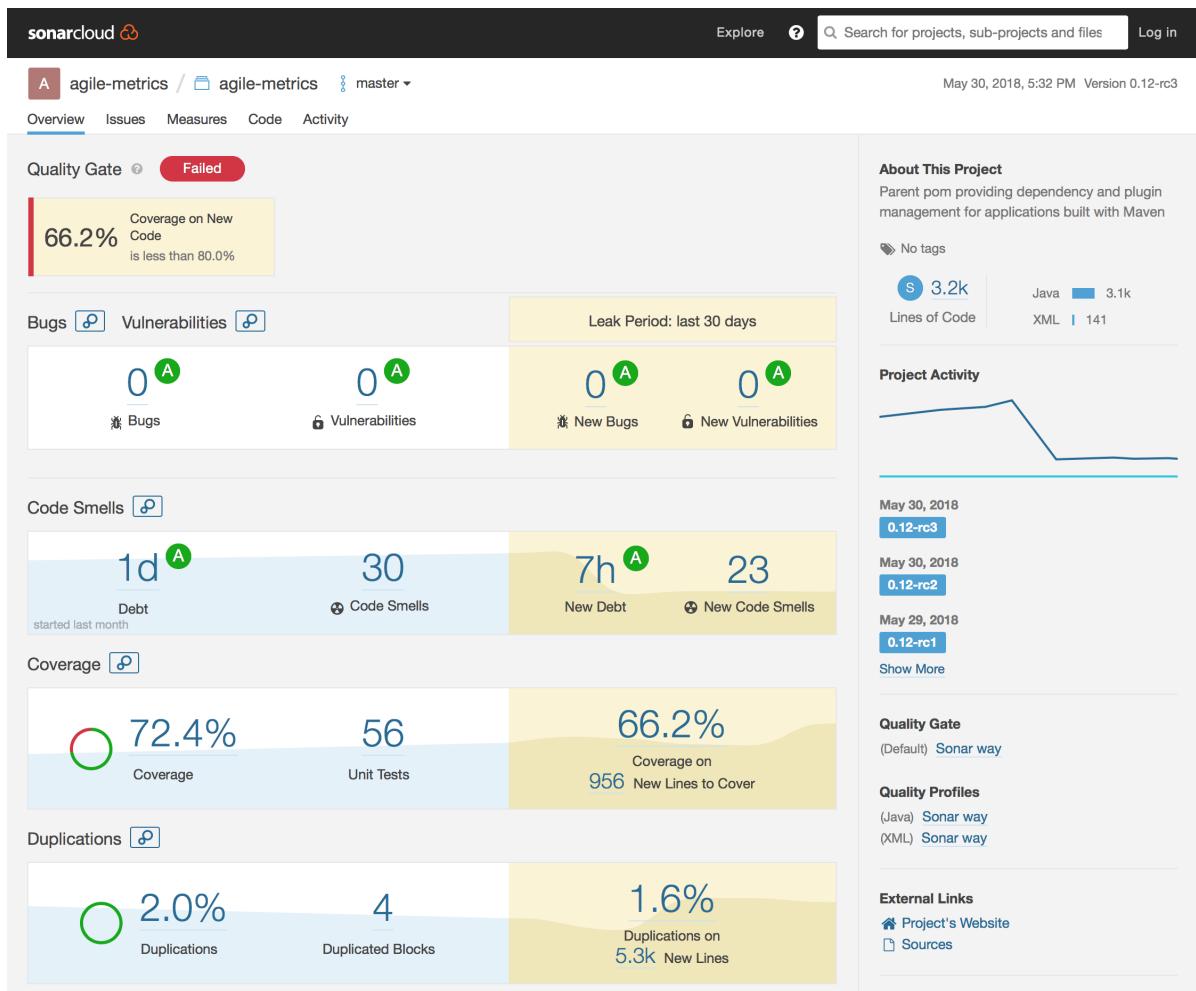


Abbildung 4.5: SonarCloud - Agile Metrics

Jeder Build in Travis CI sendet mit Hilfe eines Addons Informationen über die Qualität des Quellcodes an SonarCloud. Abbildung 4.5 zeigt die Übersichtsseite von SonarCloud mit den wichtigsten Qualitätsmetriken für das Projekt.

## 4.4 Inbetriebnahme

Betrieben wird die Software auf einem eigens für Java-Applikationen bereitgestellten Server. Wichtig dabei ist, dass alle benötigten Systeme erreichbar sind. Es wird auch ein Web-Proxy unterstützt, dieser kann in den Einstellungen konfiguriert werden. Folgende Schritte sind notwendig, um Agile-Metrics in Betrieb zu nehmen:

**Neuestes Release** aus dem Repository auf GitHub herunterladen (zu diesem Zeitpunkt aktuellster Stand ist v1.0) und im gewünschten Verzeichnis ablegen, in dem es später gestartet soll.

**Einstellungen** in der Konfigurationsdatei (application.properties) vornehmen, wie in der Beschreibung der Software (README.md) näher erklärt. Benötigt werden hier vor allem die Zugangsdaten zu den einzelnen Systemen und einige Einstellungen für die Systeme selbst.

**Service** einrichten, wenn benötigt. Das unterscheidet sich je nach Betriebssystem und muss daher individuell recherchiert werden.

**Logdatei** auf etwaige Fehler oder Probleme überprüfen (/logs/rollingfile.log). Beim Start werden die Zugangsdaten der Systeme getestet und das Ergebnis in die Logdatei geschrieben, wie in Abbildung 4.6 ersichtlich.

```
[INFO ] 2018-06-06 14:43:37.275 [main] Application - Starting Application v1.0 on [REDACTED] with PID 28076 (/
[DEBUG] 2018-06-06 14:43:37.276 [main] Application - started by [REDACTED]
[INFO ] 2018-06-06 14:43:37.277 [main] Application - Running with Spring Boot v2.0.0.RELEASE, Spring v5.0.4.RELEASE
[INFO ] 2018-06-06 14:43:41.323 [main] MetricQueue - creating directory queuedFiles
[INFO ] 2018-06-06 14:43:41.787 [main] MetricQueue - creating directory queuedFiles/error
[INFO ] 2018-06-06 14:43:43.055 [main] Initializer - Initializing application
[INFO ] 2018-06-06 14:43:43.190 [main] Initializer - Elasticsearch configuration detected, registering as consumer.
[INFO ] 2018-06-06 14:43:43.589 [main] BitBucketServerRestClient - BitBucket connection check returns ApplicationProperties{version='5.4.8', buildNumber='5004008', buildDate='1521092392556', displayName='Bitbucket'}
[INFO ] 2018-06-06 14:43:43.589 [main] Initializer - Bitbucket configuration detected, registering as producer.
[INFO ] 2018-06-06 14:43:43.674 [main] JiraSoftwareServerRestClient - JIRA connection check returns ServerInfo(baseUrl='https://[REDACTED]', version='7.5.4', buildNumber='75010', scmInfo=[REDACTED], buildPartnerName='null', serverTitle='[REDACTED]')
[INFO ] 2018-06-06 14:43:43.674 [main] Initializer - Jira Software configuration detected, registering as producer.
[INFO ] 2018-06-06 14:43:43.786 [main] SonarQubeRestClient - SonarQube authentication test returns HTTP status 200
[INFO ] 2018-06-06 14:43:43.786 [main] Initializer - SonarQube configuration detected, registering as producer.
[DEBUG] 2018-06-06 14:43:43.819 [threadPoolTaskScheduler-1] CronObserver - Checking metrics queue...
```

Abbildung 4.6: Logausgaben bei erfolgreichem Start

Wird die Uhrzeit für den täglichen Durchlauf nicht anders eingestellt, werden ab dann täglich alle Metriken um 00:15 generiert und abgelegt. Diese Metriken können dann visualisiert werden, wie in diesem Fall in Kibana.

## 4.4.1 Visualisierung der Metriken

Die Ergebnisse wurden in einem Kibana Dashboard visualisiert. Kibana ermöglicht es, Daten, die in einer ElasticSearch-Datenbank gespeichert sind, auf unterschiedlichste Weise zu visualisieren. Aufgrund der fehlenden Authentifizierung kann das Dashboard von jedem aufgerufen und angepasst werden. Es ist dadurch für alle Teammitglieder leicht zugänglich und anpassbar. Wird eine Authentifizierung in Kibana gewünscht, muss der Elastic Stack über das Produkt X-Pack ergänzt werden. Die Metriken werden im Dashboard in kurzfristige Metriken und langfristige Metriken unterteilt. Die Unterteilung wurde gemacht, da bei Kibana ein Zeitraum angegeben wird, in dem die Metriken angezeigt werden sollen. Außerdem sind die kurzfristigen Metriken eher für die Entwicklerinnen und die langfristigen Metriken eher für Scrum-Master und Product-Owner interessant.

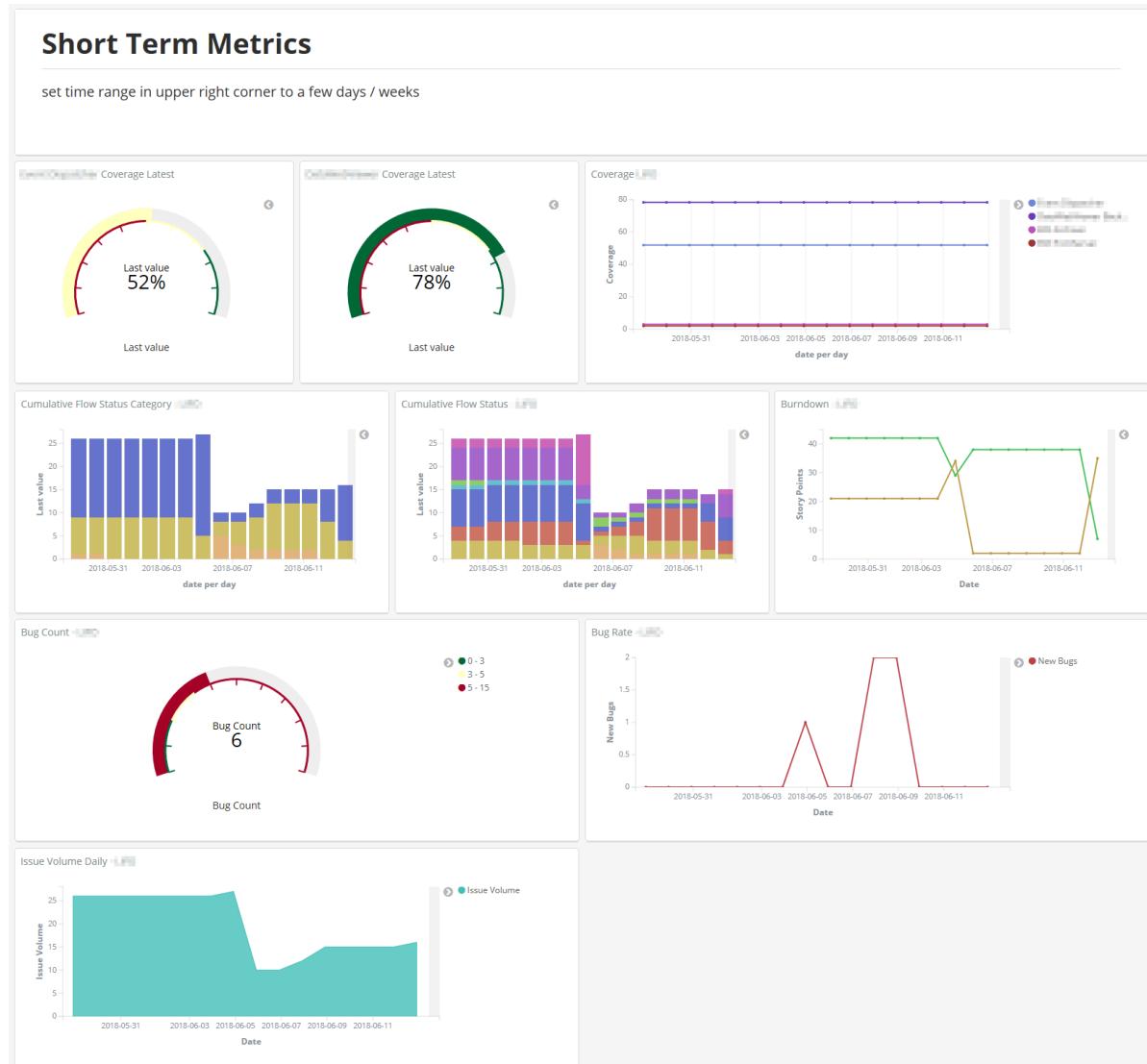


Abbildung 4.7: Teil des Dashboards mit den kurzfristigen Metriken

## **Coverage**

Das Team hat die Verantwortung über mehrere Produkte, deshalb wurde die Testabdeckung von zwei Produkten in Form einer Messuhr dargestellt. In einem weiteren Liniendiagramm wird der Verlauf der Testabdeckung aller Produkte über die Zeit dargestellt.

## **Burndown**

Die Anzahl erledigter und noch offener Story-Points im aktuellen Sprint bilden das Burndown-Chart. Es wird klassisch als Liniendiagramm dargestellt.

## **Cumulative Flow**

Beim Cumulative Flow werden nicht nur die einzelnen Stati dargestellt, sondern in einem separaten Diagramm auch noch die Status-Kategorien. Diese können bei sehr vielen unterschiedlichen Stati einen besseren Überblick bieten. Dargestellt werden beide als gestapelte Blockdiagramme, da sie einen Anteil an einem Gesamtvolumen darstellen sollen.

## **Bug Count und Bug Rate**

Die Anzahl der Bugs wird als Messuhr mit bestimmten Grenzwerten dargestellt. Die Bug-Rate als Liniendiagramm, um zu erkennen, wann wie viele neue Bugs erstellt wurden.

## **Issue Volume**

Das Aufgabenvolumen des aktiven Sprints wird als Flächendiagramm dargestellt.

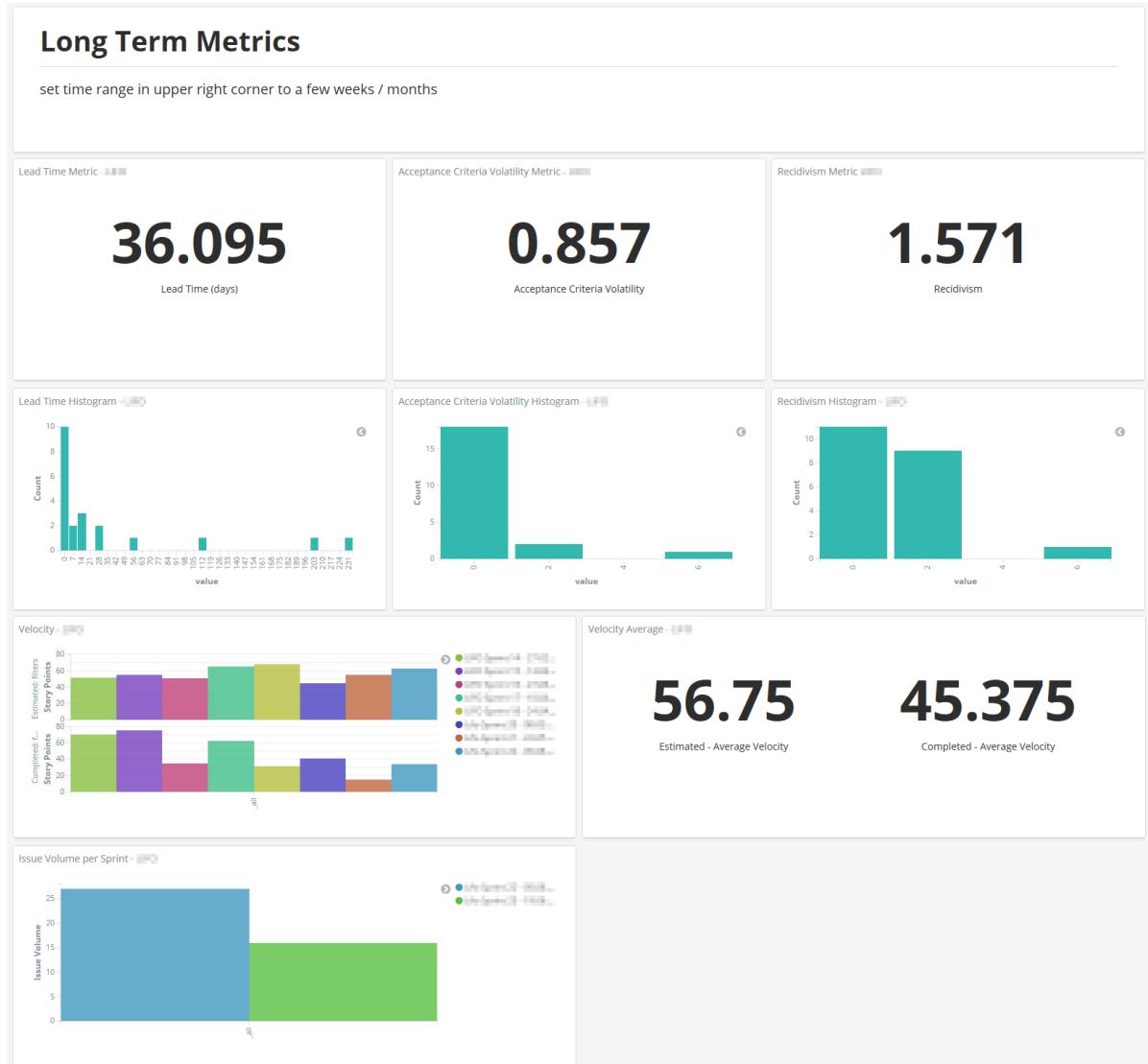


Abbildung 4.8: Teil des Dashboards mit den langfristigen Metriken

### Lead Time

Die Lead Time wird zum Einen als Metrik dargestellt, zum Anderen auch als Histogramm. Das hat den Hintergrund, dass hier einzelne Ausreißer nach oben oder unten die Metrik stark beeinflussen können. Das Histogramm stellt diese Ausreißer und die Verteilung noch besser dar.

### Acceptance Criteria Volatility

Auch bei der Flüchtigkeit der Akzeptanzkriterien wird die Metrik durch ein Histogramm ergänzt, da hier ebenfalls Ausreißer großen Einfluss haben können.

### Recidivism

Die Rückfälligkeit wird auch als Metrik und Histogramm dargestellt.

### Velocity

Wird als Blockdiagramm dargestellt. Sichtbar sind die geschätzten und die tat-

sächlich fertiggestellten Story-Points pro Sprint.

#### Issue Volume

Ein Blockdiagramm der Aufgabenvolumina vergangener Sprints, um die Entwicklung darzustellen.

#### 4.4.2 Status-Schnittstelle

Über die RESTful Schnittstelle der Software kann der aktuelle Status abgefragt werden.

```
{  
    consumers: [  
        "at.grisa.agilemetrics.consumer.elasticsearch.ElasticSearchConsumer@2a3888c1"  
    ],  
    producers: [  
        "at.grisa.agilemetrics.producer.bitbucketserver.BitBucketServerProducer@1722011b",  
        "at.grisa.agilemetrics.producer.jirasoftwareserver.JiraSoftwareServerProducer@7d3d101b",  
        "at.grisa.agilemetrics.producer.sonarqube.SonarQubeProducer@421e361"  
    ],  
    processedMetricsLastRun: 572,  
    lastRun: "2018-06-06T00:16:18.858+02:00"  
}
```

Abbildung 4.9: Ergebnis der Abfrage der Status-Schnittstelle

Abbildung 4.9 zeigt eine solche Antwort der Schnittstelle. Ersichtlich ist eine Liste von registrierten Consumern und Producern, sowie Informationen zur letzten zeitgesteuerten Generierung der Metriken (Anzahl generierter Metriken und Zeitstempel).

# 5 Evaluierung

Um die Effektivität der erarbeiteten Lösung feststellen zu können, wird eine Evaluierung durchgeführt. Ergänzend werden zur qualitativen Evaluierung ausgewählte Teammitglieder interviewt.

## 5.1 Quantitative Evaluierung

### Acceptance Criteria Volatility

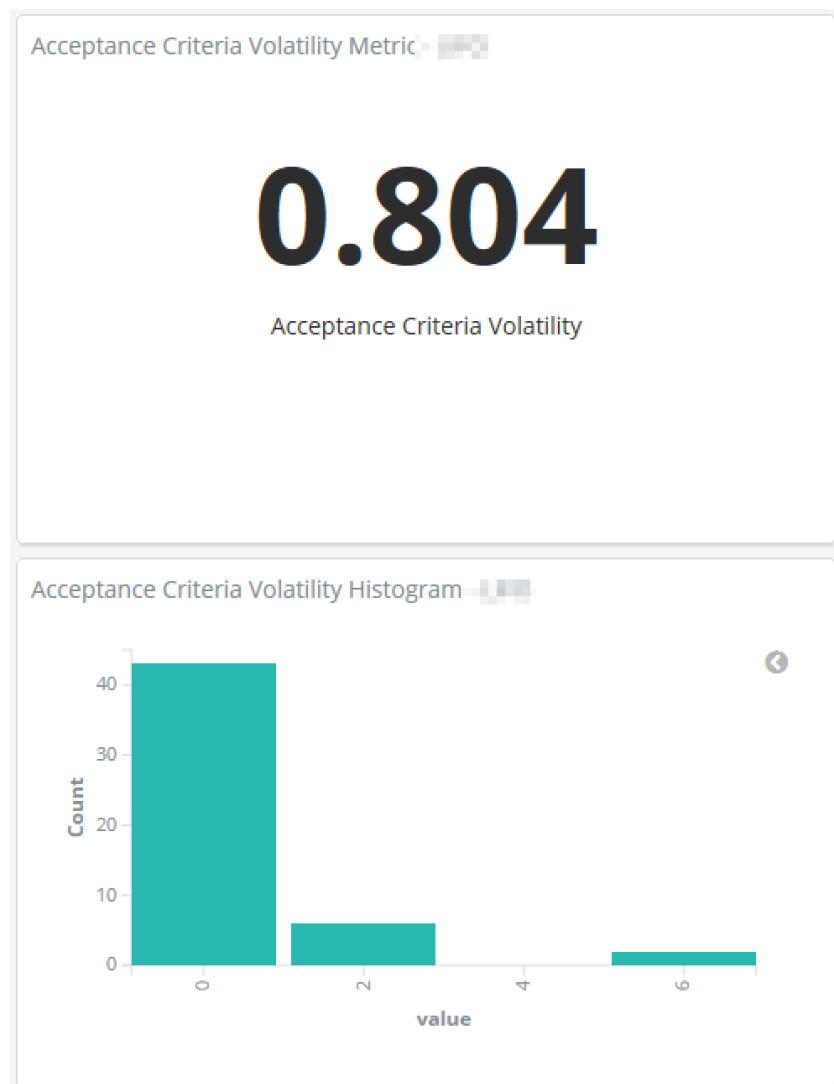


Abbildung 5.1: Quantitative Evaluierung - Acceptance Criteria Volatility

Im Durchschnitt werden die Akzeptanzkriterien einer Aufgabe 0,8-mal geändert, was auf den ersten Blick gut aussieht. Auch in der Verteilung im Histogramm ist zu sehen, dass ein großer Teil der Akzeptanzkriterien nach dem Erstellen der Aufgabe nicht mehr verändert wird. Nur sehr wenige Aufgaben wurden oft verändert. Das kann bedeuten, dass der Product-Owner inzwischen so gut versteht, was das Team von ihm verlangt, dass es zu fast keinen Änderungen mehr kommt. Es kann aber auch bedeuten, dass die Aufgaben in den Planungsmeetings nicht kritisch hinterfragt werden.

## Bug Count und Bug Rate

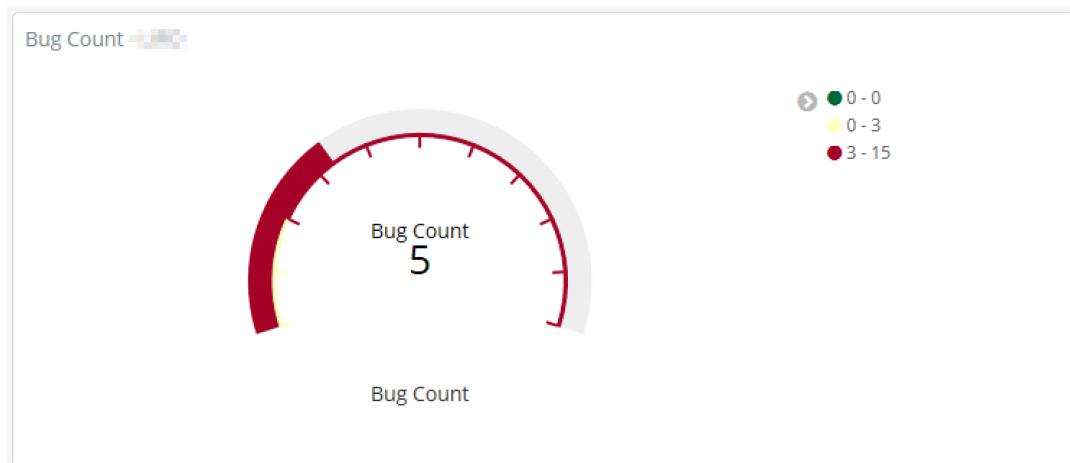


Abbildung 5.2: Quantitative Evaluierung - Bug Count



Abbildung 5.3: Quantitative Evaluierung - Bug Rate

Wie in der Bug Rate in Abbildung 5.3 ersichtlich, wurden in regelmäßigen Abständen Bugs erzeugt. Abbildung 5.2 zeigt, dass noch fünf Bugs im Backlog liegen, woraus sich schließen lässt, dass Bugs im Team abgearbeitet werden. Hier wäre noch ein Diagramm sinnvoll, wie viele Bugs pro Tag abgearbeitet werden (Bug-Fertigstellungsrate).

## Burndown

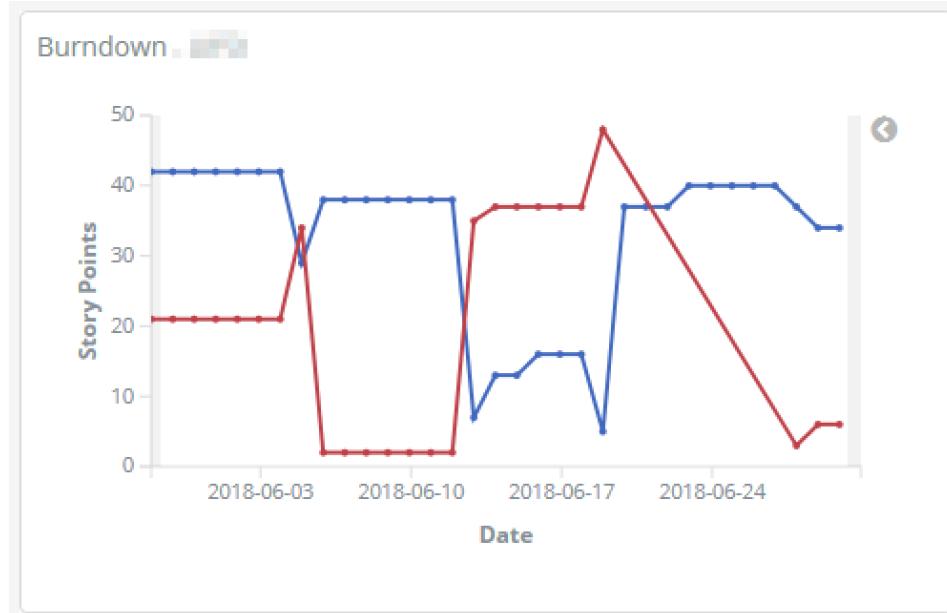


Abbildung 5.4: Quantitative Evaluierung - Burndown

Die Linien zeigen die offenen (blau) und die abgeschlossenen Punkte (rot). Das Diagramm ist sehr sprunghaft, vor allem gegen Sprintende werden plötzlich viele Punkte abgeschlossen. Ebenfalls ersichtlich ist, dass während dem Sprint immer wieder Punkte dazukommen. Hier besteht noch viel Verbesserungspotential. Bei den abgeschlossenen Punkten (rot) ist ebenfalls ersichtlich, dass ein paar Datenpunkte fehlen. Das kann ein Problem der Agile-Metrics-Software sein und muss genauer analysiert werden.

## Coverage

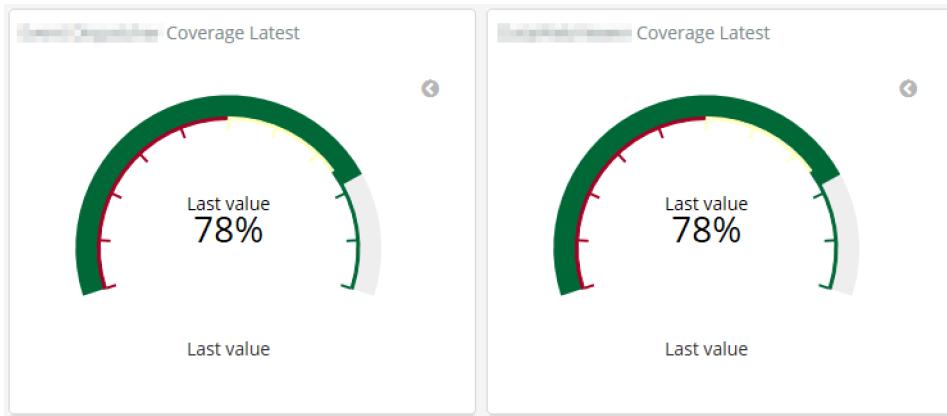


Abbildung 5.5: Quantitative Evaluierung - Coverage der kritischen Komponenten

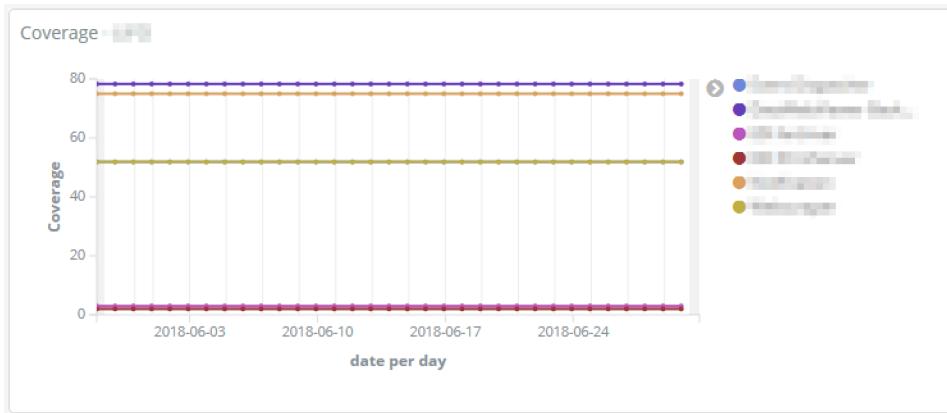


Abbildung 5.6: Quantitative Evaluierung - Coverage aller Komponenten

Bei der Coverage hat sich im Testzeitraum nichts verändert. Zwei der Hauptkomponenten sind mit 78% Testabdeckung auf einem guten Niveau, wie in Abbildung 5.5 ersichtlich. Andere Komponenten befinden sich fast bei null Prozent, wie in Abbildung 5.6 ersichtlich, und haben daher noch viel Verbesserungspotential.

## Cumulative Flow

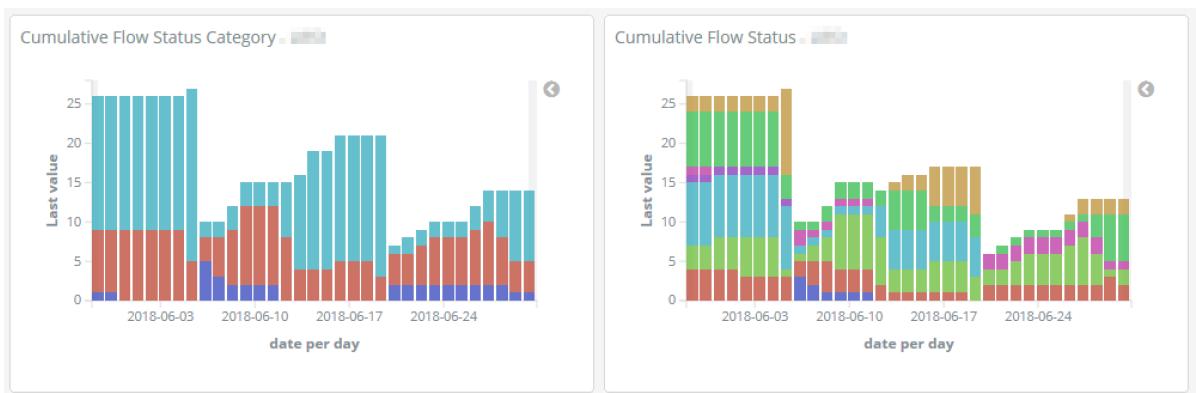


Abbildung 5.7: Quantitative Evaluierung - Cumulative Flow

Anhand des Cumulative Flow, vor allem dem der Status-Kategorien, ist der Fortschritt des Sprint-Backlogs besser zu erkennen. Anders als im Burndown-Diagramm ist hier ein kontinuierlicher Fortschritt ersichtlich. Auffallend ist, dass bei jedem Sprint im Testzeitraum nicht abgeschlossene Punkte in den nächsten Sprint übernommen wurden.

## Labels



Abbildung 5.8: Quantitative Evaluierung - Labels

Die Labels wurden vom Team bei der Retrospektive für jede abgeschlossene Aufgabe vergeben. Für die Arbeit vergaben sie die Labels 'WorkPlus' (Arbeit gut verlaufen) und 'WorkMinus' (Arbeit schlecht verlaufen) und für die Aufbereitung wurde 'IssuePlus' (Aufgabe gut aufbereitet) und 'IssueMinus' (Aufgabe schlecht aufbereitet) vergeben. Dadurch kann das subjektive Empfinden des Teams während eines Sprints oder eines bestimmten Zeitraums dargestellt werden. In diesem Fall wurde im Testzeitraum die Arbeit als durchwegs gut und die Aufbereitung als mehrheitlich gut empfunden.

## Lead Time

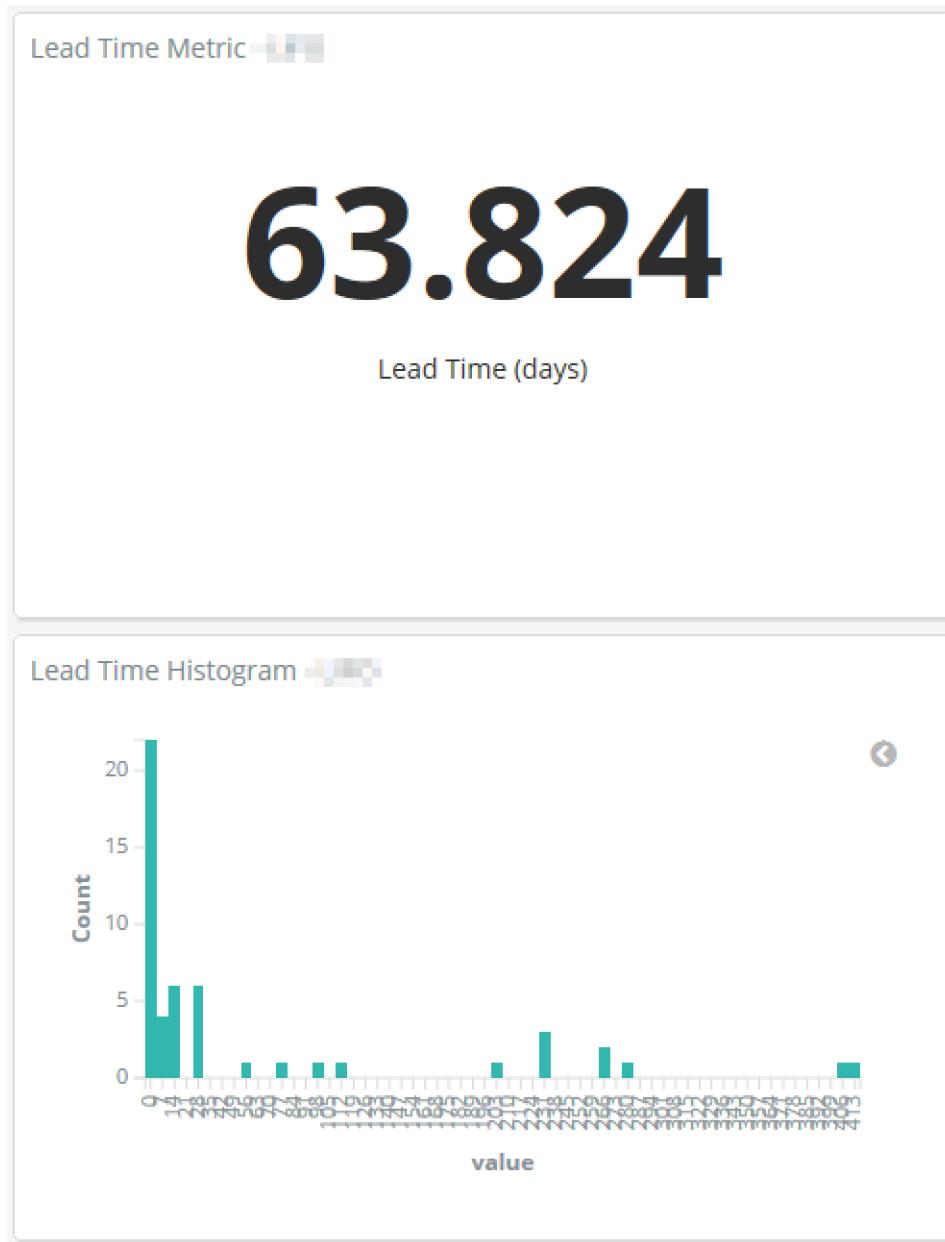


Abbildung 5.9: Quantitative Evaluierung - Lead Time

Bei der Lead Time ist das Histogramm sehr hilfreich, da hier die Ausreißer sehr gut ersichtlich sind. Nicht nur, dass der größte Teil der Aufgaben am selben Tag abgeschlossen wurde, an dem sie erstellt wurden, sondern auch, dass gewisse Aufgaben mehr als 100 Tage lang, manche sogar bis zu 400 Tage bis zur Fertigstellung benötigten. Leider fehlt hier noch eine Möglichkeit, solche Ausreißer noch genauer zu analysieren.

## Recidivism

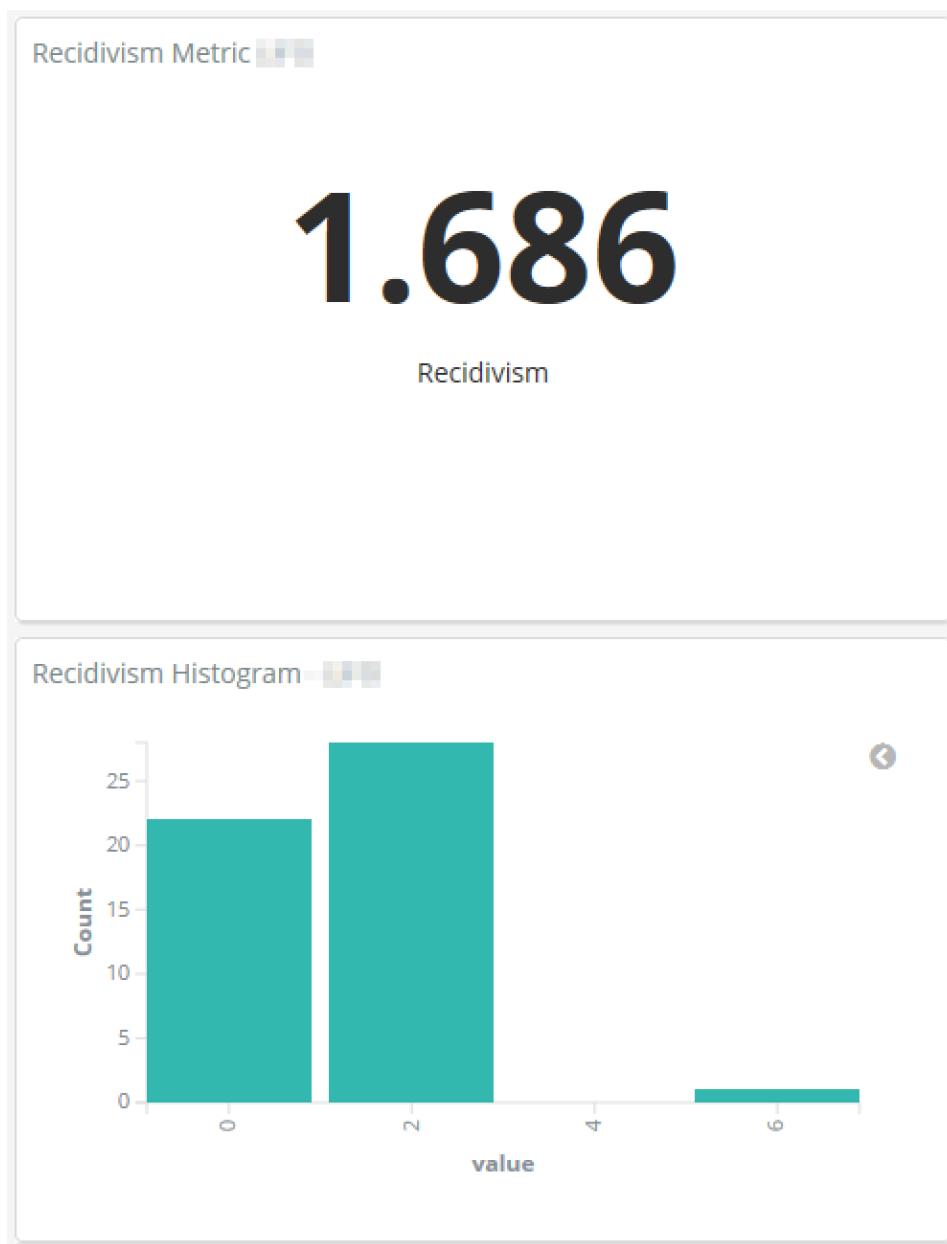


Abbildung 5.10: Quantitative Evaluierung - Recidivism

Auch bei der Rückfälligkeit gibt es noch Verbesserungspotential. Die Metrik zeigt, dass im Testzeitraum eine Aufgabe im Schnitt 1,6-mal im Arbeitsfluss rückwärts ging. Im Histogramm ist auch in der Verteilung zu erkennen, dass die meisten Aufgaben zweimal im Arbeitsfluss rückwärts gingen. Eine Aufgabe ging sogar 6-mal rückwärts und auch hier fehlt noch eine Detailansicht, um solche Ausreißer zu identifizieren.

## Test Execution Time

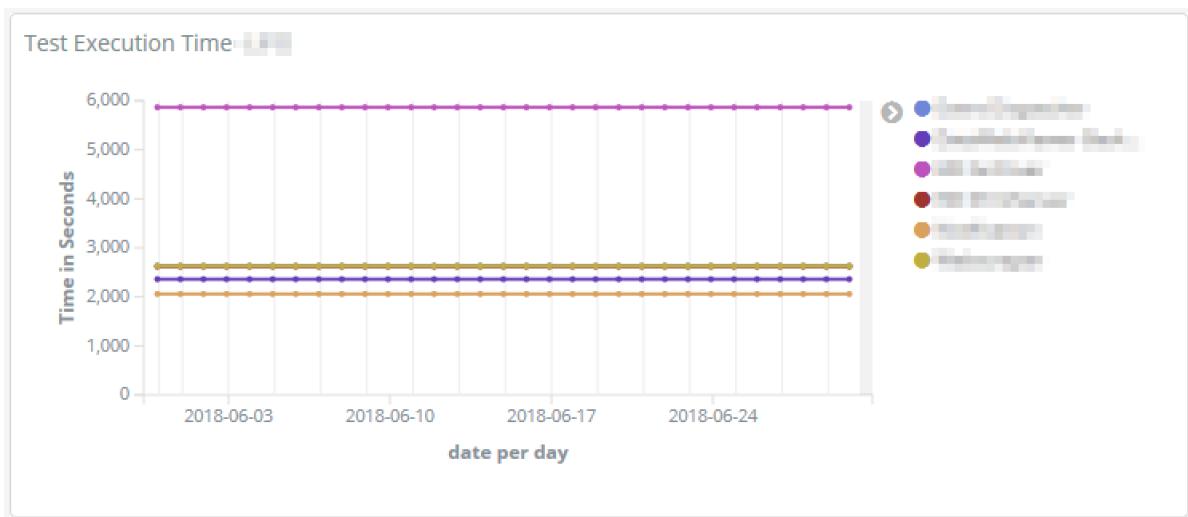


Abbildung 5.11: Quantitative Evaluierung - Test Execution Time

Die Laufzeit der Tests ist noch ausbaufähig. Sie blieb zwar im gesamten Testzeitraum konstant, was aber auch bedeuten kann, dass in diesem Zeitraum keine neuen Tests geschrieben wurden. Hier kann oft schon viel durch parallele Testausführung erreicht werden.

## Velocity



Abbildung 5.12: Quantitative Evaluierung - Velocity

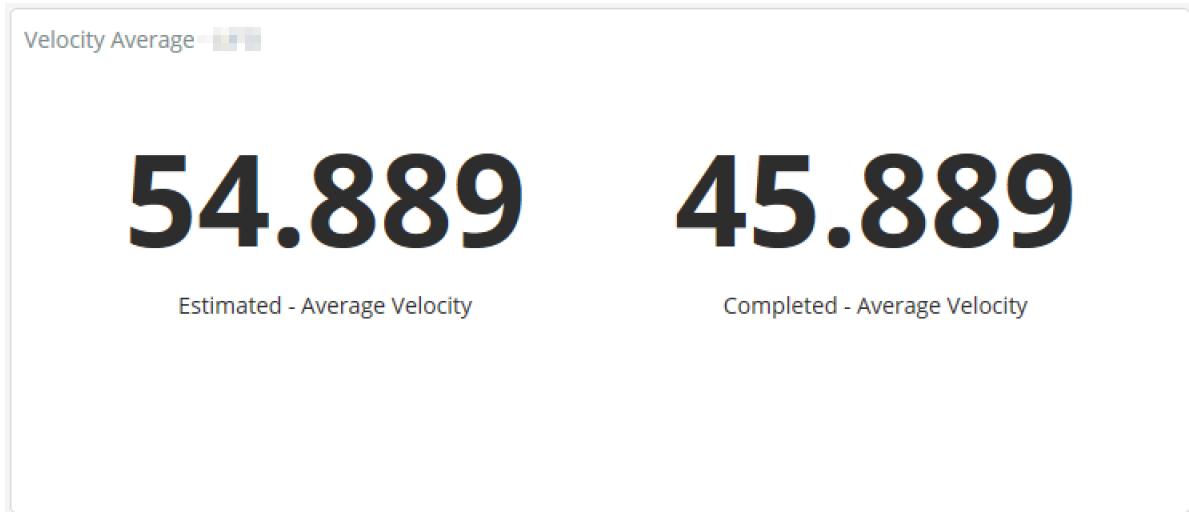


Abbildung 5.13: Quantitative Evaluierung - Velocity Durchschnitt

Velocity ist die einzige Metrik, die für mehrere Sprints rückwirkend generiert wurde. Abbildung 5.12 zeigt, dass im letzten Sprint, in dem das Dashboard aktiv war, wieder mehr Punkte abgearbeitet wurden als anfangs geschätzt. In den zwei Sprints zuvor wurde mehr geschätzt als abgeschlossen. Beim Durchschnitt in Abbildung 5.13 ist es das Ziel, die beiden Kennzahlen über die Zeit möglichst gut anzunähern.

## Issue Volume

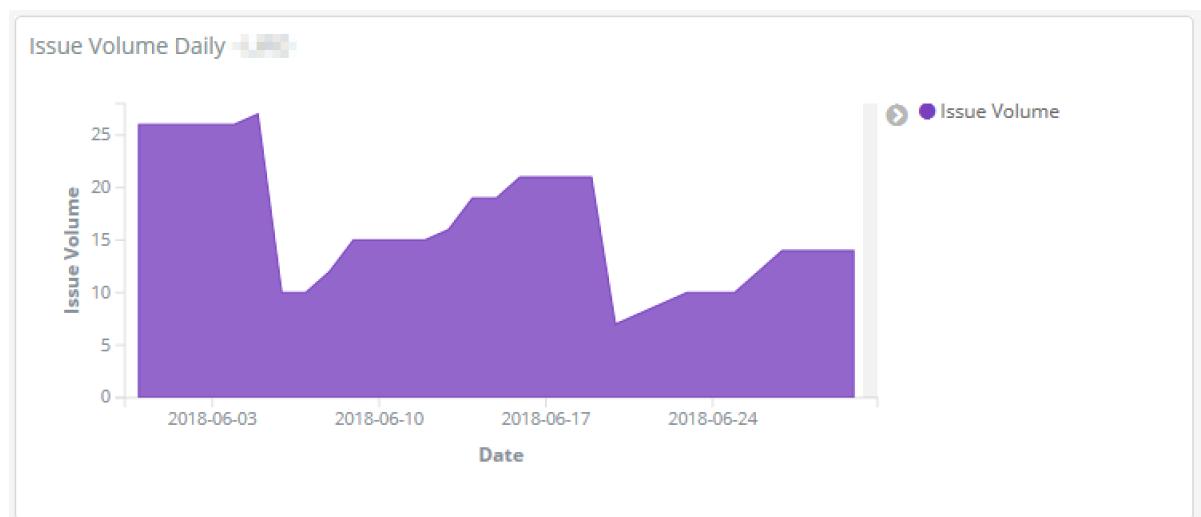


Abbildung 5.14: Quantitative Evaluierung - Issue Volume

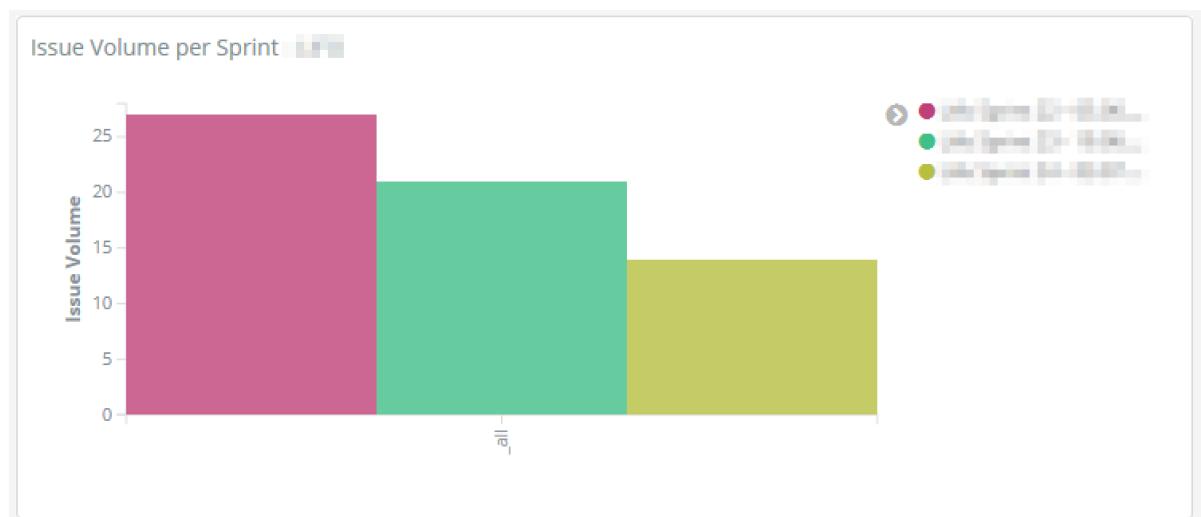


Abbildung 5.15: Quantitative Evaluierung - Issue Volume Total

Das absolute Issue Volume hat sich in den drei Sprints des Testzeitraums kontinuierlich verkleinert, wie in Abbildung 5.15 ersichtlich. Trotzdem zeigt Abbildung 5.14, dass während jedem Sprint immer wieder neue Aufgaben zum Sprint-Backlog hinzugefügt wurden.

## **Zusammenfassung**

In der quantitativen Evaluierung sind bei einigen Metriken schon Tendenzen erkennbar. Ob die Verbesserungen allerdings durch das Dashboard eingetreten sind, ist in diesem kurzen Zeitraum von nicht ganz drei Sprints nicht eindeutig zu sagen. Um eine solche Aussage treffen zu können, müsste das Team länger begleitet werden, um auch Korrelationen zwischen den Metriken aufzeigen zu können. Was aber eindeutig gesagt werden kann, ist, dass viele der Metriken Schwachstellen aufzeigen konnten und somit gezeigt werden konnte, dass die Auswahl an Metriken richtig getroffen wurde. Einige Schwachstellen, wie das Anzeigen von mehr Details zu den Metriken, könnten noch behoben werden, um das Dashboard effizienter zu machen.

## 5.2 Qualitative Evaluierung

Zur qualitativen Evaluierung werden Interviews mit dem Scrum-Master, dem Product-Owner und einer Entwicklerin geführt. Das ermöglicht einen Einblick aus Managementsicht (Product-Owner) und aus Prozesssicht (Scrum-Master und Entwicklerin), welche womöglich ganz unterschiedliche Anforderungen an die Lösung stellen. Dabei soll herausgefunden werden, ob die richtigen Metriken ermittelt wurden und welche Metriken als besonders nützlich angesehen werden. Ebenfalls wird nach einer nachweisbaren und spürbaren Qualitätsverbesserung gefragt. Interessant ist auch noch, wann und wie das Dashboard genutzt wird und wie es um die Benutzbarkeit steht.

### 5.2.1 Interview-Fragen

Folgende Interviewfragen sollen helfen, den roten Faden im Gespräch zu behalten. Als Einstieg wird das Dashboard geöffnet, um es während dem Gespräch immer sichtbar vor sich zu haben.

1. Wird das Dashboard von dir genutzt? Wenn ja, wann und wie nutzt du das Dashboard?
2. Wie ist der Zugang und die Bedienbarkeit des Dashboards?
3. Ist das Dashboard übersichtlich und klar eingeteilt?
4. Rückblickend gesehen, wurden die richtigen Metriken ermittelt und auf dem Dashboard visualisiert?
5. Welche Metriken auf dem Dashboard sind besonders nützlich oder werden oft genutzt?
6. Ist bereits eine Qualitätsverbesserung im Prozess oder in einem Produkt spürbar? Oder sogar nachweisbar?
7. Gibt es aus deiner Sicht Verbesserungspotential? Wo liegen aus deiner Sicht die Schwächen dieser Lösung?

Die Interviews wurden mit Zustimmung der Interviewten aufgezeichnet, ein Transkript befindet sich in Anhang A.4.

## 5.2.2 Interview-Antworten

Auffallend bei den Antworten sind die unterschiedlichen Sichten auf das Dashboard. Während der Product-Owner das Dashboard mehr für Werbezwecke genutzt hat, um andere Abteilungen auf die Möglichkeit von Metriken aufmerksam zu machen, hat es der Scrum-Master eher für die Langzeit-Sicht des Teams genutzt und seinen Fokus auf die Verbesserung des Prozesses gelegt. Die Entwicklerin wiederum hatte den Fokus auf die kurzfristigen Metriken, wie den Bug Count, um schnell auf Probleme reagieren zu können. Genutzt wurde es daher von allen Befragten, wichtiger ist aber, dass alle einen Nutzen darin sahen und es für sich bestmöglich nutzen können. Bei der Bedienbarkeit sind einzelne Schwächen aufgetaucht, vor allem was die Detailansicht von Kennzahlen betrifft. Die Einteilung der Metriken auf dem Dashboard wurde von allen als übersichtlich empfunden, eine Kennzeichnung der betroffenen Rollen (Scrum-Master, Product-Owner oder Entwicklerinnen) wäre eventuell noch sinnvoll. Auch eine genauere Beschreibung der Ziele und gewisser Metriken könnte Missverständnisse verhindern. Die Auswahl der Metriken wurde als gut empfunden, manche Metriken machen erst Sinn, wenn sie mit anderen kombiniert werden. Diese Kombination von Metriken war vor allem dem Scrum Master bewusst und wird in absehbarer Zukunft auch noch so umgesetzt. Welche Metriken als besonders nützlich angesehen werden, hängt stark von der Rolle ab. Wie bereits anfangs erwähnt, interessiert sich die Entwicklerin mehr für die kurzzeitigen Metriken, die unmittelbar Feedback zur Entwicklung liefern, während das Interesse des Scrum-Masters und Product-Owners mehr bei den Langzeit-Metriken liegt. Eine Qualitätsverbesserung ist noch nicht eindeutig ersichtlich, da es ein noch zu kurzer Zeitraum ist, um darüber Aussagen treffen zu können. Was aber von der Entwicklerin als spürbar erwähnt wurde, ist der Umgang mit den Bugs, die nun frühzeitig erkannt und abgearbeitet werden können. Verbesserungspotential sehen alle noch bei der Nutzung der Metriken, was aber auch vor allem auf den zu kurzen Zeitraum der Tests rückführbar ist. Einige gute Vorschläge, wie das Beschreiben von Zielen, eine bessere Kategorisierung nach Rollen und das Interesse an neuen Metriken zeigen aber, dass das Dashboard durchaus genutzt und das Ziel dahinter, das Ganze als Werkzeug für das gesamte Team zu nutzen, bereits erkannt wurde.

# 6 Schlussfolgerungen

In dieser Arbeit konnte in einer Fallstudie gezeigt werden, dass mithilfe der GQM-Methodik, ergänzt durch eine Umfrage im entsprechenden Scrum-Team, Metriken ermittelt werden können, die es ermöglichen, die Schwachstellen in einem Produkt und im agilen Prozess in Zahlen zu fassen. Diese Metriken ermöglichen es dem agilen Team, seine Fortschritte bei den Retrospektiven nachvollziehbar zu bewerten und weitere Maßnahmen zu treffen. Außerdem werden durch die Kombination unterschiedlicher Metriken Korrelationen zwischen bestimmten Metriken nachgewiesen oder widerlegt. Reichen die bekannten Metriken in der Literatur nicht aus, besteht die Möglichkeit, dass auch eigene Metriken erstellt werden.

Die entwickelte Software hilft dabei, die Daten aus den unterschiedlichen Systemen im Entwicklungsprozess als Metriken aufzubereiten und zu speichern. Dabei wurde bei der Architektur auf Fehlertoleranz und einfache Erweiterbarkeit geachtet, sodass ein einfacher Betrieb und eine Erweiterung der unterstützten Systeme und Metriken möglich ist. Für einen einfachen Zugang und eine uneingeschränkte Erweiterbarkeit wurde der Quellcode der Software unter der quelloffenen MIT-Lizenz veröffentlicht. Bei der Visualisierung von Metriken bietet Kibana eine geeignete Plattform, um aus den gespeicherten Metriken einfach Dashboards mit unterschiedlichen Visualisierungen bereitzustellen. Dabei muss berücksichtigt werden, dass Entwicklerinnen, Scrum-Master und Product-Owner jeweils unterschiedliche Interessen an den Metriken haben. Durch eine geeignete Gruppierung der Metriken ist es möglich, den unterschiedlichen Teammitgliedern auf einen Blick die wichtigsten Metriken anzuzeigen. Dadurch wird das Dashboard auch regelmäßig genutzt und somit die Akzeptanz noch weiter erhöht. Der Einsatz der Software ermöglicht den Aufbau einer zentralen Stelle, an der die Metriken aus allen relevanten System gesammelt, dargestellt und kombiniert werden.

Schwachstellen wurden bei der Art der Darstellung mancher Metriken identifiziert. So wurde zum Beispiel der Wunsch nach einer besseren Beschreibung der Metriken, um Diskussionen über deren Bedeutung zu vermeiden, geäußert. Zusätzlich soll künftig eine detailliertere Beschreibung von Zielen helfen, die Absicht hinter Metriken zu erklären. Bei manchen Darstellungen wurden noch zusätzliche Detailinformationen zu den einzelnen Datensätzen gewünscht, um Ausreißer leichter zu identifizieren. Trotzdem wurde das erstellte Dashboard trotz des relativ kurzen Testzeitraums von nicht ganz drei Sprints gut angenommen. Bei der Evaluierung wurde auch klar, dass die Teammitglieder bereits erkannten, wie sie das Dashboard persönlich am besten nutzen.

Durch den Einsatz der entwickelten Software und der vorgestellten Modelle zur Identifizierung von relevanten Metriken, kann die Qualität in einem agilen Team dadurch erhöht werden, dass Qualitätsprobleme durch Metriken sichtbar gemacht und in den Retrospektiven Gegenmaßnahmen dafür getroffen werden können.

# 7 Zusammenfassung

Scrum ist inzwischen eine sehr weit verbreitete agile Vorgehensweise von Entwicklungsteams. Dabei ist die Reflexion der bisherigen Arbeit eine der Grundideen von Scrum, der sogenannte Empirismus. Empirismus ist die Theorie, dass Wissen aus Erfahrung erlangt wird. Um auf Basis dieses Wissens Entscheidungen zu treffen, benötigt ein Scrum-Team Kennzahlen. Solche Kennzahlen können in Form von Metriken in einem leicht zugänglichen Dashboard visualisiert werden. Metriken dienen hauptsächlich dazu, Qualitätsmerkmale eines Produkts oder Prozesses als Zahlenwert darzustellen.

Eingeführt wurde ein solches Dashboard bei einem relativ fortgeschrittenen Scrum-Team in einem Unternehmen mit rund 6700 Mitarbeitern. Gestartet wurde mit der Ermittlung von geeigneten Metriken für das Team. Dazu wurden zuerst alle bisherigen Retrospektiven ausgewertet und basierend auf diesen Daten mit der GQM-Methodik Metriken ermittelt. Zusätzlich wurde aufgrund des erfahrenen Teams noch eine Umfrage gemacht, bei der den Teammitgliedern die gängigsten Metriken vorgestellt und in einer Skala von eins bis zehn nach Wichtigkeit bewertet wurden.

Um diese ermittelten Metriken dann darzustellen, wurde eine leicht erweiterbare, quelloffene Software in Java erstellt, die alle relevanten Daten aus den Systemen im Entwicklungsprozess über Schnittstellen sammelt, aufbereitet und als Metriken in einer Datenbank speichert. Konkret wurde der sogenannte Elastic Stack eingesetzt, also eine Elasticsearch Datenbank zur Speicherung und Kibana zur Visualisierung der Metriken und Bereitstellung über Dashboards. Diese Metriken wurden anschließend vom Scrum-Team über den Zeitraum von nicht ganz drei Sprints getestet. Dabei konnte gezeigt werden, dass je nach Rolle im Team zwar andere Metriken wichtig sind, aber jeder das Dashboard für sich zu nutzen wusste. Besonders der Scrum-Master bekam schnell einen Eindruck, welche Möglichkeiten dem Team dadurch eröffnet werden.

## Ausblick

Folgende Erweiterungen dieser Arbeit wären möglich:

### Testzeitraum

Die Entwicklung des Teams und des Dashboards mit den Metriken könnte noch über einen längeren Zeitraum verfolgt werden, um ausführlichere Rückschlüsse über die Effektivität und den Nutzen zu ziehen.

### Testumfang

Der Testumfang kann auf mehrere Teams erweitert werden, um ein noch umfangreicheres Feedback zu bekommen.

## **Systeme**

Mehr unterstützte Systeme kann die Akzeptanz der Software erhöhen.

## **Metriken**

Auch neue Metriken erhöhen die Akzeptanz der Software und erlauben es Teams, noch mehr Einsicht in ihre Prozesse und Systeme zu erlangen.

## **Geschäftsebenen**

Das Dashboard könnte noch weiteren Ebenen im Unternehmen bereitgestellt werden, zum Beispiel dem Management, um eine Übersicht über den Gesamtprozess zu erlangen.

# Literatur

- Abts, Dietmar und Wilhelm Mülder. *Masterkurs Wirtschaftsinformatik: Kompakt, praxisnah, verständlich - 12 Lern- und Arbeitsmodule*. de. Springer-Verlag, Okt. 2009. ISBN: 978-3-8348-0002-2.
- Apache JMeter - Apache JMeter<sup>TM</sup>*. 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. *Atlassian Team Playbook - Team Building Activities that Work*. de-DE. URL: <https://de.atlassian.com/team-playbook> (besucht am 30.06.2018).
- *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).
- Basili, Victor R, Gianluigi Caldiera und H Dieter Rombach. *THE GOAL QUESTION METRIC APPROACH*. en. URL: <http://www.cs.umd.edu/~mvz/handouts/gqm.pdf>.
- 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).
- GitHub Pages - Agile Metrics*. URL: <https://dagrisa.github.io/agile-metrics/> (besucht am 31.05.2018).
- GitHub Repository - Agile Metrics*. URL: <https://github.com/DaGrisa/agile-metrics> (besucht am 31.05.2018).
- 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).

*LeSS Overview Diagram*. URL: <https://less.works/img/LeSS-overview-diagram.png> (besucht am 06.06.2018).

*Manifest für Agile Softwareentwicklung*. URL: <http://agilemanifesto.org/iso/de/manifesto.html> (besucht am 16.03.2018).

*MIT Lizenz*. URL: <https://choosealicense.com/licenses/mit/> (besucht am 31.05.2018).

*Overview - Large Scale Scrum (LeSS)*. URL: <https://less.works/less/framework/index.html> (besucht am 06.06.2018).

*Prinzipien hinter dem Agilen Manifest*. URL: <http://agilemanifesto.org/iso/de/principles.html> (besucht am 16.03.2018).

*Scrum at Scale Framework / SMPO Cycle*. URL: <https://www.scrumatscale.com/wp-content/uploads/SMPO-Cycle.png> (besucht am 06.06.2018).

*Scrum of Scrums / Agile Alliance*. URL: <https://www.agilealliance.org/glossary/scrum-of-scrums/> (besucht am 06.06.2018).

*SonarCloud - Agile Metrics*. URL: <https://sonarcloud.io/dashboard?id=at.grisa.agile-metrics%3Aagile-metrics> (besucht am 31.05.2018).

*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 At Scale® Guide*. en-US. URL: <https://www.scrumatscale.com/scrum-at-scale-guide-read-online/> (besucht am 06.06.2018).

*The Scrum Framework Poster / Scrum.org*. URL: <https://www.scrum.org/resources/scrum-framework-poster> (besucht am 01.04.2018).

*Travis CI - Agile Metrics*. URL: <https://travis-ci.org/DaGrisa/agile-metrics> (besucht am 31.05.2018).

Wagner, Stefan. *Software Product Quality Control*. Englisch. 2013. Aufl. New York: Springer, Aug. 2013. ISBN: 978-3-642-38570-4.



# Anhang

## A.1 Metriken aus dem Entwicklungsprozess

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

Tabelle A.1: Kennzahlen aus dem VCS

---

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

---

Tabelle A.2: Kennzahlen aus dem PTS

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

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

---

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

---

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. ueberblick (3)
- 14. umgestellt (3)
- 15. altnetz (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 beschäftigt 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. impedance (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)

## **A.3 Umfrage Scrum Team**

### **A.3.1 Fragebogen**

# Metriken Scrum Team LIFO

Hier soll die Relevanz einzelner Metriken und der Fragen, die sie beantworten können, bestimmt werden.

\* Erforderlich

## 1. Rolle \*

Markieren Sie nur ein Oval.

- Scrum Master
- Product Owner
- Developer

# Metriken aus dem VCS

Hier können Daten darüber gesammelt werden, wie viel gearbeitet und auch wie viel zusammengearbeitet wird.

## 2. Changed Lines Of Code (CLOC) \*

Wie viele Änderungen passieren in der Codebasis? Wo finden die meisten Änderungen statt?  
Markieren Sie nur ein Oval.

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

## 3. CLOC pro Entwickler \*

Wie viel Code ändert jeder im Team? Wer ist wie oft in welchem Modul?  
Markieren Sie nur ein Oval.

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

## 4. CLOC pro Commit \*

Wie groß sind die Commits?  
Markieren Sie nur ein Oval.

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**5. Commits \***

Wie viel Änderungen wurden im Quellcode vorgenommen?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**6. Commits pro Entwickler \***

Wie viel Änderungen wurden im Quellcode von einem Entwickler vorgenommen?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**7. Kommentare pro Commit \***

Wer arbeitet zusammen? Wie viel wird zusammengearbeitet?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**8. Pull Requests \***

Wird mit Pull Requests gearbeitet? Werden Reviews gemacht?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**9. Gemergte Pull Requests \***

Wie oft werden erfolgreiche Änderungen in die Codebasis übernommen?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

#### 10. Abgelehnte Pull Requests \*

Wie oft werden Änderungen an der Codebasis abgelehnt? Wie klar sind die Erwartungen des Teams an eine abgeschlossene Änderung (DoD)?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

#### 11. Kommentare pro Pull Request \*

Wer arbeitet zusammen? Wie viel wird zusammengearbeitet?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

### Metriken aus dem PTS (JIRA)

Hier können Daten über das Projektverständnis des Teams, die Geschwindigkeit und vor allem die Konsistenz der Arbeit gesammelt werden.

#### 12. Burn Down \*

Erfüllt das Team seine Commitments? Plant das Team seine Arbeit realistisch?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

#### 13. Velocity \*

Wie konsistent arbeitet das Team?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

#### 14. Cumulative Flow \*

Gibt es Engpässe oder Schwachstellen im Prozess? Müssen gewisse Abläufe im Prozess optimiert werden?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**15. Lead Time \***

Wie schnell können Aufgaben vom Team erledigt werden? Wie lange dauert die Umsetzung eines neuen Features?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**16. Bug Counts \***

Wie viele Fehler werden vom Team im Entwicklungsprozess übersehen? Wie viel ungeplante Arbeit kam zum Sprint dazu?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**17. Bug-Erzeugungsrate \***

Wie viele Fehler wurden zu einem bestimmten Zeitpunkt erzeugt?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**18. Bug-Fertigstellungsrate \***

Wie viele Fehler wurden zu einem bestimmten Zeitpunkt beseitigt?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**19. Aufgaben-Volumen \***

Wie viel ungeplante Arbeit kam zum Sprint dazu? Wie groß ist die durchschnittliche Aufgabe?  
Gibt es Ausreißer?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

## 20. Aufgaben-Rückfälligkeit \*

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)?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

# Metriken aus CI- und CD-Systemen

## 21. Build-Dauer \*

Wie lange dauert es ein Softwareartefakt zu erstellen? Wie verändert sich die Dauer der Erstellung eines Softwareartefakts über die Zeit?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

## 22. Build-Status \*

Gibt es ein Problem im Freigabeprozess?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

## 23. Build-Frequenz \*

Wird oft genug ein neues Softwareartefakt erstellt?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

## 24. Test Reports \*

Wie lange dauert ein kompletter Testdurchlauf? Gibt es Tests, die optimiert werden müssen?  
Wie oft werden fehlerhafte Tests in die Codebasis aufgenommen?

*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**25. Code Coverage \***

Gibt es Module, die nicht oder schlecht getestet sind? Wie sieht die Entwicklung der Testabdeckung über die Zeit aus?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**26. Stresstests oder Benchmarking \***

Ist das Produkt auch noch unter Last verwendbar? Wie verändert sich die Leistung über die Zeit?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**Metriken aus den APM- und BI -Systemen**

Ermöglichen Aussagen darüber, ob die Kunden zufrieden sind und wie das System arbeitet.

**27. CPU Nutzung \***

Arbeitet die Software technisch effizient? Ist die Hardware ausreichend? Gibt es eine erhöhte Auslastung nach einer Änderung?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**28. Heap Size \***

Arbeitet die Software technisch effizient? Ist die Hardware ausreichend? Gibt es eine erhöhte Auslastung nach einer Änderung?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**29. Fehlerraten \***

Werden seit einer Änderung mehr Fehler produziert? Wie entwickelt sich die Fehlerrate über die Zeit?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**30. Antwortzeiten \***

Reagiert und arbeitet das Produkt noch schnell genug? Gibt es Geschwindigkeitsprobleme seit der letzten Änderung? Wie entwickeln sich die Antwortzeiten über die Zeit?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**31. Benutzeranzahl \***

Wie entwickeln sich die Benutzerzahlen mit der Zeit? Geht das Produkt in die richtige Richtung? Ist mit höheren Lasten zu rechnen?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**32. Aufenthaltsdauer \***

Welche Features werden besonders oft / selten genutzt? Hat das neue Feature den gewünschten Effekt? Wird es genutzt?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**33. Conversion Rate \***

Wie entwickelt sich die Zahl der zahlenden Neukunden?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**34. Verfügbarkeit \***

Wie hoch ist die Ausfallsicherheit? Wie lange war die Applikation nicht verfügbar?  
*Markieren Sie nur ein Oval.*

1	2	3	4	5	6	7	8	9	10	
nicht interessant	<input type="radio"/>	sehr interessant								

**35. Semantisches Logging \***

Hier können Daten zu anderen Fragen gesammelt werden, die für den Prozess wichtig sind.  
Markieren Sie nur ein Oval.

1	2	3	4	5	6	7	8	9	10
nicht interessant	<input type="radio"/>	sehr interessant							

**36. Wenn von Interesse, welche Fragen wären noch interessant?**

---

---

---

---

## Ergänzungen und Feedback

**37. Gibt es deiner Meinung nach Schwachstellen im Prozess, im Produkt oder in Ressourcen, die nicht durch eine der vorher genannten Metriken abgedeckt ist?**

---

---

---

---

**38. Gibt es noch Metriken, die interessant wären, aber du hier vermisst hast? Wenn ja, welche?**

---

---

---

---

**39. War irgend etwas unklar? Müssen gewisse Abschnitte oder Metriken besser erklärt werden?**

---

---

---

---

**40. Ist die gewählte Skala sinnvoll? Gibt es Verbesserungsvorschläge?**

---

---

---

---

---

---

Bereitgestellt von

[Google Formulare](#)

### A.3.2 Ergebnisse

Rolle	Product Owner	Scrum Master	Developer	Developer	Developer	Developer	Durchschnitt
<b>Changed Lines Of Code (CLOC)</b>	3	4	3	7	3	8	7
<b>CLOC pro Entwickler</b>	3	2	6	4	6	7	3
<b>CLOC pro Commit</b>	4	4	8	4	3	4	7
<b>Commits</b>	3	6	7	8	2	8	8
<b>Commits pro Entwickler</b>	3	2	3	5	2	8	4
<b>Kommentare pro Commit</b>	6	7	5	3	3	7	3
<b>Pull Requests</b>	6	9	9	10	3	8	3
<b>Gemergte Pull Requests</b>	3	7	9	10	3	8	8
<b>Abgelehnte Pull Requests</b>	3	10	7	8	3	9	7
<b>Kommentare pro Pull Request</b>	5	9	4	6	4	3	4
<b>Burn Down</b>	10	10	10	8	6	10	9
<b>Velocity</b>	10	10	9	10	5	10	9
<b>Cumulative Flow</b>	10	9	9	6	7	8	9
<b>Lead Time</b>	10	10	9	6	4	10	8
<b>Bug Counts</b>	10	9	7	2	6	10	9
<b>Bug-Erzeugungsrate</b>	10	8	5	8	3	10	8
<b>Bug-Fertigstellungsrate</b>	10	6	5	8	3	7	8
<b>Aufgaben-Volumen</b>	10	10	8	7	8	8	9
<b>Aufgaben-Rückfälligkeit</b>	7	10	9	3	7	8	9
<b>Build-Dauer</b>	5	3	3	1	4	8	8
<b>Build-Status</b>	10	8	3	4	4	8	9
<b>Build-Frequenz</b>	5	2	3	4	5	4	8
<b>Test Reports</b>	10	4	6	4	6	10	9
<b>Code Coverage</b>	10	4	7	8	6	10	9
<b>Stresstests oder Benchmarking</b>	10	4	9	5	7	10	10
<b>CPU Nutzung</b>	7	4	2	10	6	10	9
<b>Heap Size</b>	7	4	2	10	6	10	9
<b>Fehlerraten</b>	10	3	3	7	6	7	10
<b>Antwortzeiten</b>	10	2	2	7	6	8	9
<b>Benutzeranzahl</b>	10	2	2	5	6	10	8
<b>Aufenthaltsdauer</b>	10	2	2	5	3	10	8
<b>Conversion Rate</b>	10	1	2	9	3	8	9
<b>Verfügbarkeit</b>	10	4	2	9	6	10	10
<b>Semantisches Logging</b>	10	1	2	3	4	8	8

**Wenn von Interesse, welche Fragen wären noch interessant?**

Obige Metriken (Metriken aus den APM- und BI -Systemen) sind sehr interessant allerdings nicht für Scrum-Retrospektive?! Diese Infos sollte an einem anderen Ort/Zeit besprochen werden. Dann allerdings sehr interessant, zwischen 7 und 9.

**Gibt es noch Metriken, die interessant wären, aber du hier vermisst hast? Wenn ja, welche?**

Flüchtigkeit der Anforderungen: Wie oft wurde eine Anforderung angepasst

Anforderungen: wie oft und in welchem Zeitrahmen wird ein Feature geändert? (Bestellen, release, änderung, änderung, release, änderung, release)

**War irgend etwas unklar? Müssen gewisse Abschnitte oder Metriken besser erklärt werden?**

Mir war zunächst nicht klar was generell gefragt ist bzw das Ziel des Fragebogens.

**Ist die gewählte Skala sinnvoll? Gibt es Verbesserungsvorschläge?**

Lieber eine ungerade Skala verwenden, wo es keine Mitte gibt, sonst sinnvoll

**Gibt es deiner Meinung nach Schwachstellen im Prozess, im Produkt oder in Ressourcen, die nicht durch eine der vorher genannten Metriken abgedeckt ist?**

## A.4 Transkripte Interviews

Im Folgenden finden sich die Transkripte zu den Interviews, die zu Evaluationszwecken geführt wurden. Der Interviewer wird mit A und der oder die Interviewte mit B bezeichnet.

### A.4.1 Product Owner

A: "Wird das Dashboard von dir genutzt? Wenn ja, wie?"

B: "Bisher primär zur Demonstration, was du in deiner Arbeit gemacht hast. Ich habe es vielen gezeigt, auch darum, um zu zeigen, man kann einfach Daten sammeln und Metriken darstellen. Probleme habe ich teilweise noch mit den Metriken selbst, weil sie nicht alle ausreichend verständlich sind für mich. Uns wären auch noch viele andere eingefallen, beispielsweise zur Darstellung der Testabdeckung der Oberflächentests. Die Herausforderung für die Zukunft wird sein, sinnvolle Metriken finden zu können. Es muss klarer sein, was das Ziel der Metriken ist. Große Probleme hatte ich mit der Acceptance Criteria Volatility. Ich war aufgrund der Zahl irritiert, da ich Prozent gewohnt bin. Diese Metrik muss auch dahingehend verändert werden, dass die Veränderungen erst ab dem Status Prepared gezählt werden, denn davor ist es noch Aufarbeitung der Aufgabe, da kann sich noch viel ändern. Mir war ebenfalls nicht klar, was ist gut, was ist schlecht. Sehr nützlich wäre aus meiner Sicht auch noch eine textuelle Beschreibung der Ziele, die mit den Metriken verfolgt werden. Damit auch in der Retrospektive klar ist, was ist gut gelaufen und was nicht und auch Empfehlungen gegeben werden, wie ein Ziel verbessert werden kann. Ein Beispiel wären die Atlassian Playbooks<sup>44</sup>, die spielerisch versuchen, Probleme im Prozess zu beseitigen."

A: "Rückblickend gesehen, wurden die richtigen Metriken ermittelt und auf dem Dashboard visualisiert?"

B: "Schwierig zu sagen, für mich ist primär die Acceptance Criteria Volatility interessant, da sie eine Aussage über meine Arbeit trifft. Ein Vorschlag wäre noch, aufzuteilen, für wen welche Metriken interessant sein könnten. Die Einteilung könnte noch etwas klarer sein. Interessant wäre jetzt auch eine Kombination der Metriken, zum Beispiel die Bug Rate kombiniert mit den Releasezyklen. Ich lasse mich überraschen, was der Scrum Master für Ideen hat, weil es grundsätzlich seine Herausforderung sein wird."

### A.4.2 Scrum Master

A: "Ich sehe, du hast schon etwas vorbereitet, du kannst gerne anfangen."

B: "Ein paar Dinge, die mir aufgefallen sind: Im Dashboard wäre es sehr nützlich, den Zeitraum auf 'ab dem letzten Sprint' setzen zu können, das muss jetzt immer manuell als absolutes Datum gesetzt werden. Sinnvoll wäre auch noch, wenn die Generierung der Daten vor Mitternacht passiert, damit sie den Zeitstempel des richtigen Tages haben.

---

<sup>44</sup>Atlassian. *Atlassian Team Playbook - Team Building Activities that Work.* de-DE. URL: <https://de.atlassian.com/team-playbook> (besucht am 30.06.2018).

Interessant sind auch die Tage, an denen der alte Sprint geschlossen und der neue gestartet wird. Wie es sich dann damit verhält. Weiters ist mir noch aufgefallen, dass die Visualisierung bei den Labels nicht sinnvoll ist für unser Team, eine solche Wortwolke ist eher nützlich für das Management, für uns wäre eine Verteilung interessanter, um zu sehen, wie sie sich entwickeln. Bei der Lead Time ist mir aufgefallen, dass ich im Vergleich zum PTS noch zu wenig Detailinformationen habe. Für mich sehr nützlich ist die Velocity über die Sprints, weil man sieht, ob sich die geschätzten und erledigten Punkte über einen längeren Zeitraum angleichen. Im PTS finden sich noch weitere Metriken, die interessant sein könnten. In Zukunft werden wir auch mit Versionen arbeiten, diese könnten ebenfalls interessant sein im Dashboard, beispielsweise, wie lange wurde an einer Version gearbeitet oder wie viele Entwickler waren beteiligt.”

A: “Wird das Dashboard von dir genutzt? Wenn ja, wie?”

B: “Ich öffne es regelmäßig, für mich sind die Long Term Metrics interessanter, dort betrachte ich meist die letzten 30 Tage, um zu sehen, wie entwickelt sich das Team über die letzten zwei Sprints. Vor allem die Velocity ist hier interessant, eine Trendlinie wäre hier super, um noch besser zu sehen, wie sich die Kennzahlen entwickeln.”

A: “Ist das Dashboard übersichtlich und klar eingeteilt?”

B: “Die Einteilung in Short Term und Long Term Metrics macht auf jeden Fall Sinn, die Anordnung ist für mich okay, mit den Coverage Daten kann ich jetzt weniger anfangen, mit dem Cumulative Flow schon mehr. Wobei wir diesen jeden Tag auf unserem physischen Scrumboard sehen. Kombiniert mit anderen Metriken würde das dann mehr Sinn machen, beispielsweise mit der Bug Rate. Burndown Chart macht in dieser Form auch wenig Sinn, das haben wir im PTS, aber auch das macht in Kombination mit anderen Metriken Sinn. Issue Labels könnten als Liniendiagramm dargestellt und in die Long Term Metrics übernommen werden, um Trends erkennbar zu machen. Diese lässt sich dann auch einfacher mit anderen Metriken kombinieren, um Verbindungen herstellen zu können. Sehr interessant sind auch die Aufgaben, die zusätzlich zum Sprint hinzukommen. Generell kann jetzt begonnen werden, gewisse Metriken zu kombinieren, um noch bessere Einsichten in den Prozess zu bekommen.”

A: “Rückblickend gesehen, wurden die richtigen Metriken ermittelt und auf dem Dashboard visualisiert?”

B: “Ja die Metriken sind recht gut gewählt. Je mehr wir sammeln und anzeigen können, desto besser. Aber auch je mehr wir kombinieren können, desto besser.”

A: “Welche Metriken auf dem Dashboard sind besonders nützlich oder werden oft genutzt?”

B: “Für mich sind die Trends in den Long Term Metrics interessanter. Gut wären auch noch detailliertere Ansichten, beispielsweise, wenn ich einen Punkt in einem Diagramm anklische, dass ich sofort sehe, welche Daten dort dahinter liegen. Derzeit muss ich das dann über das PTS ermitteln.”

A: “Ist bereits eine Qualitätsverbesserung im Prozess oder in einem Produkt spürbar? Oder sogar nachweisbar?”

B: “Noch nicht, wir haben es verwendet und alle einmal hinein gesehen, aber momentan ist es noch so, dass wir Daten sammeln, weil mit den zwei oder drei Sprints noch keine klare Aussage getroffen werden kann. Außerdem sind unsere Sprints noch sehr

unterschiedlich, wie du sehen kannst.”

### A.4.3 Entwicklerin

A: “Wird das Dashboard von dir genutzt? Wenn ja, wann und wie nutzt du das Dashboard?”

B: “Aktuell nutze ich das Dashboard nicht sehr oft, aber wenn ich es nutze, finde ich es sehr hilfreich, dass ich gleich die Code Coverage sehe, so erspare ich mir den Blick ins SonarQube. Auch sehr interessant für mich sind die Anzahl Bugs und zusätzlich aufnehmen könnte man noch die Vulnerabilities aus SonarQube, um den aktuellen Status der Software besser erkennen zu können.”

A: “Ist das Dashboard übersichtlich und klar eingeteilt?”

B: “Aktuell ist die Einteilung für mich klar, ich persönlich würde noch den Cumulative Flow weiter nach unten schieben, da für mich vor allem die Short Term Metrics interessant sind.”

A: “Rückblickend gesehen, wurden die richtigen Metriken ermittelt und auf dem Dashboard visualisiert?”

B: “Ja, in Zukunft kann es noch weiter ausgebaut werden, auch, dass alle Produkte noch detaillierter dargestellt werden.”

A: “Werden Long Term Metrics auch von dir genutzt?”

B: “Interessant finde ich die Velocity, um erkennen zu können, was wir uns vorgenommen haben und was wir wirklich abschließen konnten. Auch die Acceptance Criteria Volatility finde ich sehr interessant, vor allem um den Prozess noch weiter zu verbessern. Hier sind auch die Histogramme hilfreich, um die Verteilung besser erkennen zu können. Für manche war diese Kennzahl verwirrend, da nicht klar war, ob es Prozent sind. Eine bessere Beschreibung wäre hier sinnvoll. Generell werden die Long Term Metrics eher bei den Retrospektiven genutzt.”

A: “Ist bereits eine Qualitätsverbesserung im Prozess oder in einem Produkt spürbar? Oder sogar nachweisbar?”

B: “Mir persönlich ist aufgefallen, dass wir durch den Bug Count frühzeitig gewarnt werden und auch gleich reagieren können.”

A: “Gibt es aus deiner Sicht Verbesserungspotential? Wo liegen aus deiner Sicht die Schwächen dieser Lösung?”

B: “Was mir noch nicht ganz klar ist, bei der Coverage, auf welchen Branch sich das bezieht.”

A: “Die SonarQube Komponente kann bei der Erstellung des Diagramms angegeben werden, in diesem Fall der Master Branch.”

A: “Rückblickend gesehen, wurden die richtigen Metriken ermittelt und auf dem Dashboard visualisiert?”

B: “Ich finde die Short Term Metrics sehr gut gewählt, wie schon gesagt, vermisste ich hier noch die Vulnerabilities aus SonarQube, zumindest für die wichtigsten Komponenten.”

# **Eidesstattliche Erklärung**

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 08. Juli 2018

Daniel Grießer