

# 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 Grieser  
Dornbirn, Juli 2018

# Kurzreferat

**[Deutscher Titel Ihrer Arbeit]**

[Text des Kurzreferats]

# Abstract

**[English Title of your thesis]**

[text of the abstract]



# Inhaltsverzeichnis

Abbildungsverzeichnis	7
Tabellenverzeichnis	8
Abkürzungsverzeichnis	9
<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	12
2.2 Scrum <sup>1</sup>	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	18
2.3 Software-Qualität	20
2.3.1 Produkt-Qualität	22
2.3.2 Prozess-Qualität	24
2.4 Metriken	25
2.4.1 Versionsverwaltung	26
2.4.2 Projektmanagement	27
2.4.3 Kontinuierliche Integration und Auslieferung	28
2.4.4 Produktionssystem	29
2.4.5 Übersicht Kennzahlen im Entwicklungsprozess	29
2.4.6 Eigene Metriken erstellen	29
2.4.7 Veröffentlichung von Metriken	30
2.4.8 Agile Prinzipien messen	30
2.4.9 Qualitätsmodelle	33
2.4.10 Goal Question Metric (GQM)	34
2.4.10.1 Beispiel	34

---

<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.

2.4.11	Umfrage (Quelle fehlt) . . . . .	35
<b>3</b>	<b>Vorgehensweise</b>	<b>36</b>
3.1	Metriken bestimmen . . . . .	36
3.2	Software . . . . .	37
<b>4</b>	<b>Umsetzung</b>	<b>38</b>
4.1	Gegebenheiten . . . . .	38
4.2	Metriken Identifizieren . . . . .	39
4.2.1	GQM . . . . .	39
4.2.2	Umfrage im Team . . . . .	41
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	Visualisierte Ergebnisse . . . . .	49
4.4.2	Statusschnittstelle . . . . .	53
<b>5</b>	<b>Evaluierung</b>	<b>54</b>
5.1	Quantitative Evaluierung . . . . .	54
5.2	Qualitative Evaluierung . . . . .	55
5.2.1	Interview-Fragen . . . . .	55
5.2.2	Interview-Antworten . . . . .	55
<b>6</b>	<b>Schlussfolgerungen</b>	<b>56</b>
<b>7</b>	<b>Zusammenfassung</b>	<b>57</b>
	<b>Literaturverzeichnis</b>	<b>58</b>
	<b>Anhang</b>	<b>60</b>
A.1	Metriken aus dem Entwicklungsprozess . . . . .	61
A.2	Ergebnisse Analyse Retrospektiven . . . . .	65
A.3	Umfrage Scrum Team . . . . .	68
A.3.1	Fragebogen . . . . .	68
A.3.2	Ergebnisse . . . . .	78
	<b>Eidesstattliche Erklärung</b>	<b>80</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 . . . . .	31
2.10	hierarchische Struktur des GQM Modells . . . . .	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 Statusschnittstelle . . . . .	53

# Tabellenverzeichnis

2.1	Beispiel GQM Modell . . . . .	35
4.1	GQM-Modell - Ablenkung der Entwickler . . . . .	40
4.2	GQM-Modell - Schwachstellen im Prozess . . . . .	40
4.3	GQM-Modell - Aufgabengröße . . . . .	41
A.1	Kennzahlen aus dem VCS . . . . .	61
A.2	Kennzahlen aus dem Project Tracking System (PTS) . . . . .	62
A.3	Kennzahlen aus den CI- und Continuous Delivery (CD) . . . . .	63
A.4	Kennzahlen aus den Application Performance Monitoring (APM)- und Business Intelligence (BI) . . . . .	64



# Abkürzungsverzeichnis

<b>APM</b>	Application Performance Monitoring
<b>BI</b>	Business Intelligence
<b>CD</b>	Continuous Delivery
<b>CI</b>	Continuous Integration
<b>CLOC</b>	Changed Lines of Code
<b>DoD</b>	Definition of Done
<b>EAT</b>	Executive Action Team
<b>EMS</b>	Executive MetaScrum
<b>GQM</b>	Goal Question Metric
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>LeSS</b>	Large Scale Scrum
<b>LOC</b>	Lines of Code
<b>MTTF</b>	Mean Time to Failure
<b>MTTR</b>	Mean Time to Release
<b>NASA</b>	National Aeronautics and Space Administration
<b>NoSQL</b>	Not only SQL
<b>PBR</b>	Product Backlog Refinement
<b>PTS</b>	Project Tracking System
<b>QS</b>	Qualitätssicherung
<b>SoS</b>	Scrum of Scrums
<b>VCS</b>	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, arbeiten nach einem evolutionären Ansatz. 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 Produkts 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 qualitative Schwachstellen im Prozess oder im Produkt abbilden können. Weiters sollen diese Metriken gesammelt und grafisch dargestellt werden. Diese Metriken können aus Daten erzeugt werden, die bei der tagtäglichen 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). Es werden dort tagtäglich Daten erzeugt, die über Schnittstellen abgefragt und anschließend aggregiert abgelegt werden können. Umgesetzt wird das Ganze in einem relativ jungen, aber im Scrum Prozess bereits weit fortgeschrittenen Scrum-Team.

## 1.2 Aufbau der Arbeit

Einen Einblick in die unterschiedlichen Themen dieser Arbeit gibt das Kapitel “Stand der Technik”. Zuerst werden die Grundsäulen der agilen Softwareentwicklung, das agile Manifest und die agilen Prinzipien, genauer erklärt. Darauf folgend wird Scrum genauer erklärt, zusätzlich Ansätze für Scrum in mehreren Teams. Anschließend wird zu Qualität übergegangen, im Speziellen Software- und Prozessqualität. Basierend auf den beiden vorherigen Themen, wird dann genauer auf Metriken eingegangen, was auch der Hauptteil dieses Kapitels darstellt. Im ersten Teil werden Metriken aus den unterschiedlichsten Systemen vorgestellt und wie eigene Metriken erstellt werden können. Danach folgen Hinweise zur Veröffentlichung von Metriken und der Messung von Agilen Prinzipien. Am Schluss folgen noch ein Überblick über Qualitätsmodelle und der Goal-Question-Metric Ansatz im Detail.

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

Das Kapitel “Umsetzung” zeigt dann, wie der Titel schon sagt, die Umsetzung der Lösung. Anfangs werden die Gegebenheiten erläutert, in der die Lösung eingesetzt wird. Dann wird mit der Identifizierung der Metriken gestartet und diese anschließend in der entwickelten Software gesammelt und am Ende genauer auf die Darstellung eingegangen. Am Ende folgt das Kapitel “Evaluierung”, in dem die Ergebnisse qualitativ in Form von Interviews evaluiert werden. In den Kapiteln “Schlussfolgerungen” und “Zusammenfassung” werden die Ergebnisse nochmal reflektiert, zusammengefasst und ein Ausblick für mögliche weitere Verbesserungen gegeben.

## 2 Stand der Technik

### 2.1 Agile Softwareentwicklung

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

#### 2.1.1 Agiles Manifest

Das agile Manifest ist 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)



- **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 2 bis 4 Wochen, in dem ein fertiges, verwendbares und potentiell releasebares Produkt-Inkrement entwickelt wird.
- **Sprint Planning:** Planung eines Sprints. Hier committed sich das Scrum Team, eine gewisse Anzahl an Aufgaben im kommenden Sprint abzuarbeiten.
- **Daily Scrum:** Tägliches, zeitlich begrenztes Meeting, bei dem von jedem Teammitglied folgende drei Fragen beantwortet werden:
  1. Was habe ich gemacht?
  2. Was werde ich machen?
  3. Was behindert mich bei meiner Arbeit?
- **Sprint Review:** Abschluss eines Sprints. Hier präsentiert das Team dem Product Owner die Ergebnisse des letzten Sprints.
- **Sprint Retrospective:** Das Team reflektiert den Sprint-Ablauf und ergreift Maßnahmen, um den Prozess weiter zu verbessern.

- **Artefakte**

- **Product Backlog:** Ist eine Sammlung von möglichen Aufgaben für das Team am Produkt. Sollte einen Ausblick auf die zukünftige Entwicklung des Produktes geben. Oben im Product Backlog befinden sich die bereits fein geplanten Aufgaben, weiter unten die groben.
- **Sprint Backlog:** Entspricht den Aufgaben, die vom Team in den Sprint genommen und dem Product Owner zugesagt wurden.
- **Increment:** Entsteht am Ende eines jeden Sprints und ist eine lauffähige Version des Produkts, die releasefähig ist.

### 2.2.1 Scrum in mehreren Teams<sup>4</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>4</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.

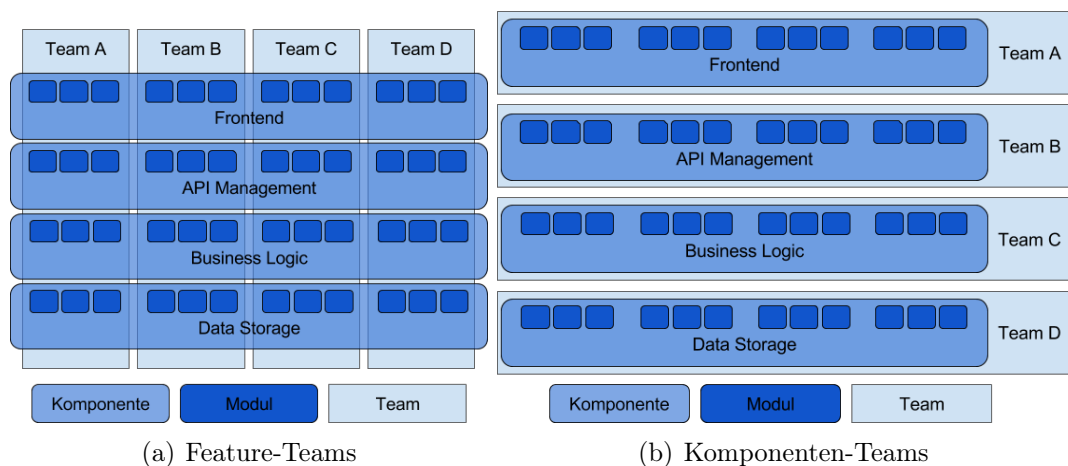


Abbildung 2.2: Scrum Teams

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 SoS<sup>5</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 ein Botschafter bestimmt, um an einem täglichen Meeting mit anderen Botschaftern teilzunehmen, das sogenannte Scrum of Scrums. Je nach Kontext sind die Botschafter technische Teilnehmer, Scrum Master oder sogar Team Manager. Das SoS hat sein eigenes Backlog, bei dem es Fertigstellungen, nächste Schritte und Hindernisse im Auge behält.

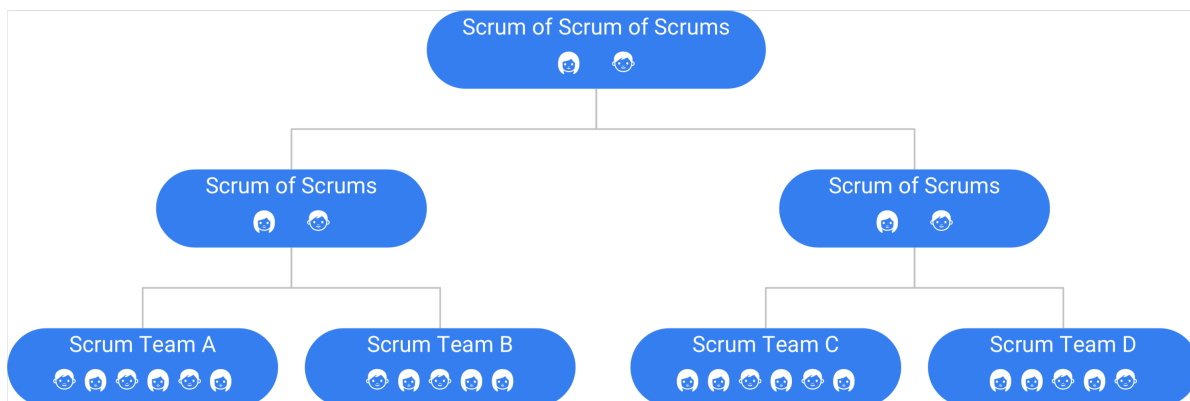


Abbildung 2.3: SoS auf 3 Ebenen

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

### 2.2.1.2 LeSS<sup>6</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>5</sup>vgl. *Scrum of Scrums* / Agile Alliance. URL: <https://www.agilealliance.org/glossary/scrum-of-scrums/> (besucht am 06.06.2018).

<sup>6</sup>vgl. *Overview - Large Scale Scrum (LeSS)*. URL: <https://less.works/less/framework/index.html> (besucht am 06.06.2018).

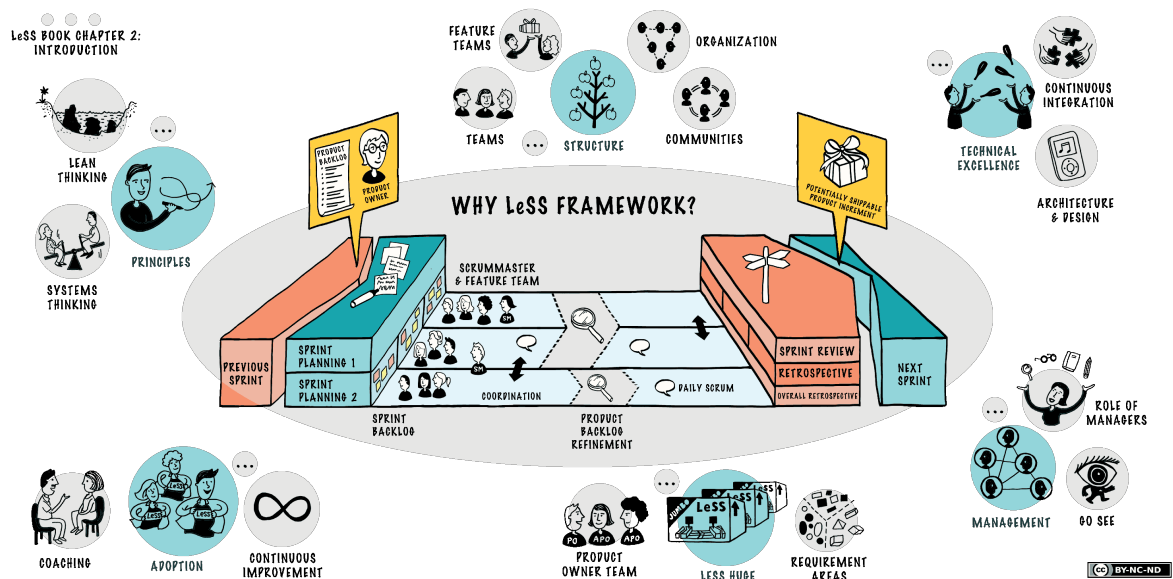


Abbildung 2.4: LeSS Framework<sup>7</sup>

Es gibt viele Dinge, die gleich bleiben, wie bei einem Team: Produkt 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 Ownern und rotierenden Personen der Teams, um den Gesamtprozess zu verbessern.

### 2.2.1.3 Scrum at Scale<sup>8</sup>

Scrum at Scale versucht Scrum skalierbar zu machen, durch ein Minimum an Bürokratie und einer frei skalierbaren Architektur. Es besteht aus Scrum Teams, die durch SoS und

<sup>7</sup> *LeSS Overview Diagram*. URL: <https://less.works/img/LeSS-overview-diagram.png> (besucht am 06.06.2018).

<sup>8</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).

MetaScrums koordiniert werden.

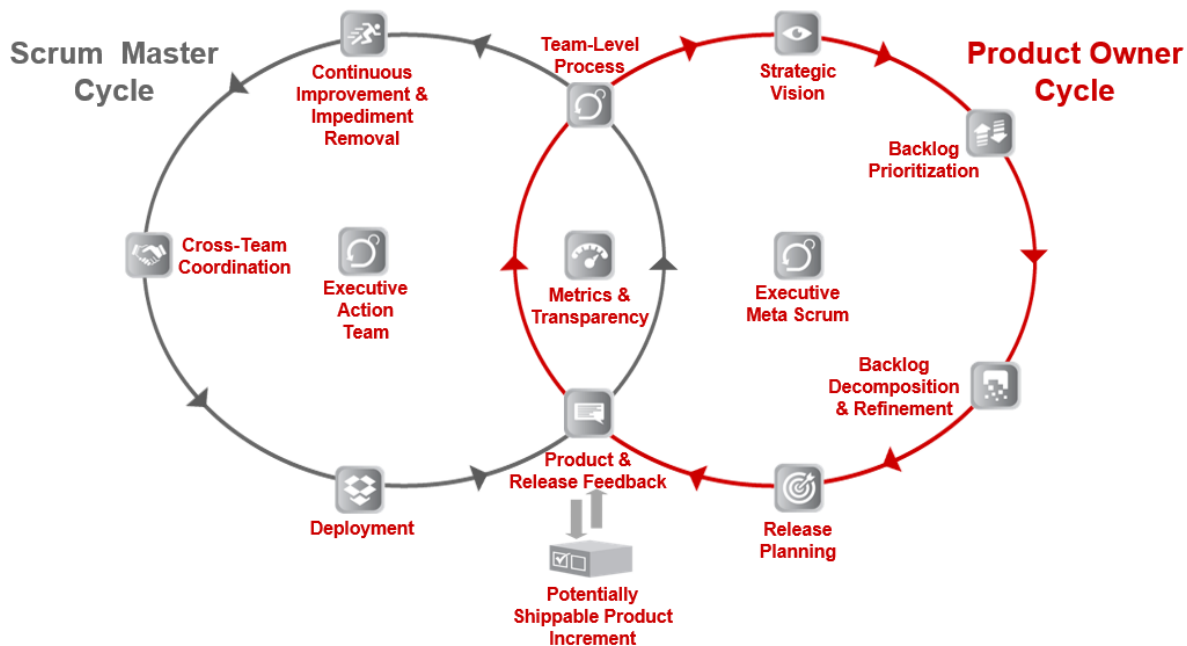


Abbildung 2.5: Scrum at Scale Framework - SMPO Zyklus<sup>9</sup>

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 Ownern, die ein SoS Backlog verantworten, nennt man MetaScrum. MetaScrums 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 MetaScrum (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. Ein Minimum an Metriken, die empfohlen werden, sind: Produktivität, Wertschöpfung, Qualität und Nachhaltigkeit.

<sup>9</sup>Scrum at Scale Framework / SMPO Cycle. URL: <https://www.scrumatscale.com/wp-content/uploads/SMP0-Cycle.png> (besucht am 06.06.2018).

## 2.3 Software-Qualität<sup>10</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-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen.”

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

### **Kundenorientierte Merkmale**

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

#### **Funktionalität (Functionality, Capability)**

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

#### **Laufzeit (Performance)**

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

#### **Zuverlässigkeit (Reliability)**

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

#### **Benutzbarkeit (Usability)**

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

### **Herstellerorientierte Merkmale**

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

#### **Wartbarkeit (Maintainability)**

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

#### **Transparenz (Transparency)**

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

---

<sup>10</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 (z.B. 32-Bit zu 64-Bit oder Desktop zu Mobile).

## Testbarkeit (Testability)

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

Je nach Anwendungsgebiet und den Anforderungen der Software haben die Merkmale unterschiedliche Relevanz und einige können sich auch gegenseitig beeinflussen, wie aus der Korrelationsmatrix in Abbildung 2.6 ersichtlich. Dabei sind die positiv korrelierenden Merkmale mit “+” und die negativ korrelierenden mit “-” gekennzeichnet.

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

Abbildung 2.6: Korrelationsmatrix Qualitätskriterien<sup>11</sup>

<sup>11</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.

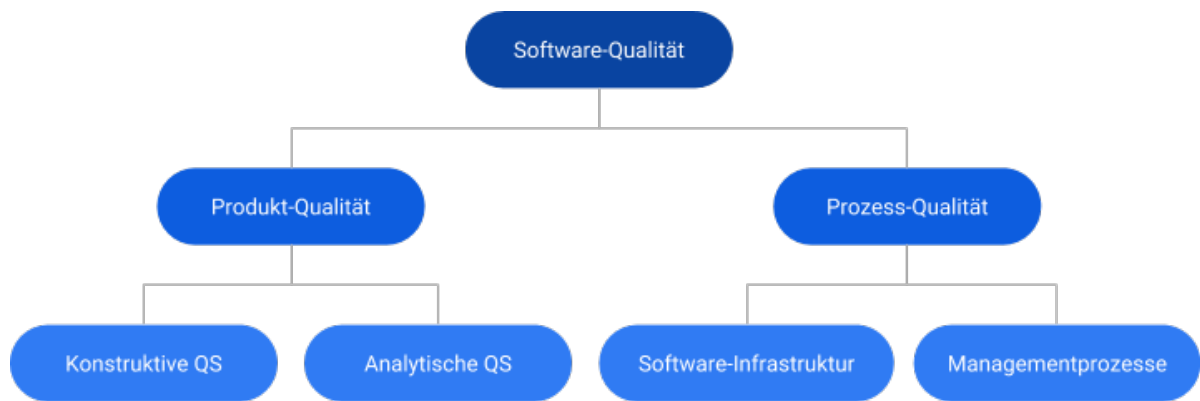


Abbildung 2.7: Software-Qualität

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 Prozess-Qualität einteilen, welche im Folgenden noch genauer beschrieben werden.

### 2.3.1 Produkt-Qualität<sup>12</sup>

Die Produkt-Qualität lässt sich in die zwei Bereiche konstruktive und analytische Qualitätssicherung unterteilen. Während die konstruktive Qualitätssicherung sich mit den Vorgaben beschäftigt, die ein Software-Produkt im Vorhinein erfüllen muss, dient die analytische Qualitätssicherung dem Analysieren und Bewerten im Nachhinein.

#### 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>12</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-Produkt mit vordefinierten Tests, welche aus Eingabewerten und erwarteten Ausgabewerten bestehen. Die Auswahl dieser Tests beeinflusst dabei direkt die Qualität. Es wird zwischen Black-Box- und White-Box-Tests unterschieden und Testmetriken verwendet, 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 Prozess-Qualität<sup>13</sup>

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

#### **Software Infrastruktur**

Unterstützt den Entwickler bei der täglichen Arbeit.

##### **Configurationsmanagement**

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

##### **Build-Automatisierung**

Erzeugt die Artefakte voll automatisch.

##### **Test-Automatisierung**

Genause wie das Erstellen der Artefakte, wird auch das Testen voll automatisch 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.

---

<sup>13</sup>vgl. Hoffmann, *Software-Qualität*, Kapitel 1.4.2.



## 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>14</sup>

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



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

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

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

### **Changed Lines of Code (CLOC)**

Anzahl der geänderten Code Zeilen.

### **CLOC pro Entwickler**

Anzahl der geänderten Zeilen im Quellcode pro Entwickler.

### **Commits**

Gesamtzahl an Commits in einem bestimmten Zeitraum.

### **Commits pro Entwickler**

Gesamtzahl an Commits in einem bestimmten Zeitraum pro Entwickler.

### **Kommentare pro Commit**

Anzahl der Kommentare pro Commit.

### **CLOC pro Commit**

Anzahl der geänderten Zeilen im Quellcode pro Commit.

### **Pull Requests**

Gesamtzahl an Pull Requests in 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.

---

<sup>15</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.62ff.

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

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

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

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

Dadurch ergeben sich folgende Kennzahlen aus einem PTS:

### **Burn Down**

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

### **Velocity**

Eine relative Messung der Konsistenz erledigter Arbeit über die Sprints.

### **Cummulative Flow**

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

### **Lead Time**

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

### **Bug Counts**

Die Anzahl an Bugs über die Zeit.

#### **Bug-Erzeugungsrate**

Anzahl Bugs nach Erstellungsdatum.

#### **Bug-Fertigstellungsrate**

Anzahl Bugs nach Erledigungsdatum.

### **Aufgaben-Volumen**

Die Anzahl der Aufgaben. 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.

---

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

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

CI- und CD-Systeme stellen sicher, dass die erstellte Software zu jedem Zeitpunkt auslieferbar ist, in dem sie zu definierten Zeitpunkten automatisch neu gebaut und ausgeliefert wird. In einer solchen Build-Pipeline können sehr viel nützliche Daten erzeugt werden, vor allem mit Tools für statische Analysen (wie zum Beispiel SonarQube<sup>18</sup>). Diese Systeme sind aber auch jene Elemente im Softwareentwicklungsprozess, die von Team zu Team am meisten variieren können. Daher hängen die erzeugten Daten auch stark vom jeweiligen Setup ab. Grundsätzlich können aber folgende Kennzahlen aus diesen Systemen ermittelt werden:

#### **Build-Dauer**

Geschätzte und tatsächliche Dauer der Builds.

#### **Build-Status**

Es können die Anzahl der erfolgreichen und fehlerhaften Builds gegenüber gestellt werden.

#### **Build-Frequenz**

Wie oft wird ein Build ausgelöst.

#### **Test Reports**

Anzahl erfolgreicher und fehlerhafter Tests, Gesamtdauer der Tests.

#### **Code Coverage**

Wie viel Prozent des Quellcodes ist mit Tests abgedeckt.

#### **Stresstests oder Benchmarking**

Wird oft im Build Prozess mit getestet mit Tools wie JMeter<sup>19</sup> oder Gatling<sup>20</sup>.

---

<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).

<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).

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

Daten aus den Produktionssystemen können gesammelte APM- oder auch BI-Kennzahlen sein. Diese Kennzahlen ermöglichen Aussagen, ob die Kunden zufrieden sind und wie das System arbeitet. Die BI-Kennzahlen sollten möglichst nahe am Entwicklungsteam gehalten werden, damit es verstehen kann, wie die Kunden die Applikation nutzen. Dazu können Frameworks wie StatsD<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.

### **Benutzeranzahl**

Anzahl gleichzeitiger Benutzer in der Applikation über die Zeit.

### **Aufenthaltsdauer**

Verweildauer der Benutzer auf bestimmten Seiten.

### **Conversion Rate**

Anzahl Benutzer die zu Kunden wurden.

### **Semantisches Logging**

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

### **Verfügbarkeit**

Verfügbarkeit der Applikation über die Zeit.

## 2.4.5 Übersicht Kennzahlen im Entwicklungsprozess

Die Metriken finden sich nochmal als Tabelle dargestellt und mit den 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 2 Dinge notwendig:

- Daten

<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).

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

- eine Funktion, um die Metrik zu berechnen

Dabei sollte darauf geachtet werden,

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

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

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

### Dashboards

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

### Emails

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

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

## 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.

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

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



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

# Effektive Software

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

31

– Heapgröße / 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



## 2.4.9 Qualitätsmodelle

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, Meta-Modell basierte 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, dass sie messbar wird. Verschiedenste Kritiken heben hervor, dass die Prinzipien zur Dekomposition von Qualitätscharakteristiken oft mehrdeutig sind. Außerdem sind die resultierenden Qualitätscharakteristiken nicht spezifisch genug, um direkt gemessen werden zu können.

### **Meta-Modell basierte Qualitätsmodelle**

Entstanden in den 1990er Jahren, als Forscher mehr ausgereifere Wege aufzeigten, um Qualitätscharakteristiken zu zerlegen. Dadurch entstanden auch ausführlichere Meta-Modelle. Ein Meta-Modell beschreibt, wie gültig Qualitätsmodelle strukturiert sind. Die vielen Meta-Modelle zeigen, dass Qualität mehr Struktur in Qualitätsmodellen, als in abstrakten Qualitätscharakteristiken 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.

## 2.4.10 GQM<sup>28</sup>

GQM ist ein Qualitätsmodell, um 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.

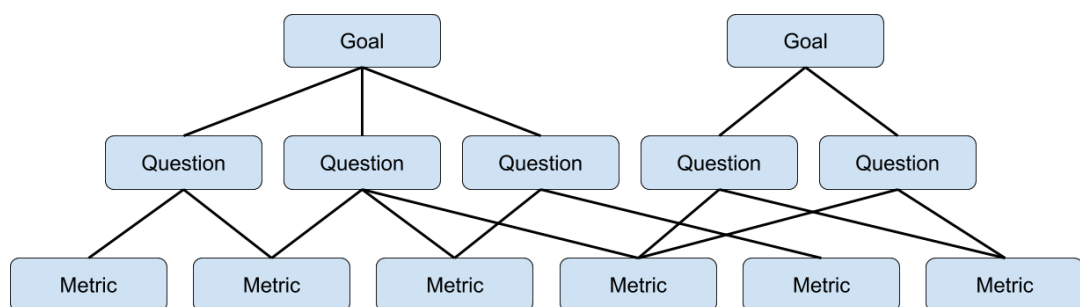


Abbildung 2.10: hierarchische Struktur des GQM Modells

Abbildung 2.10 zeigt die hierarchische Struktur des GQM-Modells.

### 2.4.10.1 Beispiel<sup>29</sup>

Anhand des Beispiels in Tabelle 2.1 soll veranschaulicht werden, wie so ein GQM-Modell 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 Pull Requests.

<sup>28</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>29</sup>Basili, Caldiera und Rombach, *THE GOAL QUESTION METRIC APPROACH*.

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.

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

\* Durchschnitt und Standardabweichung pro Sprint

Tabelle 2.1: Beispiel GQM Modell

#### 2.4.11 Umfrage (Quelle fehlt)

Quellen zu wissenschaftlichen Umfragen finden, passende einfügen. Grundsätzlich: Team wurden die gängigsten Metriken vorgestellt und die Fragen, die sie beantworten können. Jene Metriken, die in einer Skala von 1-10 eine x und größer hatten, wurden in dieser Arbeit umgesetzt. Am Ende wurden offene Fragen gestellt, falls ein Teilnehmer noch eine Metrik vermisste. Zusätzlich wurden qualitative Fragen zur Umfrage gestellt, um ein Feedback zur Methode zu bekommen.

## 3 Vorgehensweise

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

### 3.1 Metriken bestimmen

Bestimmung von relevanten Qualitätsmetriken von agilen Teams. Dabei müssen folgende Punkte beachtet werden:

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

## 3.2 Software

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

- Umsetzung in Java
- einzubindende Systeme: BitBucket Server<sup>30</sup>, JIRA<sup>31</sup>, Jenkins<sup>32</sup>, SonarQube<sup>33</sup>, Icinga<sup>34</sup>
- Speicherung und Darstellung der Metriken erfolgt in einem Elastic Stack<sup>35</sup>

---

<sup>30</sup>Atlassian. *Bitbucket Server*. en. URL: <https://www.atlassian.com/software/bitbucket/server> (besucht am 31.03.2018).

<sup>31</sup>Atlassian. *Jira / Software zur Vorgangs- und Projektverfolgung*. de-DE. URL: <https://de.atlassian.com/software/jira> (besucht am 31.03.2018).

<sup>32</sup>Jenkins. URL: <https://jenkins.io/index.html> (besucht am 31.03.2018).

<sup>33</sup>*Continuous Code Quality / SonarQube*.

<sup>34</sup>*Icinga*. en-US. URL: <https://www.icinga.com/> (besucht am 31.03.2018).

<sup>35</sup>*Elastic Stack*. de-de. URL: <https://www.elastic.co/de/products> (besucht am 31.03.2018).

## 4 Umsetzung

### 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 Softwareentwicklungs-Abteilung, in der 4 Scrum-Teams angesiedelt sind. 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 Prozessüberwachung 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 (es waren genau 15) analysiert und eine Topliste von Schlagwörtern der folgenden Fragestellungen aus den Retrospektiven erstellt:

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

Dazu wurden die Ergebnisse in 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 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 nervt uns noch immer?**

- Hier lassen die Schlagwörter "updat", "erreichbar", "infrastruktur", "jenkin", "test" und "umgebung" auf ein Infrastruktur Problem 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-Modelle in Tabelle 4.1, 4.2 und 4.3 ableiten.

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

Tabelle 4.1: GQM-Modell - Ablenkung der Entwickler

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

Tabelle 4.2: GQM-Modell - Schwachstellen im Prozess



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

Tabelle 4.3: GQM-Modell - 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, in Form einer Umfrage vorgestellt. Die einzelnen Metriken wurden von den Teammitgliedern nach Wichtigkeit mit einer Skala von 1 bis 10 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 10 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 wird. Dies kann zum einen durch die oben genannte Aufgaben-Rückfälligkeit und andererseits durch folgende Metrik abgebildet werden:

- **Anforderungen-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 genannten Systeme im Einsatz. Dadurch sind die Plattform und die zu unterstützenden Systeme vorgegeben. Abbildung 4.1 zeigt die Position in einer möglichen Systemlandschaft.

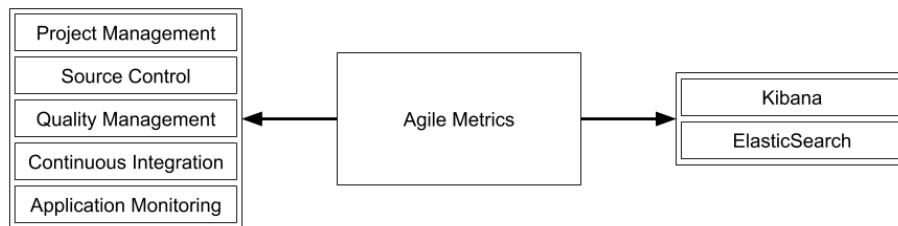


Abbildung 4.1: Position der Software

### 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>36</sup> mit ElasticSearch und Kibana). ElasticSearch ist dabei die Datenbank für die Metriken, genau genommen eine volltext-indizierte Not only SQL (NoSQL)-Datenbank. Kibana dient zur Darstellung der in der ElasticSearch-Datenbank gespeicherten Metriken.

---

<sup>36</sup> *Elastic Stack*.

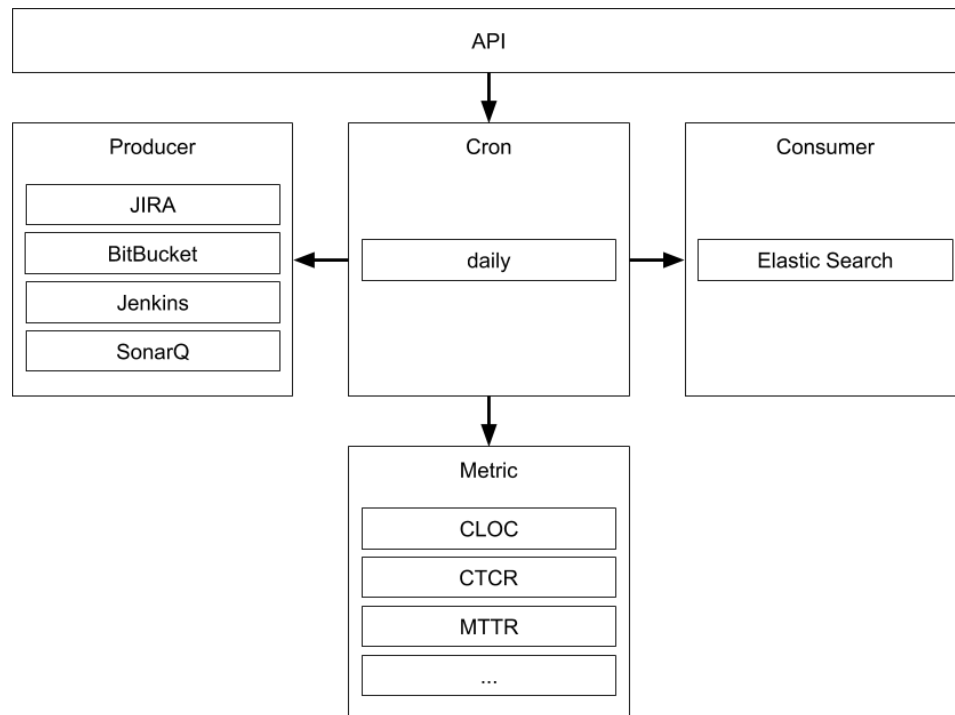


Abbildung 4.2: Übersicht der Software-Architektur

#### 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 (OpenSource) angeboten. Als Lizenz wurde dabei die MIT-Lizenz<sup>37</sup> gewählt, da sie eine einfache und großzügige Lizenz ist, die lediglich die Erhaltung des Urheberrechts und eine Lizenzangabe erfordert.

<sup>37</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>38</sup> in GitHub verwendet. In diesem sind der Quellcode und die Veröffentlichungen der Software zugänglich. Zusätzlich wird über GitHub Pages eine statische Seite<sup>39</sup> für das Projekt bereitgestellt, wie in Abbildung 4.3 ersichtlich.

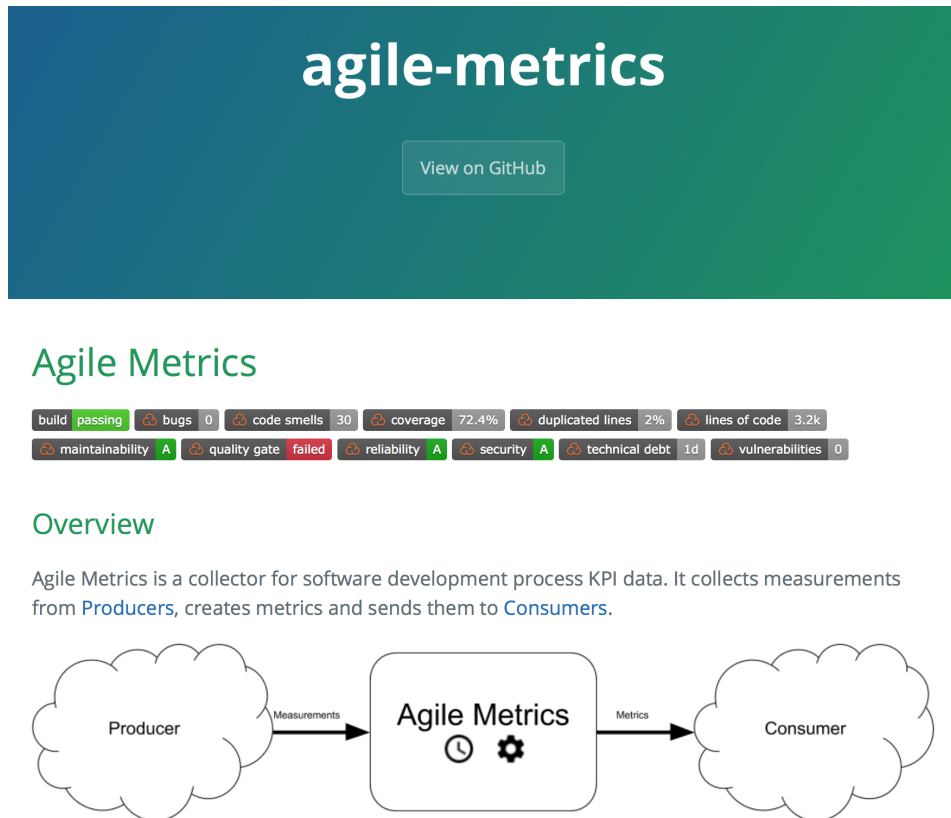


Abbildung 4.3: GitHub Pages - Agile Metrics



Travis CI<sup>40</sup> bietet einen kostenlosen CI-Service und SonarCloud<sup>41</sup> einen kostenlosen QS-Service für quelloffene Projekte.

<sup>38</sup> *GitHub Repository - Agile Metrics*. URL: <https://github.com/DaGrisa/agile-metrics> (besucht am 31.05.2018).

<sup>39</sup> *GitHub Pages - Agile Metrics*. URL: <https://dagrisa.github.io/agile-metrics/> (besucht am 31.05.2018).

<sup>40</sup> *Travis CI - Agile Metrics*. URL: <https://travis-ci.org/DaGrisa/agile-metrics> (besucht am 31.05.2018).

<sup>41</sup> *SonarCloud - Agile Metrics*. URL: <https://sonarcloud.io/dashboard?id=at.grisa.agile-metrics%3Aagile-metrics> (besucht am 31.05.2018).


DaGrisa / agile-metrics

build passing

Current
Branches
Build History
Pull Requests

More options

✓ v0.12-rc3 daniel	rc3	🔗 #63 passed d283d6d	🕒 2 min 40 sec 📅 a day ago	⌂
✓ master daniel	rc3	🔗 #62 passed d283d6d	🕒 4 min 1 sec 📅 a day ago	⌂
✓ master DaGrisa	Merge pull request #1 from DaGrisa/rele	🔗 #61 passed 71b165a	🕒 3 min 2 sec 📅 a day ago	⌂
✓ release/rc3 daniel	cron unit test	🔗 #59 passed 5e1d592	🕒 2 min 25 sec 📅 a day ago	⌂
✓ release/rc3 daniel	bitbucket producer test	🔗 #58 passed c068738	🕒 2 min 52 sec 📅 a day ago	⌂
✓ release/rc3 daniel	finish jira producer tests	🔗 #57 passed 3b993cf	🕒 4 min 38 sec 📅 a day ago	⌂
✓ release/rc3 daniel	jira producer tests	🔗 #56 passed 1ad40ec	🕒 2 min 41 sec 📅 a day ago	⌂
✓ v0.12-rc2 daniel	v0.12-rc2	🔗 #55 passed bb2f826	🕒 3 min 45 sec 📅 2 days ago	⌂

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 durchgeführten Builds mit einigen Informationen.

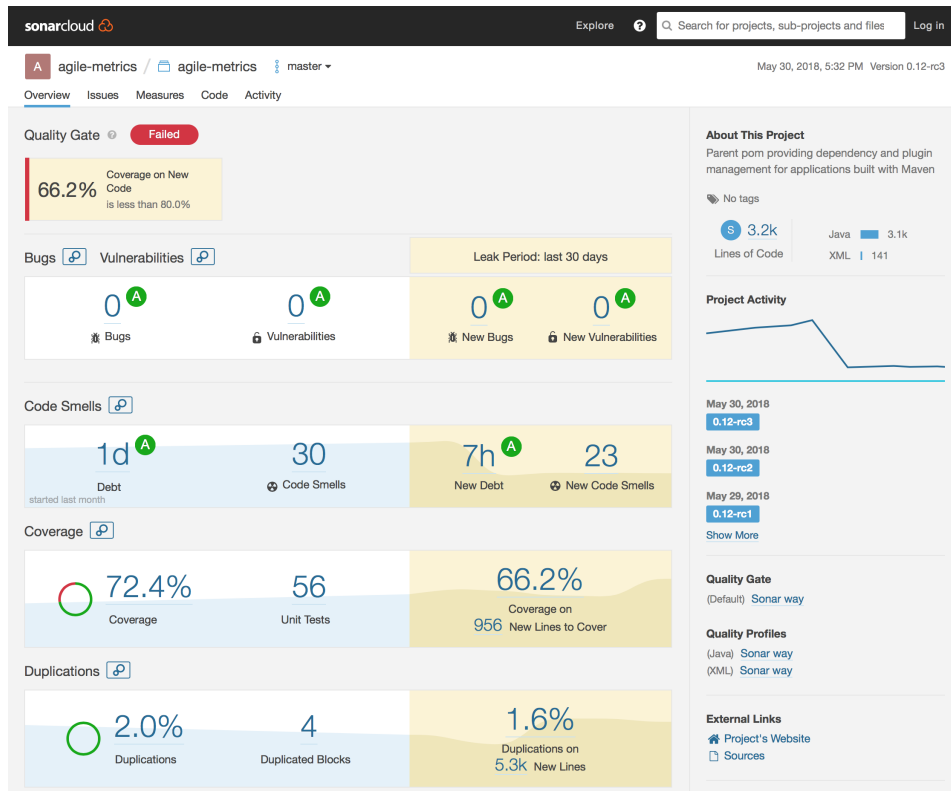


Abbildung 4.5: SonarCloud - Agile Metrics

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 Webproxy 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 laufen 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 dafür 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 ... with PID 28076 (/
... started by ...
[DEBUG] 2018-06-06 14:43:37.276 [main] Application - Running with Spring Boot v2.0.0.RELEASE, Spring v5.0.4.RELEASE
[INFO ] 2018-06-06 14:43:37.277 [main] Application - No active profile set, falling back to default profiles: default
[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 ApplicationPropertie
s{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='ht
tps://...', version='7.5.4', buildNumber='75010', scmInfo=..., buil
dPartnerName='null', serverTitle='...'}
[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 Visualisierte Ergebnisse

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, von dem die Metriken angezeigt werden sollen.

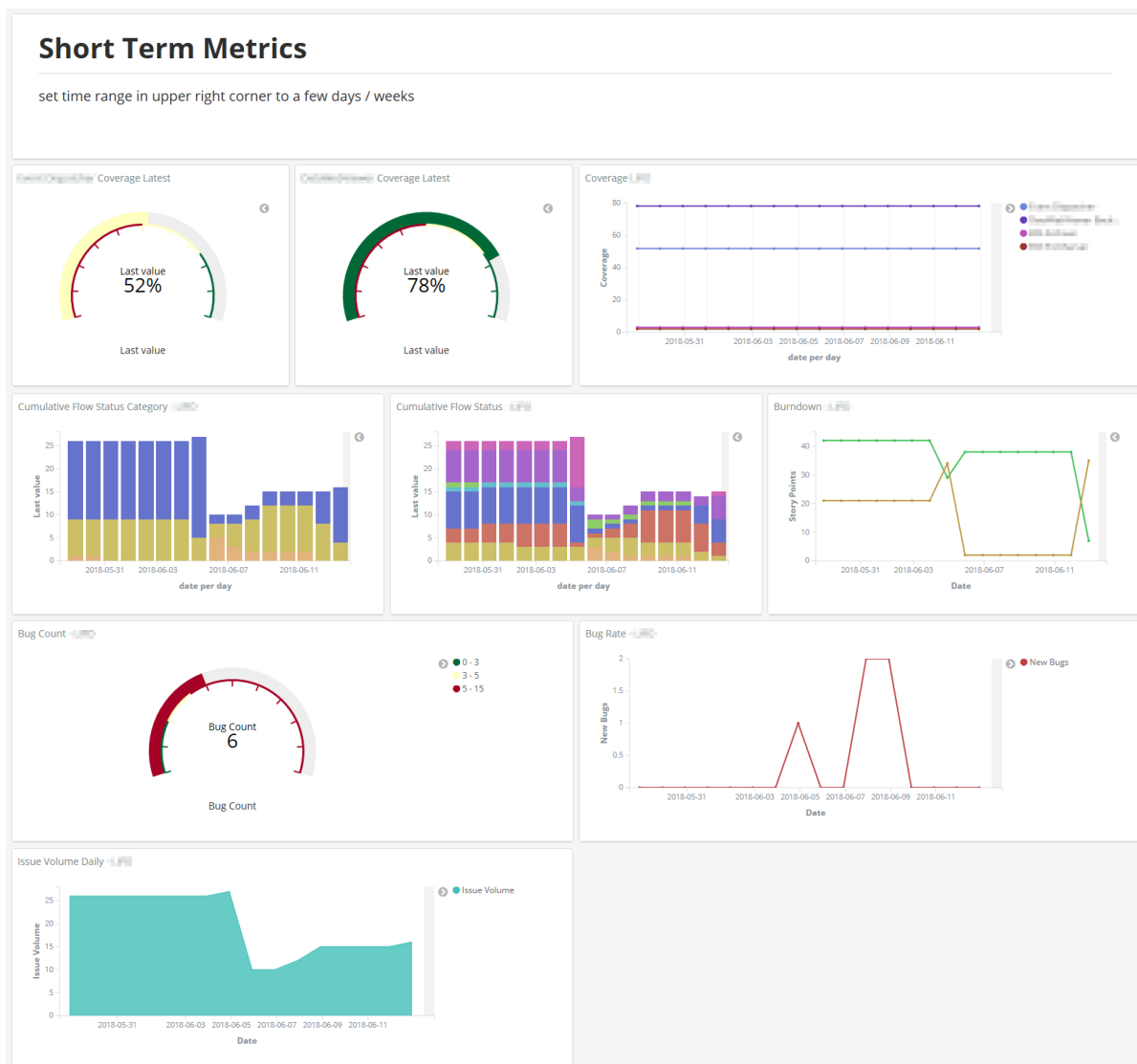


Abbildung 4.7: Teil des Dashboards mit den kurzfristigen Metriken

**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 erledigte und noch offene 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 Statis dargestellt, sondern in einem separaten Diagramm auch noch die Status-Kategorien. Diese können bei sehr vielen unterschiedlichen Statis 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 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.

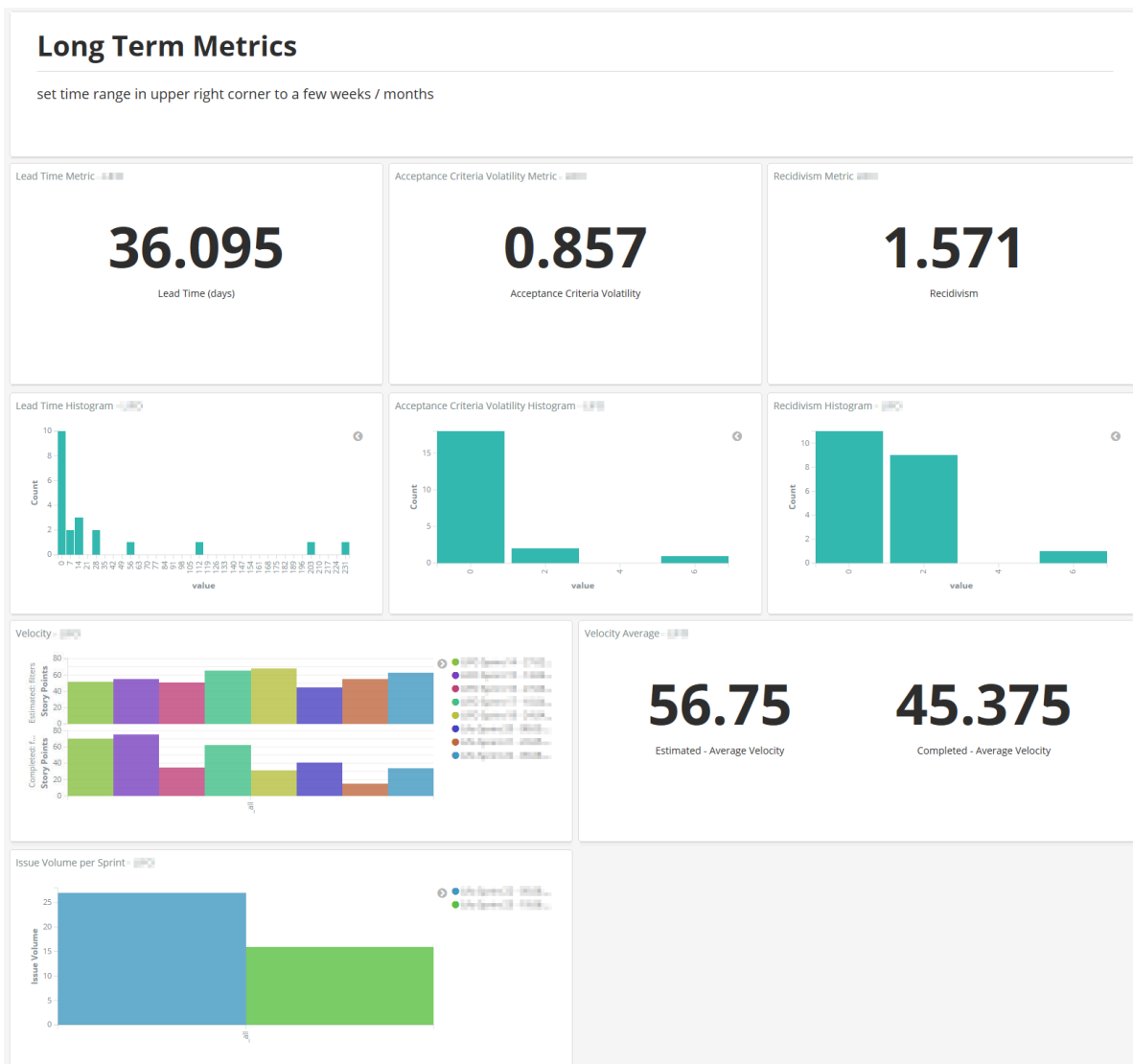


Abbildung 4.8: Teil des Dashboards mit den langfristigen Metriken

**Lead Time**

Die Lead Time wird zum einen als Metrik dargestellt, zum anderen auch als Histogramm. Das hat den Grund, da 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.

**Recisivism**

Die Rückfälligkeit wird auch als Metrik und Histogramm dargestellt.

**Velocity**

Wird als Blockdiagramm dargestellt. Sichtbar sind die geschätzten und die tatsächlich fertiggestellten Story Points pro Sprint.

**Issue Volume**

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

## 4.4.2 Statusschnittstelle

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 Statusschnittstelle

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

## 5.1 Quantitative Evaluierung

Anhand der zeitlichen Entwicklung der gemessenen Metriken wäre eine quantitative Evaluierung möglich. Allerdings ist einerseits der Zeitraum von 2-3 Sprints noch zu gering, um eine Entwicklung auf die Maßnahmen rückschließen zu können, andererseits fehlen bei manchen Metriken die Werte vor der Inbetriebnahme, was eine Änderung ab der Inbetriebnahme ebenfalls schwer erkennen lässt.

## 5.2 Qualitative Evaluierung

Zur qualitativen Evaluierung werden Interviews mit dem Scrum-Master, dem Product Owner und einem Entwickler geführt. Das ermöglicht einen Einblick aus Management-sicht (Product Owner) und aus Prozesssicht (Scrum Master und Entwickler), welche womöglich ganz unterschiedliche Anforderungen an das Produkt 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. Gibts 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 XYZ. Interne und vertrauliche Informationen wurden mit \*\*\* ersetzt.

### 5.2.2 Interview-Antworten

# 6 Schlussfolgerungen

Effektivität des Modells, Erkenntnisse aus der Einführung bei Gebrüder Weiss



# 7 Zusammenfassung

Erkenntnisse und Ausblick

# Literatur

- 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. *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/SMP0-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).



# Anhang

## A.1 Metriken aus dem Entwicklungsprozess

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

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 Softwareartefakt zu erstellen?</i> <i>Wie verändert sich die Dauer der Erstellung eines Softwareartefakts 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 Softwareartefakt erstellt?</i>
<b>Test Reports</b>	Anzahl erfolgreicher und fehlerhafter Tests, Gesamtdauer der Tests. <i>Wie lange dauert ein kompletter Testdurchlauf?</i> <i>Gibt es Tests, die optimiert werden müssen?</i> <i>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?</i> <i>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?</i> <i>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>Benutzeranzahl</b>	Anzahl gleichzeitiger Benutzer in der Applikation über die Zeit.
<i>Wie entwickeln sich die Benutzerzahlen 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 Benutzer 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 Benutzer 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. erledigt (2)
10. information (2)
11. issue (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. durchgeführt (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. issues (3)
9. planungswoch (3)
10. retro (3)
11. richtlini (3)
12. system (3)

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

## **Was können wir besser machen?**

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

## Was nervt uns noch immer?

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

## 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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<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	Developer	Durchschnitt
Changed Lines Of Code (CLOC)	3	4	3	7	3	8	7	5,00
CLOC pro Entwickler	3	2	6	4	6	7	3	4,43
CLOC pro Commit	4	4	8	4	3	4	7	4,86
Commits	3	6	7	8	2	8	8	6,00
Commits pro Entwickler	3	2	3	5	2	8	4	3,86
Kommentare pro Commit	6	7	5	3	3	7	3	4,86
Pull Requests	6	9	9	10	3	8	3	6,86
Gemergte Pull Requests	3	7	9	10	3	8	8	6,86
Abgelehnte Pull Requests	3	10	7	8	3	9	7	6,71
Kommentare pro Pull Request	5	9	4	6	4	3	4	5,00
Burn Down	10	10	10	8	6	10	9	9,00
Velocity	10	10	9	10	5	10	9	9,00
Cumulative Flow	10	9	9	6	7	8	9	8,29
Lead Time	10	10	9	6	4	10	8	8,14
Bug Counts	10	9	7	2	6	10	9	7,57
Bug-Erzeugungsrate	10	8	5	8	3	10	8	7,43
Bug-Fertigstellungsrate	10	6	5	8	3	7	8	6,71
Aufgaben-Volumen	10	10	8	7	8	8	9	8,57
Aufgaben-Rückfälligkeit	7	10	9	3	7	8	9	7,57
Build-Dauer	5	3	3	1	4	8	8	4,57
Build-Status	10	8	3	4	4	8	9	6,57
Build-Frequenz	5	2	3	4	5	4	8	4,43
Test Reports	10	4	6	4	6	10	9	7,00
Code Coverage	10	4	7	8	6	10	9	7,71
Stresstests oder Benchmarking	10	4	9	5	7	10	10	7,86
CPU Nutzung	7	4	2	10	6	10	9	6,86
Heap Size	7	4	2	10	6	10	9	6,86
Fehlerraten	10	3	3	7	6	7	10	6,57
Antwortzeiten	10	2	2	7	6	8	9	6,29
Benutzeranzahl	10	2	2	5	6	10	8	6,14
Aufenthaltsdauer	10	2	2	5	3	10	8	5,71
Conversion Rate	10	1	2	9	3	8	9	6,00
Verfügbarkeit	10	4	2	9	6	10	10	7,29
Semantisches Logging	10	1	2	3	4	8	8	5,14

#### 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?

# Eidesstattliche Erklärung

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

Dornbirn, am [Tag. Monat Jahr anführen]

[Vor- und Nachname Verfasser/in]