Fakultät für Informatik und Mathematik 07

Bachelorarbeit

über das Thema

Microservices und technologische Heterogenität
Entwicklung einer sprachunabhängigen Microservice Framework Evaluationsmethode

Autor: René Zarwel
zarwel@hm.edu

Prüfer: Prof. Dr. Hammerschall

Abgabedatum: 13.03.2017

I Kurzfassung

Insert Abstract Here

Abstract

Insert Abstract Here

II Inhaltsverzeichnis

I Kurzfassung **I**

II Inhaltsverzeichnis **II**

III Abbildungsverzeichnis **IV**

IV Tabellenverzeichnis **V**

V Listing-Verzeichnis **V**

VI Abkürzungsverzeichnis **VI**

VII Glossar **VII**

1 Einleitung **1**

2 Microservices **2**
2.1 Microservice Frameworks . . . 3

3 Qualitätsbewertung von Softwarearchitektur **6**
3.1 Qualitätsmerkmale der Softwarearchitektur . . . 7
3.2 Soll-Ist-Vergleich . . . 8
3.3 Architecture Tradeoff Analysis Method (ATAM) . . . 9
3.3.1 Vorbereitung . . . 11
3.3.2 Kickoff-Phase . . . 12
3.3.3 Bewertungsphase . . . 13
3.3.4 Abschlussphase . . . 16
3.4 Software Architecture Evaluation Model (SAEM) . . . 16
3.4.1 Spezifikation Qualität . . . 17
3.4.2 Bestimmung von Metriken . . . 18
3.4.3 Evaluation . . . 19
3.5 Architekturbewertungsmethoden und Microservice Frameworks . . . 19

4 Microservice Framework Evaluation Method (MFEM) **22**
4.1 Kickoff-Phase . . . 23
4.2 Analysephase . . . 24
4.2.1 Architektur Analyse und Wahl der Anforderungen . . . 24
4.2.2 Metriken definieren über Goal Question Metrik (GQM) . . . 28
4.3 Evaluationsphase . . . 30
4.3.1 Evaluation definieren . . . 30
4.3.2 Metriken zuordnen . . . 37
4.3.3 Evaluation durchführen . . . 37
4.4 Abschlussphase . . . 38
4.4.1 Ergebnisse auswerten . . . 38
4.4.2 Resultat präsentieren . . . 41

5 Evaluation der Methode an Beispielen 42
5.1 Wahl der Kandidaten . . . 42
5.2 Kickoff-Phase . . . 43
5.2.1 Resümee Kickoff . . . 45
5.3 Analysephase . . . 46
5.3.1 Architektur Analyse und Wahl der Anforderungen . . . 46
5.3.2 Metriken definieren . . . 47
5.3.3 Resümee Analysephase . . . 49
5.4 Evaluationsphase . . . 49
5.4.1 Evaluation definieren . . . 49
5.4.2 Metriken zuordnen . . . 51
5.4.3 Evaluation durchführen: Spring Boot . . . 53
5.4.4 Evaluation durchführen: Go-Kit . . . 58
5.4.5 Resümee Evaluationsphase . . . 62
5.5 Abschlussphase . . . 63
5.5.1 Resümee Abschlussphase . . . 65
5.6 Gesamtauswertung Microservice Framework Evaluation Method (MFEM) 66

6 Fazit und Ausblick 67
6.1 Microservice Framework Evaluation Method (MFEM) Anpassung/Erweiterung . . . 67
6.2 Ausblick . . . 67

7 Quellenverzeichnis 68

Todo list 70

Selbstständigkeitserklärung I

Anhang II

A MFEM - Vollständiger Quality Utility Tree der Basisanforderungen II

III Abbildungsverzeichnis

Abb. 2.1 Vorteile Microservices 2
Abb. 2.2 Anforderungen und Frameworks 4
Abb. 3.1 Ziele Architekturbewertung 7
Abb. 3.2 Phasen von ATAM 11
Abb. 3.3 Beispiel Qualitätsbaumes 14
Abb. 3.4 Qualitätsbaum und Szenarien 15
Abb. 3.5 Qualitätsmodell der ISO/IEC 25010 17
Abb. 3.6 Beispiele GQM 19
Abb. 3.7 Bewertung der drei Frameworkseiten 20
Abb. 4.1 Ablaufschema MFEM 23
Abb. 4.2 MFEM-Analysephase 24
Abb. 4.3 Servicediscovery Typen 25
Abb. 4.4 MFEM Quality Utility Tree Beispiel 27
Abb. 4.5 MFEM Quality Utility Tree Beispiel mit Prioritäten 28
Abb. 4.6 MFEM Qualitätsbaum erweitert um Metriken 30
Abb. 4.7 MFEM-Evaluationsphase 30
Abb. 4.8 MFEM Kreislauf Evaluation 32
Abb. 4.9 Installation eines Frameworks 33
Abb. 4.10 Einfacher Service 34
Abb. 4.11 Datenmodell 34
Abb. 4.12 Service mit Datenmodell 35
Abb. 4.13 Zuordnung Metriken 37
Abb. 4.14 MFEM-Abschlussphase 38
Abb. 4.15 Netzdiagramm 39
Abb. 4.16 Auswertung Beispiel 41
Abb. 5.1 Architektur für die Evaluation 44
Abb. 5.2 Ablauf Service Discovery 45
Abb. 5.3 Ablauf kaskadierter Anfrage 45
Abb. 5.4 Quality Utility Tree Evaluation 47
Abb. 5.5 Messgruppen für Evaluation 51
Abb. 5.6 Auswertung Spring Wartbarkeit 63
Abb. 5.7 Gesamtergebnis von Spring und Go-Kit 65

IV Tabellenverzeichnis

Tab. 4.1 Beispiele für Ordinalskalen 29
Tab. 4.2 Anforderungsprofile 36
Tab. 4.3 Ordinalskala Normalisierung 40
Tab. 4.4 Normalisierung quantitativer Daten 40
Tab. 5.1 Evaluation: Goal Question Metrik Tabelle 49
Tab. 5.2 Anforderungsprofile 50
Tab. 5.3 Metriken Szenarios 52
Tab. 5.4 Metriken Objektive Evaluation 53
Tab. 5.5 Szenario 1 Ergebnis Spring 54
Tab. 5.6 Szenario 2 Ergebnis Spring 55
Tab. 5.7 Szenario 3 Ergebnis Spring 57
Tab. 5.8 Quantitative Evaluation Ergebnis Spring 58
Tab. 5.9 Szenario 1 Ergebnis GoKit 59
Tab. 5.10 Szenario 2 Ergebnis GoKit 60
Tab. 5.11 Szenario 3 Ergebnis GoKit 61
Tab. 5.12 Quantitative Evaluation Ergebnis Go-Kit 62

V Listing-Verzeichnis

Lst. 1 „Hello-Word“ in Spring 55
Lst. 2 Umsetzung der Entität „Department“ mittels JPA 56
Lst. 3 Erstellung der REST-Schnittstelle für das „Department“ über ein Interface 56
Lst. 4 „Hello-Word“ in Go-Kit 59

VI Abkürzungsverzeichnis

SEI	Software Engineering Institute
SAAM	Software Architecture Analysis Method
ATAM	Architecture Tradeoff Analysis Method
SAEM	Software Architecture Evaluation Model
GQM	Goal Question Metrik
QUT	Quality Utility Tree
MFEM	Microservice Framework Evaluation Method
HATEOAS	Hypermedia As The Engine Of Application State
REST	Representational State Transfer
CW	Cognitive Walkthrough
JWT	JSON Web Token
JPA	Java Persistence API
LOC	Lines of Code

VII Glossar

Anforderung

Die Grundlage einer jeden Bewertung sind spezifisch definierte Anforderungen. Diese stellen die notwendige Beschaffenheit oder Fähigkeit eines Produktes dar, dass es für die Abnahme erfüllen muss. Dabei ist eine Unterscheidung von funktionalen und nicht-funktionalen Anforderungen weit verbreitet. Funktionale Anforderungen bestimmen, *was* das Produkt tun soll [17]. Nicht-funktionalen Anforderungen beschreiben *in welcher Qualität* die zu erbringenden Leistung erfolgen soll [17]. In der Regel werden diese Anforderungen durch Verträge, Normen oder Spezifikationen festgelegt.

Framework

Ein Framework bietet einen Rahmen zur Erstellung einer Anwendung. Dabei ist es selber noch kein fertiges Programm. Es gibt vielmehr ein Grundgerüst vor und unterstützt den Entwickler soweit es geht. Dabei bietet es eine Designgrundlage und beeinflusst maßgeblich die Architektur.

Merkmale

Merkmale sind charakteristischen Kennzeichen eines Produktes. Im allgemeinen geben sie an, wie sich das Produkt von anderen unterscheidet. Zudem ergibt eine Messung der Merkmale am Produkt Aufschluss über die Erfüllung der Anforderungen.

Qualität

Nach der DIN EN ISO 9000:2015-11 ist Qualität, der „Grad, in dem ein Satz inhärenter Merkmale eines Objekts Anforderungen erfüllt“. Sie gibt somit das Ausmaß an, wie ein Produkt den bestehenden Anforderungen nachkommt. Dabei sind „inhärente Merkmale“ die Merkmale, die dem Produkt innewohnen. Sie grenzen sich damit von den zugeordneten Merkmalen ab, wie z. B. dem Preis.

Ziele

Ziele definieren, was mit dem Produkt erreicht werden soll. Im Gegensatz zu Anforderungen geht es dabei weniger um die technischen Aspekte, als das zugrundeliegende Problem, dass mit dem Produkt gelöst werden soll.

1 Einleitung

Mit der Microservice Architektur lassen sich verschiedenste Technologien vermischen. Jeder Service kann mit unterschiedlichen Frameworks und Programmiersprachen entwickelt werden.
Diese Freiheit spiegelt sich im Auswahlprozess der Softwareentwickler wider. Sie laufen unter Umständen einem Hype hinterher [13] und die Erwartungen an das neue Framework sind häufig stark überzogen. Dieser Investition in die neue Technologie folgt Ernüchterung. Im Detail sind viele Funktionen oft nicht so, wie man es erwartet hat und im schlimmsten Fall schlägt die Entwicklung fehl.

Zudem ist die Microservice Architektur nicht eindeutig definiert [22, S. 11]. Jede Ausprägung ist anderes und folgt den spezifischen Anforderungen oder dem Geschmack des Architekten. So kann ein Framework für eine Ausprägung ideal und für eine andere ein regelrechter Fehlgriff sein.

Dies bietet jedoch keinen Anlass sämtliche unbekannten Frameworks zu meiden und nur die jahrelang Etablierten zu nutzen. Neue Technologien können auch eine Chance bieten und echten Mehrwert produzieren. Sie lösen mitunter ein gravierendes Problem oder die Entwicklungszeit verringert sich immens. So lohnt sich gerade in einer Microservice Architektur das Risiko einzugehen, da der Fehlschlag eines einzelnen Services vertretbar ist.

Um dieses Risiko weiter zu minimieren und das richtige Framework zu finden, soll in dieser Arbeit eine Bewertungsmethode für Microservice Frameworks etabliert werden. Damit verbunden, sollen folgende Eigenschaften von der Methode erfüllt werden:

Sprachunabhängig So wie die Erstellung eines Microservices in jeder Programmiersprache erfolgen kann, soll auch die Methode unabhängig sein.

Kein Vorwissen nötig Damit sämtliche Frameworks einbezogen werden können, darf die Methode vom Bewertungsteam kein Vorwissen über das Framework verlangen.

Spezifisch Die Methode muss in Abhängigkeit zu einer spezifischen Ausprägung der Architektur angepasst werden können.

Vergleichbarkeit Das Ergebnis der Methode soll sowohl für sich stehen, als auch ein Vergleich zu anderen Frameworks eröffnen.

Mit der Bewertung lässt sich eine Microservice Architektur aufbauen, die technologisch Heterogen ist und bei der Chancen genutzt sowie Risiken minimiert werden.

2 Microservices

„*Do One Thing and Do It Well*“
- Douglas McIlroy (A Quarter Century of Unix)

Diesem Grundsatz aus der Unix Philosophie folgt auch die Microservice Architektur. Sie bietet einen Ansatz zur Modularisierung von Software. Doch im Gegensatz zur konventionellen Modularisierung von großen Systemen, sind die sogenannten Microservices eigene Programme [22, S. 10], die über eine einheitliche Schnittstelle, wie z. B. Representational State Transfer (REST) oder Messaging, kommunizieren. Ein einzelner Service ist dabei relativ klein, weitgehend entkoppelt und übernimmt nur eine kleine Aufgabe.
Doch ist dies nur ein Aspekt des Entwurfsmusters. Die Abbildung 2.1 zeigt hierzu die wesentlichen Vorteile einer Microservice Architektur.

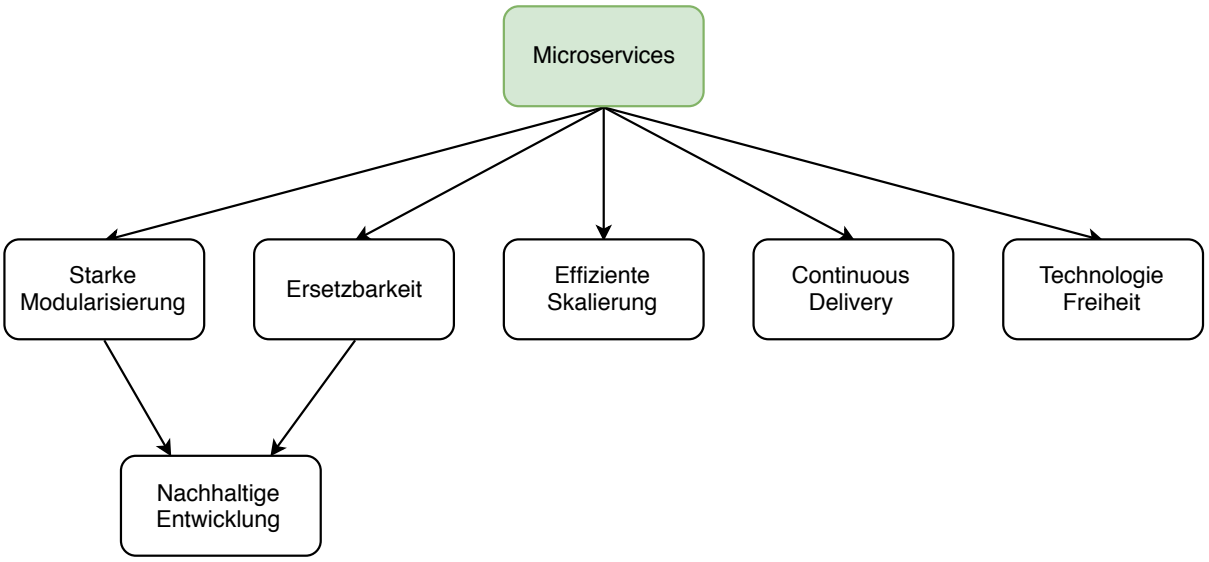

Abbildung 2.1: Wesentliche Vorteile von Microservices [22, S. 12]

Starke Modularisierung Durch die strikte Trennung der einzelnen Services in eigene Programme, schleichen sich kaum unerwünschte Abhängigkeiten unter den Modulen ein.

Ersetzbarkeit Solange die Schnittstelle eines Services gleich bleibt, kann dieser einfach ersetzt werden. Dabei muss er weder die Code-Basis noch die Technologie des be-

stehenden Services verwenden.

Nachhaltige Entwicklung Mit der starken Modularisierung und der Ersetzbarkeit können technische Schulden vermieden und so eine nachhaltige Entwicklung geschaffen werden.

Effiziente Skalierung Durch die starke Trennung lassen sich die einzelnen Module unabhängig voneinander skalieren.

Continuous Delivery Dank der kleineren Services lässt sich einfacher eine Continuous-Delivery-Pipeline aufbauen. So können einzelne Services schneller in die Produktion gebracht werden, wodurch sich die Time-To-Market Zeit verringert.

Technologie Freiheit Einzelne Services können unterschiedliche Technologien verwenden. Es besteht dabei keine Einschränkung auf eine Programmiersprache oder Plattform.

Besonders die technologische Freiheit ist an dieser Stelle interessant. Durch die Aufteilung in einzelnen Services, können für jeden Microservice unabhängig die Programmiersprache und Frameworks gewählt werden. So lassen sich neue Technologien in einem Service erproben, ohne dass andere Services davon betroffen sind. Dies senkt das Risiko für die Einführung neuer Technologien und die Kosten bleiben kalkulierbar [22, S. 13], da nur in einem kleinen Rahmen eingeführt und getestet wird.
Die Technologie-Entscheidung könnte somit auf die kleine Teilaufgabe, die der Service übernimmt, reduziert werden. Aber auch die Fähigkeiten des Entwickler-Teams spielen eine Rolle. So kann z. B. für das eine Team NodeJS gewählt werden, da überwiegend Javascript Entwickler vertreten sind, und für das andere Team Spring, da dieses hauptsächlich Java beherrschen.

2.1 Microservice Frameworks

Ein Microservice Framework soll die Erstellung eines Services vereinfachen. Es unterstützt den Entwickler bei sich wiederholenden Tätigkeiten, wie z. B. der Erstellung eines REST-Endpunktes, und bringt viele häufig benötigte Funktionen mit. Dies könnte z. B. eine Authentifizierung oder Caching sein.
Dabei ist es darauf ausgelegt, möglichst schnell Ergebnisse zu erzielen und einen lauffähigen Microservice zu erstellen. Dies ist wichtig, da aufgrund der Vielzahl von Services die Erstellung eines einzigen nicht zu komplex sein sollte und der Fokus auf der Geschäftslogik liegt.

Dank der Technologiefreiheit kann für jeden Service ein eigenes Framework gewählt werden. Dabei stellt sich die Frage, welches für die Lösung der Teilaufgabe am besten geeignet ist.
Da es einen Rahmen für die Entwicklung zur Verfügung stellt und die Programmiersprache vorgibt, ist diese Entscheidung maßgeblich für den weiteren Entwurf des Services. Aus diesem Grund sollte die Wahl nicht gänzlich unbedacht sein. Auch wen mit der Ersetzbarkeit die Hürde zum Austausch gering ist, bedeutet ein Fehlschlag mitunter mehrere Wochen vergeudetet Entwicklungszeit.

Um das richtige Framework zu finden, müssen die Anforderungen an den Service festgelegt sein. Nur wenn es diese erfüllt, macht eine Umsetzung mit dem Framework Sinn. Welche Anforderungen ein Service zu erfüllen hat, ergibt sich aus der Aufgabe, die mit dem Service umgesetzt werden soll. Dies können sowohl fachliche Aufgaben, wie eine Kundenverwaltung, oder technische, wie das Sammeln von Log-Daten der weiteren Services, sein.
Aus der Architektur ergibt sich die Aufteilung der Aufgaben. Dabei ist es wichtig, dass bei Änderungen und neuen Features nur ein Service angepasst werden muss. Hierzu ist es essentiell, dass jeder Service eine fachliche bzw. technische Einheit bildet.
Durch die Anforderungen des Gesamtsystems, die hauptsächlich vom Auftraggeber vorgegeben sind, ergibt sich die Architektur. Dieser Weg von den Anforderungen bis zur Wahl der passenden Frameworks ist in Abbildung 2.2 nochmals dargestellt.

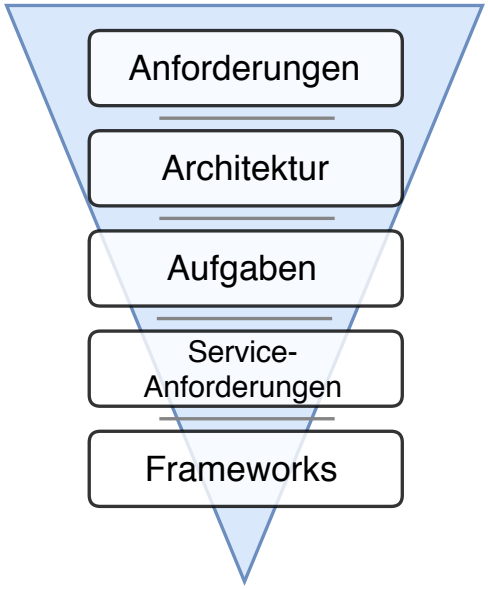

Abbildung 2.2: Der Weg von den Anforderungen zu den passenden Frameworks für die Services

Daraus wird ersichtlich, dass sich nur dann die passenden Serviceanforderungen für die Bewertung eines Frameworks ergeben, wenn die Architektur den Anforderungen entspricht. Aus diesem Grund sollte die Architekturqualität vorher bewiesen sein. Zudem bietet es sich an, die Bewertung der Frameworks an die Qualitätsbewertung von Softwarearchitektur anzulehnen.

3 Qualitätsbewertung von Softwarearchitektur

„*You cannot control what you cannot measure*“
- Tom DeMarco (Controlling Software Projects)

Die Aussage beschreibt sehr gut, dass ein IT-Projekt besser auf die Projektziele zusteuern kann, wenn Messwerte erhoben werden. Diese lassen sich z. B. in Diagrammen sehr gut darstellen und können zumeist einfach sowie unmissverständlich interpretiert werden. Neben dem aktuellen Projektstatus lassen sich so auch Trends erkennen. Diese helfen dabei rechtzeitig Abweichungen festzustellen und gegebenenfalls korrigierende Maßnahmen einzuleiten [19]. Auch wenn DeMarco 2009 seine Aussage relativiert hat [4], bleibt sie in Bezug auf die Softwarearchitektur stimmig. Gerade bei einem so entscheidendem Punkt, wie der Architektur, ist es wichtig frühzeitig Risiken und Qualitätsabweichungen festzustellen. Die Architektur ist ein sehr kritischer und wesentlicher Bestandteil im Entwicklungsprozess von einer Software. Ihrer Beschaffenheit nach, kann sie nur schwer und mit hohen Kosten verändert werden. Durch Zeit- und Kostendruck wird dies leider in der Praxis häufig erst in späten Entwicklungsphasen durchgeführt. In einem sogenannten „Audit“ oder „Review“ werden bereits produktiv laufende Systeme bewertet [19]. Sollten sich hier starke Abweichungen gegenüber den Anforderungen zeigen, kann sich dies zu einem großen Problem avancieren, hohe Kosten verursachen und Kunden verärgern. Aus diesem Grund ist die Qualität einer Architektur sehr entscheidend und sollte stichhaltig nachgewiesen sein. Um sie zu bestimmen und Restrisiken minimiert zu können, gibt es Methoden zur Qualitätsbewertung von Softwarearchitektur. Neben diesem Ziel gibt es noch weitere positive Auswirkungen, die für den Einsatz der Architekturbewertung sprechen. (Siehe Bild 3.1)

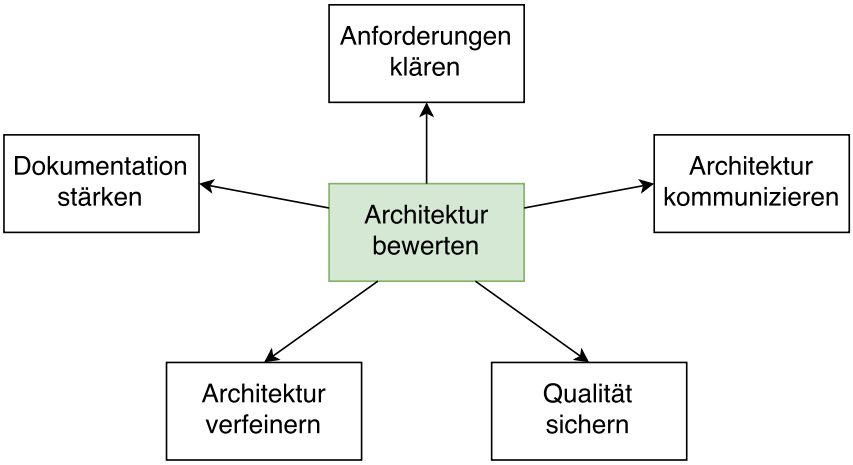

Abbildung 3.1: Allgemeine Ziele von Architekturbewertung.

Die Bewertung kann sich dabei auf die im Softwareprojekt entstehenden Artefakte stützen. Beispielsweise sind Anforderungen, Architekturen, Diagramme, Quellcode und andere Dokumente zu nennen. Diese Artefakte können dabei quantitativ oder qualitativ bewertet werden. So kann z. B. der Quellcode mittels Metriken quantitativ, d. h. in reinen Zahlen, bewertet werden. Dies lässt sich meist einfach und automatisiert, siehe SonarQube[1], erfassen und kann sehr gut reproduziert sowie verglichen werden. Der Einsatz sollte jedoch gut überlegt sein, damit nicht ziellos gemessen wird. Andere Artefakte entziehen sich dieser Bewertung und können nur auf ihre Güte hin, also qualitativ, bewertet werden. Diese, teils auch subjektiven, Bewertungen setzen einen größeren Aufwand voraus und lassen sich schwer vergleichen. Zu letzterem zählt die Softwarearchitektur.

3.1 Qualitätsmerkmale der Softwarearchitektur

Auch bei der Architekturbewertung sind die Anforderungen das Fundament. Sie charakterisieren die gewünschten Eigenschaften eines Systems [19]. Bei der Softwarearchitektur gibt es eine Vielzahl an Qualitätsmerkmale anhand derer Anforderungen bestimmt und definiert werden können [19; 16] :

Funktionalität Mit der Funktionalität wird bestimmt, ob die funktionalen Anforderungen nicht nur vollständig erfüllt werden, sondern auch richtig und angemessen umgesetzt sind.

Zuverlässigkeit Aus der Zuverlässigkeit ergibt sich die Fähigkeit eines Systems, die Funktion innerhalb einer Zeitspanne um-

[1] `https://www.sonarqube.org`

zusetzen und dabei auf etwaige Fehler zu reagieren bzw. diese zu verhindern.

Leistung Die Leistung (Performance) gibt an, wie schnell das System reagiert und wie viele Eingaben pro Zeiteinheit verarbeitet werden können. Diese Anforderungen lassen sich meist über Benchmarks testen.

Flexibilität Eine Architektur ist flexibel, wenn sie angepasst und erweitert werden kann, ohne dabei das gesamte Konstrukt zu zerstören.

Übertragbarkeit Durch die Übertragbarkeit lässt sich das System auch unter verschiedensten Voraussetzungen betreiben. (Hard- und Softwareumgebungen)

Unterteilbarkeit Die Unterteilbarkeit ermöglicht eine einfache Aufteilung in Teilsysteme. So können einzelne Teile unabhängig entwickelt oder sogar betrieben werden.

Konzeptuelle Integrität Das Konzept sollte sich auf allen Ebenen widerspiegeln und sich durch ein einheitliches Design präsentieren. Ähnliche Dinge sollen dabei in ähnlicher Art und Weise gelöst werden.

Machbarkeit Es muss möglich sein, das System mit vorhandenen Ressourcen (Technologie, Budget, usw.) umzusetzen.

Diese Aufzählung stellt hierbei eine Auswahl an verschiedenen allgemeinen Qualitätsmerkmalen dar und erhebt dabei keinen Anspruch auf Vollständigkeit.

3.2 Soll-Ist-Vergleich

Mit den Anforderungen kann ein Soll-Ist-Vergleich anhand der Artefakte durchgeführt werden. Jede einzelne Anforderung wird dabei mit Plänen, Dokumentation oder Modellen verglichen und bewertet. D. h. es wird geprüft, ob das geplante System den Anforderungen entsprechen wird. Wichtig ist, dass die geforderten Eigenschaften möglichst feingranular sind. Je detaillierter die Anforderungen vorliegen, desto geringer ist das Restrisiko einzelne Punkte zu vergessen oder unscharf zu bewerten.

Das Ergebnis dieser Prüfung kann folgendermaßen aussehen [19]:

Soll = Ist Das Soll wird erfüllt und die Architektur besitzt alle geforderten Eigenschaften.

Soll ≈ Ist Das Soll wird nur teilweise erreicht und es werden Kompromisse geschlossen. (Verbesserung einzelner Anforderungen)

Soll ≠ Ist Das Soll wird nicht erfüllt und es ergibt sich ein Risiko für das System

Die Architekturbewertung ist somit ein relativer Vergleich in Hinblick auf spezifische Kriterien und liefert keine absolute Aussage über die Qualität. Vielmehr identifizieren sich aus einer Architekturbewertung Risiken, die die Architekturentscheidungen in der Entwurfsphase mit sich gebracht haben.

Diesen Ansatz verfolgt auch Software Architecture Analysis Method (SAAM). Das Software Engineering Institute (SEI), eine US-Bundeseinrichtung für die Verbesserung von Software-Engineering Praktiken, hat mit dieser Methode den Grundstein der Architekturbewertung gelegt. Viele der heute bekannten Methoden basieren auf der SAAM und erweitern diese um spezielle Sichten oder fokussieren sich auf ein spezielles Qualitätsmerkmal. Mit der Architecture Tradeoff Analysis Method (ATAM) wurde die SAAM von der SEI weiterentwickelt und stellt heute die führende Methode zur Architekturbewertung dar [9].

3.3 Architecture Tradeoff Analysis Method (ATAM)

Im folgenden Abschnitt wird ATAM genauer vorgestellt. Das Ziel dieser Bewertungsmethode ist nicht eine exakte Vorhersage über die zu erwartende Qualität der Software zu treffen. Das ist nahezu unmöglich bei der frühen Entwurfsphase, da noch nicht genügend Informationen vorliegen. Vielmehr soll der Blick auf einzelne Qualitätsmerkmale und dessen Abhängigkeit zu gewissen Architektur-Entwurfsentscheidungen geschärft werden [16]. Mit diesem Wissen können einzelne Entscheidungen überdacht, genauer modelliert oder angepasst werden, um das gewünschte Qualitätsziel zu erreichen. Darüber hinaus wird auch die Dokumentation der Architektur verbessert, da alle Qualitätsaspekte genauer untersucht werden.

Das Ziel von ATAM ist eine Dokumentation von Risiken (Risks), Sensitivitätspunkte (sensitivity points) und Kompromisspunkte(tradeoff points), die durch eine genauere Analyse der Architektur [16] ermittelt werden konnten. Risiken stellen nicht getroffene Architekturentscheidungen oder Eigenschaften der Architektur,die nicht vollständig verstanden wurden, dar. So kann z. B. der verwendete Datenbanktyp noch ungeklärt oder die Auswirkungen einer zentral geführten Komponente unklar sein.

Sensitivitätspunkte sind starke Abhängigkeiten messbarer Qualitätsmerkmale. Diese be-

ziehen sich auf die Auslegung von einzelnen Komponenten der Architektur. Ein Beispiel für Sensitivitätspunkte ist ein Engpass zwischen zwei Modulen. D. h. wenn der Kommunikationskanal zweier Module, wovon mindestens eins essenziell für das Gesamtsystem ist, zu gering ausgelegt wurde, dann kann dieser für einen niedrigen Durchsatz des gesamten Systems verantwortlich sein. In anderen Worten beeinflusst die Dimensionierung einer einzelnen Komponente direkt das Gesamtsystem, was einen Sensitivitätspunkt darstellt.

Zusätzlich deckt der Kompromisspunkt Sensitivitätspunkte auf, die sich entgegen wirken. Würde man z. B. den zuvor genannten Engpass breiter auslegen, kann unter Umständen dies die Zuverlässigkeit des Gesamtsystems verschlechtern. Ein angebundenes Modul verwirft möglicherweise bei einer starken Belastung einige Anfragen, wodurch ein korrektes Ergebnis nicht mehr garantiert werden kann. So wirken sich diese beiden Sensitivitätspunkte entgegen und es muss ein Kompromiss gebildet werden.

Mit den Risiken, Sensitivitäts- und Kompromisspunkten kann die Architektur mit gezielteren Analysen, Erstellung von Prototypen und weiteren Entwürfen stark verbessert werden.

Die Grundvoraussetzung zum Erreichen dieses Zieles ist eine detaillierte Definition der Qualitätsanforderungen und eine genaue Spezifikation der Architektur sowie dessen zugrunde liegenden Entscheidungen [16]. In der Praxis ist es leider nicht unüblich, dass Ziele und Architekturdetails noch unklar oder mehrdeutig sind. Aus diesem Grund ist ein wichtiges Ziel von ATAM auch dies genauer zu definieren und eindeutig festzuhalten. Um die Erreichung sämtlicher Ziele zu unterstützen, ist ATAM in mehrere Phasen aufgeteilt. Bild 3.2 gibt hierzu einen Überblick.

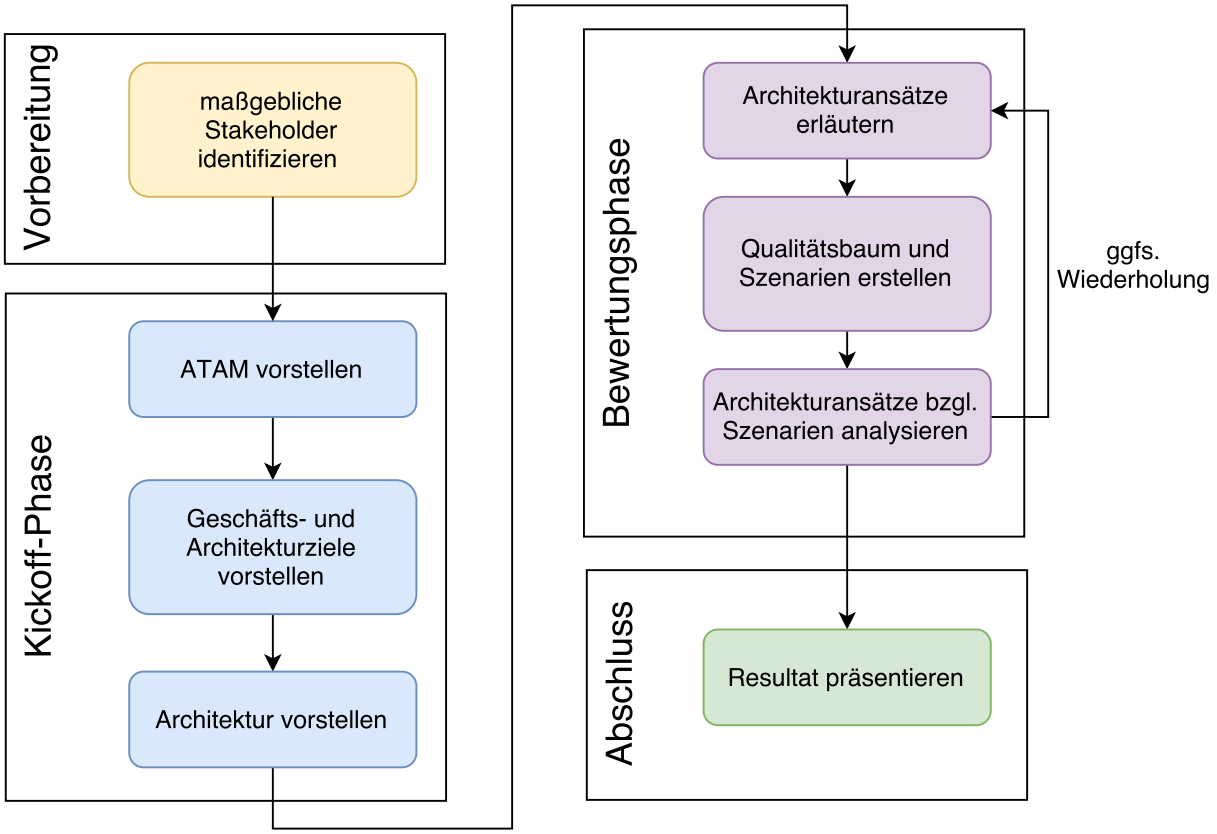

Abbildung 3.2: Phasen der Architekturbewertung nach ATAM [19]

3.3.1 Vorbereitung

Das Fundament der Bewertung stellen die Qualitätsanforderungen dar. Diese können nur lückenlos bestimmt werden, wenn alle maßgeblich vom Projekt betroffenen Personen involviert sind. In der Vorbereitungs-Phase müssen alle wichtigen Stakeholder identifiziert werden. Neben dem Kunden bzw. Auftraggeber selbst, kann dies z. B. der Benutzer, Administrator oder Tester sein. Dabei müssen jene Personen nicht direkt in den Prozess miteinbezogen werden. Es reicht schon einen Vertreter, meist aus dem Management, zu bestimmen oder die Wünsche und Ziele vorher genau zu ermitteln. In der Regel werden nur wenige Stakeholder, meist Personen aus dem Management und der Projektleitung [19], direkt zur Architekturbewertung eingeladen. Zu viele Personen würden den Prozess, durch Diskussionen und Abschweifungen, nur verlangsamen und ineffektiv gestalten.

3.3.2 Kickoff-Phase

Bewertungsmethode (ATAM) vorstellen

Im ersten Schritt soll die Bewertungsmethode vorgestellt und dessen Ziele verdeutlicht werden. Nicht jeder Stakeholder ist regelmäßig in eine Architekturbewertung involviert. So muss klar gestellt werden, welche Bedeutung das Architektur- und Qualitätsziel hat [19]. Des Weiteren sollte hervorgehoben werden, dass es bei der qualitativen Architekturbewertung um das Aufdecken von Risiken sowie um mögliche Maßnahmen geht. Es sollte nicht Ziel sein, Noten für die Architektur zu vergeben. Insbesondere sollte die Präsentation folgende Punkte enthalten [16]:

- Eine Kurze Beschreibung von ATAM und dessen Ablauf
- Methoden zur Analyse sollten erklärt werden
- Das Ziel und die Ergebnisse der Evaluation

Qualitätsziel vorstellen

Die Qualitätsanforderungen werden wesentlich vom Auftraggeber bestimmt. Aus diesem Grund sollte er auch diese vorstellen und dabei genau erläutern, was die Gründe für die Entwicklung sind und wie das System in die fachlichen Unternehmensprozesse eingeordnet werden soll. Selbst wenn die Ziele bereits in einem Anforderungsdokument erfasst wurden, ist es wichtig, die aktuelle Sichtweise des Auftraggebers zu begreifen. Zudem sind Zielformulierungen aus den Anforderungsdokumenten, meist von Systemanalytikern, gründlich gefiltert worden [19]. So können Teilnehmer Rückfragen stellen und Aha-Erlebnisse auslösen.

Architektur erläutern

In dieser Phase wird vom Softwarearchitekten die Architektur vorgestellt. Dabei wird nicht nur das eigentliche System erläutert, sondern es sollte auch der gesamte Kontext mit einbezogen werden [19]. Es beinhaltet Nachbarsysteme oder Plattformen mit denen das System in Verbindung steht. Eine angemessene Detailtiefe spielt dabei auch eine Rolle. Was angemessen ist, hängt in erster Linie von den vorhandenen Informationen (Dokumente, Diagramme, usw.) und der verfügbaren Zeit ab. Dies stellt einen wesentlichen Punkt in der Bewertung dar [16], da nur die bereitgestellten Informationen in die Analyse mit einbezogen werden können. Je mehr vorhanden ist, desto tiefer und detaillierter kann

die Bewertung durchgeführt werden. Wichtige und benötigte Dokumente oder Entscheidungen zur Architektur sollten unbedingt vor der Evaluation erstellt worden sein. Die Präsentation der Architektur sollte hier folgende Informationen umfassen [16; 19] :

- Bausteine der oberen Abstraktionsebene
- Ausgewählte Laufzeitsichten wichtiger Use-Cases
- Technische Einschränkungen, wie Betriebssystem, Hardware oder Middleware
- Weitere Systeme mit denen dieses zusammenhängt

3.3.3 Bewertungsphase

Architekturansätze identifizieren

Aus der vorangegangenen Information können nun Architekturentscheidungen identifiziert werden, die zur Erfüllung spezieller Qualitätsanforderungen dienen. Es ist zu klären, wie die Architektur, strukturell oder konzeptionell, die wesentlichen Probleme oder Herausforderungen löst. Die treibenden und prägenden Architekturansätze werden dabei von den Architekten hervorgehoben und dienen der Ergänzung des voraufgegangenem Überblicks [19]. Die Erfüllung der kritischen Anforderungen von der Architektur wird so sichergestellt.

Bildung eines Qualitätsbaumes (Quality Utility Tree)

In dieser Phase werden alle Stakeholder aktiv. Sie identifizieren und verfeinern die wichtigsten Qualitätsanforderungen des Systems. Wichtig dabei ist den Fokus nicht auf die wesentlichen Anforderungen zu verlieren und sich in Feinheiten zu verstricken [16]. Damit dies gelingen kann, wird ein Qualitätsbaum erstellt. So werden die wesentlich geforderten Anforderungen, z. B. in einem Brainstorming, erfasst und anschließend in Baum-Form angeordnet. Die globalen Anforderungen stehen auf der linken Seite und werden nach rechts hin verfeinert. Bild 3.3 stellt ein Beispiel für einen solchen Qualitätsbaum dar.

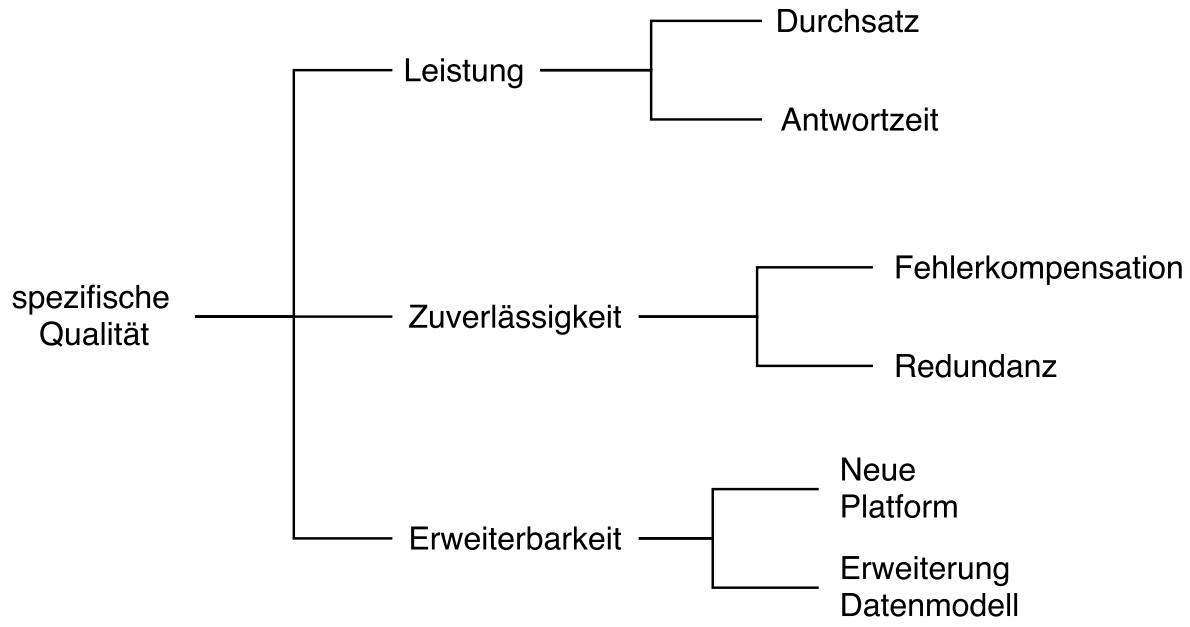

Abbildung 3.3: Beispiel eines Qualitätsbaums.

Anschließend werden zu den Qualitätsanforderungen Szenarien erstellt. Diese bilden einen wichtigen Bestandteil der Bewertungsmethode. Sie dienen dazu, die Anforderungen genauer zu definieren und den Stakeholdern somit näher zu bringen. Dabei muss die Formulierung eines Szenarios möglichst konkret sein und ein Beispiel darstellen, wie sich das System in einer bestimmten Situation verhält. Ein Szenario sollte folgende Informationen umfassen [19]:

Auslöser	Welcher spezifischer Stimulus löst das Szenario aus.
Quelle	Woher stammt der Auslöser. Z. B. intern, extern, Benutzer, Administrator usw.
Umgebung	Welcher Bestandteil des Systems ist betroffen.
Antwort	Wie reagiert das System auf den Auslöser.
Metrik	Wie kann die Antwort gemessen oder bewertet werden.

Die Szenarien sind nicht nur eine genauere Definition der Anforderungen. Auch helfen sie den Projektbeteiligten dabei, die Qualitätseigenschaften genauer zu verstehen, da sie weniger abstrakt sind und konkrete Anwendungsfälle enthalten. Ein, um Szenarien erweiterter Qualitätsbaum, ist in Bild 3.4 dargestellt.

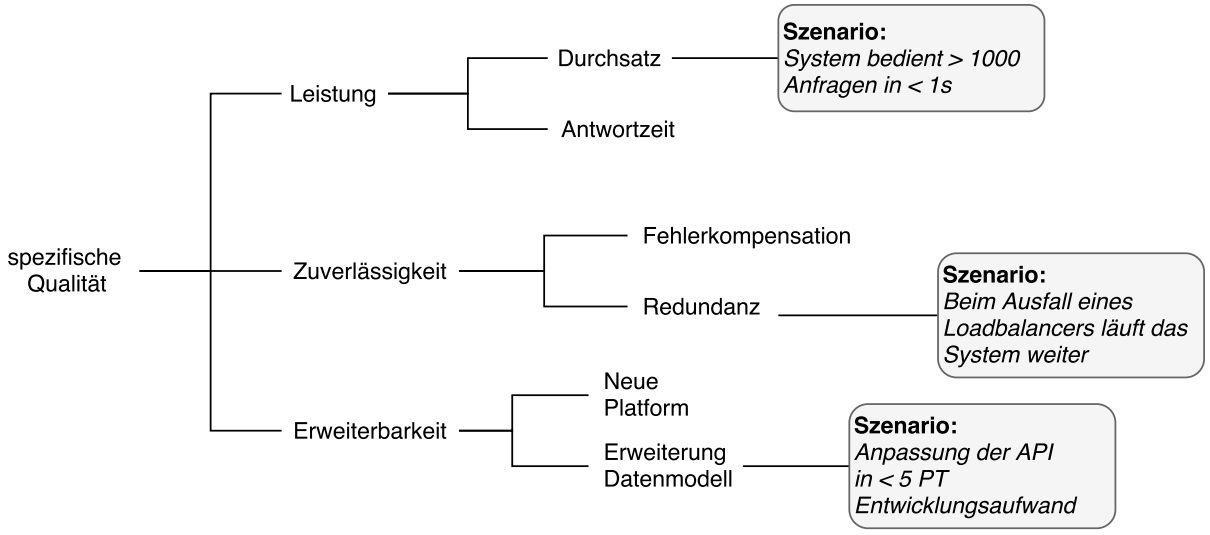

Abbildung 3.4: Qualitätsbaum mit einzelnen Szenarien erweitert

Zuletzt werden den Szenarien Prioritäten zugeordnet, so dass der Fokus zuerst auf kritische Anforderungen steht. Auch ist der Bewertungs-Workshop meist zeitlich begrenzt und somit wird sichergestellt, dass die Nichterfüllung essentieller Qualitätsanforderungen schnell zu einem Abbruch führt. Es kann durch eine einfache Skala mit wenigen Prioritäten erreicht werden, z. B. A = hoch, B = mittel und C = weniger wichtig.

Architekturansätze analysieren

Anhand der Prioritäten kann nun die eigentliche Analyse beginnen. In kleineren Gruppen, zusammen mit den Architekten oder Entwicklern, werden die Szenarien genauer untersucht und zugehörige Architekturansätze erläutert. So kann z. B. mittels eines Walkthrough genau aufgezeigt werden, wie einzelne Komponenten zur Erreichung des Ziels interagieren und welche Entwurfsentscheidungen unterstützen [19]. Dabei sollten folgende Fragen geklärt werden:

- Wie wird das Szenario in der Architektur umgesetzt?
- Warum wurde dieser Architekturansatz gewählt?
- Gibt es Kompromisse, die gemacht wurden?
- Werden andere Qualitätsmerkmale davon beeinflusst?
- Gibt es Risiken die damit verbunden sind?

- Wird die Entscheidung von Analysen, Untersuchungen oder Prototypen unterstützt?

Die Analyse ist dabei nicht auf diese Fragen beschränkt. Sie bieten lediglich einen Startpunkt für eine Diskussion, um potentielle Risiken, Sensitivitäts- oder Kompromisspunkte zu finden. An dieser Stelle gibt es leider kein Patentrezept oder ein spezifisches Vorgehen, um das Erfüllen der Qualitätsanforderung zu bestimmen. Viel mehr wird von den Teilnehmern ein analytisches und systematisches Denken verlangt. Im Fokus muss hierbei stehen, dass eine Verbindung zwischen der Architekturentscheidung und der Anforderung geschaffen wird, die sie erfüllen soll.

3.3.4 Abschlussphase

Am Ende wird das Ergebnis den Stakeholdern präsentiert. Es muss nicht zwangsläufig nur durch eine Präsentation dargestellt, sondern kann auch um einen detaillierten Berichte ergänzt werden. Sämtliche Phasen der Bewertung und dessen Ergebnisse können so noch einmal rekapituliert werden und bilden die Basis zur Definition von eventuell benötigten Maßnahmen.

Die durch ATAM gefundenen Risiken und die daraus resultierenden Maßnahmen können auch Änderungen an der Architektur darstellen. Daher bietet sich ATAM auch zum iterativen Einsatz an. Mit ersten Entwürfen einer komplexen Architektur kann überprüft werden, ob die Entscheidungen den richtigen Weg einschlagen und wichtige Anforderungen von Beginn an mit einbezogen werden. So bildet sich Schrittweise eine detaillierte Architektur unter gut dokumentierter Berücksichtigung der Qualitätsanforderungen.

Einen anderen Ansatz verfolgt die Software Architecture Evaluation Model (SAEM) Methode. Diese betrachtet die Softwarearchitektur als finales Produkt der Entwurfsphase und versucht mittels dem ISO/IEC 25010[2] Qualitätsmodell die Qualität der Architektur festzustellen.

3.4 Software Architecture Evaluation Model (SAEM)

Im Gegensatz zu ATAM ist SAEM weniger bekannt und erprobt [5]. Sie zielt darauf ab die Qualität der Softwarearchitektur selbst festzustellen. Nicht nur weil Fehler in der Architektur schwer zu beheben sind, sondern diese auch direkt einen Einfluss auf die Qualität und Kapazitäten des Systems haben [11]. Dies trifft besonders auf die nichtfunktionalen Anforderungen, wie Wartbarkeit, Skalierbarkeit oder Effizienz, zu.

[2] In der ursprünglichen Veröffentlichung von SAEM wurde das ISO/IEC 9126-1 Modell verwendet. Die ISO/IEC 9126-1 wurde 2011 durch die ISO/IEC 25010 ersetzt.

Wie beim finalen System ist die direkte Messung am Produkt eine objektive und sehr effektive Evaluation der Qualität. Um dies zu erreichen, müssen Qualitätsanforderungen definiert und ein Prozess zur Überprüfung dieser geplant, implementiert und kontrolliert werden [11]. SAEM versucht dies mittels eines Qualitätsmodells auf die Softwarearchitektur abzubilden und stellt Methoden sowie Metriken für die Evaluation bereit.

3.4.1 Spezifikation Qualität

Das Qualitätsmodell von SAEM baut dabei auf dem Modell der ISO/IEC 25010 auf. Jenes teilt die Qualität in 8 Kategorien und weitere Unterkategorien auf [10]. Anhand derer kann die Qualität der Softwarearchitektur spezifiziert werden, indem man diesen Architekturanforderungen zuordnet.

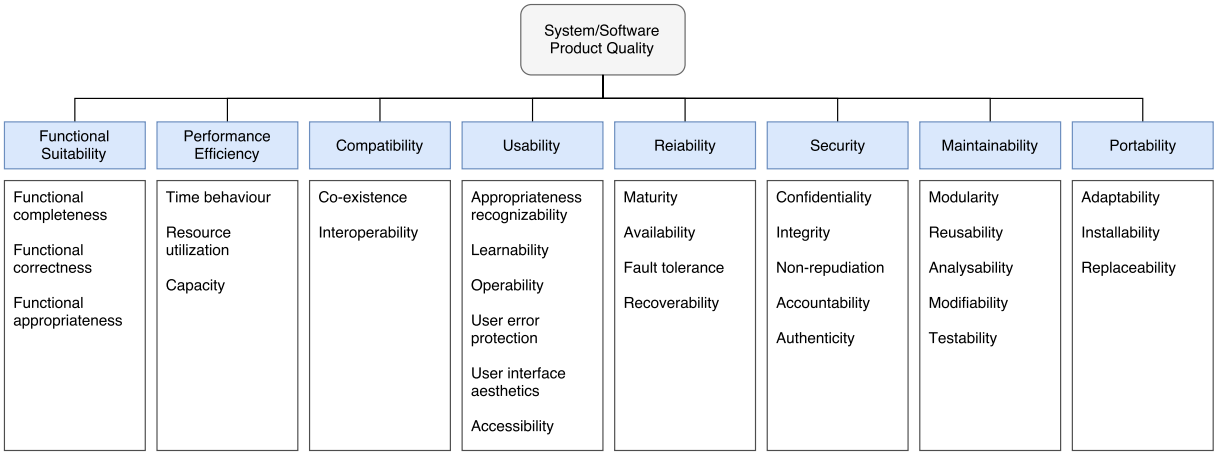

Abbildung 3.5: Qualitätsmodell der ISO/IEC 25010 [10]

Hierzu wird im ersten Schritt die externe Qualität bestimmt. Diese basiert auf messbaren Metriken unter Nutzungsbedingungen und werden vom Nutzer sowie den Architekten definiert [11]. So sollten, für alle Qualitätskategorien, Anforderungen an die Architektur ausgewählt und optimale bzw. zulässige Werte bestimmt werden. Ein Beispiel hierfür ist die Reaktionszeit auf Anfragen. Mittels der externen Qualität können die Architekten die interne Qualität bestimmen. Diese sind Anforderungen, die zur Erreichung und Zufriedenstellung des Ziels benötigt werden. Im Unterschied zu externen Anforderungen werden diese nicht durch die Nutzung bestimmt, sondern beziehen sich auf intrinsische Eigenschaften, wie z. B. Modularität oder Komplexität [11]. Die Bestimmung dieser basiert meist auf Expertenwissen oder Firmen interne Richtlinien.

3.4.2 Bestimmung von Metriken

Für die Messung der Qualitätsanforderungen müssen Metriken bestimmt werden. Hierzu kann das Bewertungsteam aus bekannten Softwaremetriken wählen oder systematisch neue definieren. Wichtig ist nur, dass die Messung zielorientiert ist. D. h. die Metriken müssen mit einem Top-Down Ansatz bestimmt werden [7]. Somit soll nicht aus vorhandenen Messungen ein Bezug zur Qualität aufgebaut, sondern mittels spezifischer Qualitätsanforderungen passende und aussagekräftige Metriken gefunden werden.

Eine sehr gute Vorlage hierzu ist die Goal Question Metrik (GQM) Methode. Diese setzt die Qualitätsanforderungen an erste Stelle, welche das Ziel darstellen. Auf dessen Basis werden Fragen abgeleitet, die das Ziel wiedergeben. Dabei muss es pro Ziel nicht nur eine Frage geben. Zur Erreichung des Ziels kann eine Komposition aus verschieden Fragen nötig sein. Dabei hilft es, das Ziel aus mehreren Sichten zu betrachten. Nach [7] sind Teilkriterien zur Erfüllung der notwendigen Fragestellung:

Sichtweise	Auftraggeber, Administrator, Entwickler usw.
Anwendungsbereich	Benötigte Technologien, angeschlossene Systeme, usw.
Zweck	Analyse, Kontrolle, Verständnis, usw.
Kontext	intern, extern

Anschließend werden für sämtliche Fragestellungen Metriken bestimmt, die diese am besten beantworten. Die damit erhobenen Daten können dabei objektiv (z. B. Unterstützung Standardformate) oder subjektiv (z. B. Skala für Zufriedenstellung) sein. Durch den Top-Down Ansatz stehen die erhobenen Daten immer im Zusammenhang mit den Zielen, womit die Interpretation leichter fällt.

Ein Beispiel für das Ergebnis von GQM wird in Bild 3.6 dargestellt.

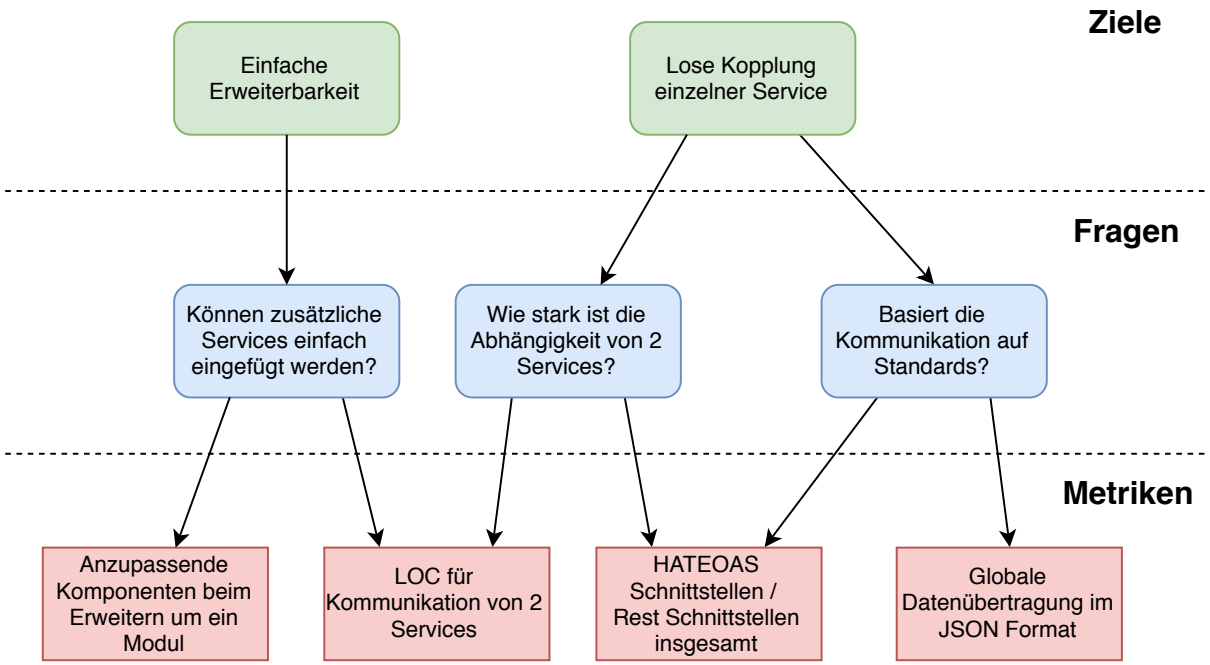

Abbildung 3.6: Beispiel für die Anwendung von GQM an einer Microservice Architektur

3.4.3 Evaluation

An diesem Punkt ist die Evaluation denkbar einfach. Sie besteht aus dem Sammeln und Auswerten der Daten mittels der Metriken. Über die gewonnenen Daten und die vorher bestimmten zulässigen Werte, wie das Vorhandensein spezifischer Funktionen, wird die Architekturqualität bestimmt. Es lässt sich auch eine Aussage über die zu erwartende Qualität des Endsystems treffen. An diesem Punkt wird der Unterschied zu ATAM wieder deutlich. SAEM versucht nicht ein tieferes Verständnis für die Architektur bei maßgeblichen Stakeholdern zu schaffen, sondern nur die Einhaltung spezifischer Anforderungen zu gewährleisten. Dies hängt dabei stark vom definiertem Qualitätsziel ab. Gerade bei der internen Qualität verlangt SAEM somit eine firmeninterne Qualitätsrichtlinie [5], die auf Erfahrung und Expertenwissen basiert.

3.5 Architekturbewertungsmethoden und Microservice Frameworks

Mit Architekturbewertungsmethoden wie ATAM oder SAEM lässt sich die Qualität einer Architektur sehr gut bewerten. So kann die spezifische Umsetzung einer Microservice Architektur auf die gegebene Anforderungen untersucht werden. Zudem schärft es nicht nur

die Anforderungen selbst, sondern kann auch die geplante Architektur weiter verfeinern und ein tieferes Verständnis bei den Stakeholdern dafür schaffen.

Mit diesem Wissen müssen anschließend die passenden Frameworks für die einzelnen Services gewählt werden. Hierzu lässt sich die reine Architekturbewertung jedoch nicht mehr verwenden, da diese ein vollständiges Verständnis über die Architektur und dessen Einfluss benötigt. D. h., in Bezug auf ein Framework muss bekannt sein, wie sich die vorhandenen Architekturentscheidungen und Entwurfsmuster innerhalb des Frameworks auf die tatsächliche Architektur der Umsetzung eines Microservices auswirken.
Beim Einsatz eines neuen Frameworks ist dieses Wissen jedoch in vielen Fällen nicht vorhanden und muss erarbeitet werden. Aus diesem Grund ist es verständlich, wenn Entwickler und Architekten bei der Wahl eines Frameworks auf ein Bekanntes zurückgreifen. Eine Bewertungsmethode sollte somit kein Expertenwissen voraussetzen, damit sich alle Frameworks in den Bewertungsprozess mit einbeziehen lassen.

Zudem kann, wie in Abbildung 3.7 gezeigt, die Bewertung eines Frameworks auf 3 Seiten aufgeteilt werden. Die Architekturbewertung betrachtet lediglich nur 1 dieser Seiten.

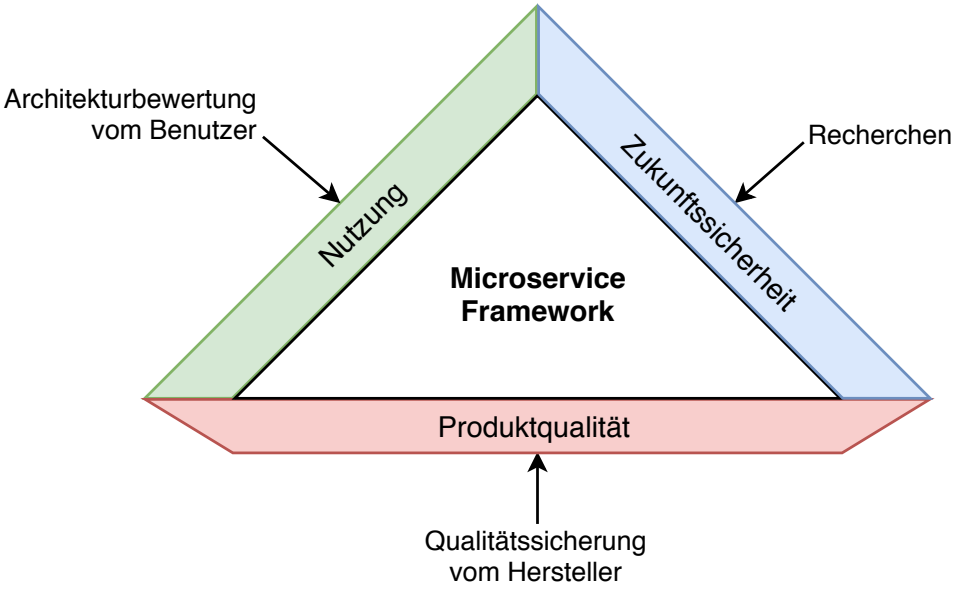

Abbildung 3.7: Die 3 Seiten eines Frameworks und dessen Bewertung

Wie ein Framework benutzt wird, stellt die erste Seite „Nutzung“ dar. Dies umfasst z. B. die Art und Weise wie Funktionen mit dem Framework umgesetzt werden, wie stark es den Nutzer dabei unterstützt oder welche Programmiersprache genutzt werden muss. Zusätzlich beinhaltet es den Architektureinfluss. Da ein Framework einen Rahmen zur Verfügung stellt, haben die Architekturentscheidungen dessen einen Einfluss auf die Architektur des

Microservices.

Des Weiteren kann die Umsetzung des Frameworks selber bewertet werden. Somit betrachtet die zweite Seite „Produktqualität“ die Qualität des Frameworks an sich, welche im Regelfall von der Qualitätssicherung des Herstellers abgesichert werden sollte. Doch hängt dies stark an den verwendeten Anforderungen. So können diese von den Anforderungen des Nutzers abweichen und somit nicht die benötigte Qualität liefern.
Die dritte Seite „Zukunftssicherheit“ betrachtet das Framework über den gesamten Software-Lebenszyklus. Die Entscheidung für ein Framework sollte auch auf lange Sicht bestand haben. So sollten z. B. Fehler vom Hersteller behoben oder neue Funktionen eingebaut werden.

Eine Framework-Bewertungsmethode sollte somit alle drei Seiten betrachten, damit diese aussagekräftige Ergebnisse liefert.

4 Microservice Framework Evaluation Method (MFEM)

Die Bewertung, ob ein Framework für den produktiven Einsatz in einer Microservice Architektur geeignet ist, lässt sich nicht auf die Servicearchitektur beschränken. Aus diesem Grund ist Microservice Framework Evaluation Method (MFEM) keine reine Architekturbewertungsmethode. Sie betrachtet einerseits den Einfluss des Frameworks auf die Servicearchitektur und versucht dabei Risiken aufzudecken sowie ein tieferes Verständnis für die Framework-Architektur zu schaffen. Des Weiteren betrachtet sie aber auch die Qualität des Frameworks selber und die daraus entstehende Umsetzung eines Microservices. Dies beinhaltet den gesamten Software-Lebenszyklus.
MFEM setzt dabei keine grundlegenden Kenntnisse über das Framework voraus, da es 2 Seiten der Analyse anbietet. Dies ist einerseits die Nutzung von bereits vorhandenem Expertenwissen, was die Phase der Evaluation verkürzt. Andererseits werden die restlichen Daten über die Erstellung von Prototypen erfasst. Auf dieser Basis kann am Ende eine begründete Entscheidung für oder gegen den Einsatz eines Frameworks getroffen werden.

Der Ablauf von MFEM ist in Bild 4.1 schematisch dargestellt und startet mit der optionalen Kickoff-Phase. Diese kann somit auch übersprungen werden, wenn die beteiligten Personen nur die Software-Architekten selbst sind. Sollten bei der Bewertung auch weitere Stakeholder mit einbezogen werden, wird zur Durchführung der Kickoff-Phase geraten.

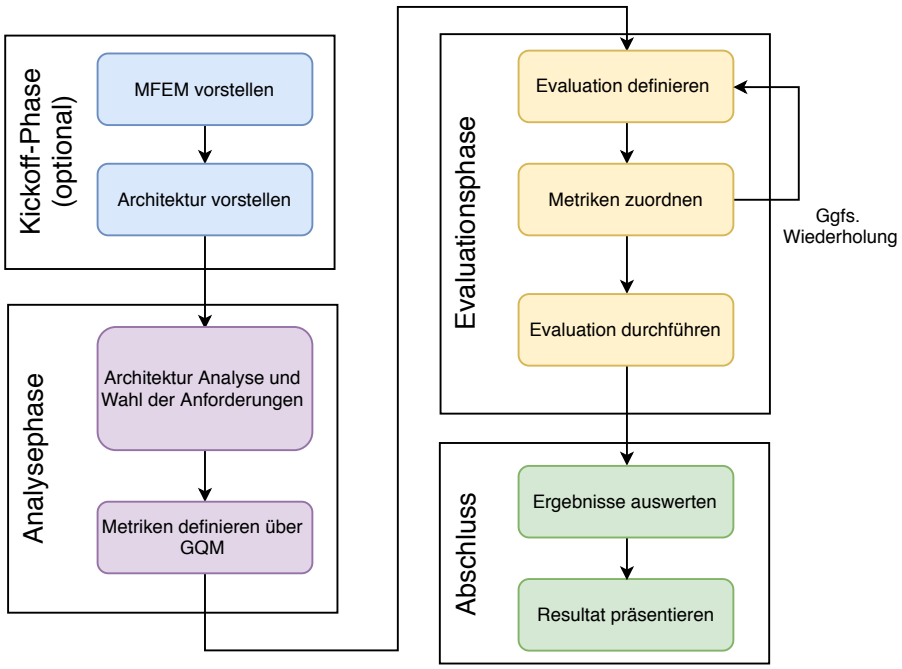

Abbildung 4.1: Schematische Darstellung vom Ablauf von MFEM

4.1 Kickoff-Phase

In der Kickoff Phase geht es darum, Klarheit zu schaffen. Den Beteiligten Personen sollten der Ablauf und die Ziele dieser Methode erläutert werden. Wie auch schon bei ATAM ist das Ziel von MFEM nicht die Vergabe von Noten. Vielmehr geht es darum den Reifegrad eines Frameworks und etwaige Risiken, wie erhöhten Entwicklungsaufwand für fehlende Funktionen, festzustellen.

In diesem Zusammenhang sollte auch die vorhandene Microservice Architektur vom Software-Architekten vorgestellt werden, damit die Grundlage der Bewertung klar ist. Neben den in der Architektur getroffenen Entscheidungen geht es auch um technische Einschränkungen und Abhängigkeiten zu Drittsystemen. Auf dieser Basis kann die Architektur analysiert und Anforderungen definiert werden.

4.2 Analysephase

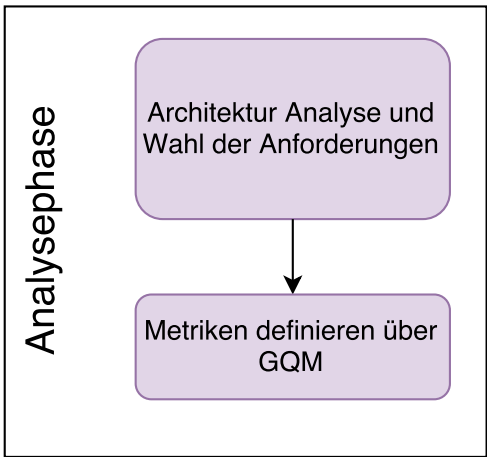

Abbildung 4.2: Analysephase von MFEM

4.2.1 Architektur Analyse und Wahl der Anforderungen

Aus dem vorangegangenen Kapitel wird ersichtlich, dass eine Bewertung nicht nur frühzeitig erfolgen muss. Viel mehr ist es wichtig, dass diese auch auf detaillierten Anforderungen basiert. Durch die maßgeblichen Stakeholder werden diese vorgegeben. In einer Microservice Architektur sind dies die definierten Umstände eines Services. Diese beschränken sich dabei nicht auf die Kommunikation zwischen zwei einzelnen Services. Gerade die Komposition der Microservices über die Infrastruktur muss gewährleistet sein. Wobei Punkte wie Sicherheit und Wartbarkeit die Komplexität noch steigern. Damit ein Service dies erfüllt und somit in der Architektur funktionieren kann, müssen anhand der Umstände, wie z. B. Servicediscovery, Logging oder Tracing, Anforderungen definiert werden. Nur wenn das Framework eines Services diese unterstützt, kann garantiert werden, dass es in die Gesamtarchitektur passt.

Funktionale Serviceanforderungen

Es müssen demnach Anforderungen gefunden werden, die die Microservice Architektur an die einzelnen Services stellt. Das ist dabei stark von der Umsetzung dieser abhängig. Ein gutes Beispiel hierfür ist die Servicediscovery. Dass sich Services gegenseitig finden können, ohne die Flexibilität der Architektur zu verlieren, darf eben diese nicht fehlen. Die Umsetzung kann dabei jedoch stark variieren. So kann ein zentraler Service, wie z. B.

Netflix-Eureka[3] oder Consul[4] , eine Registrierungsstelle anbieten. Dort können sich sämtliche Service-Instanzen registrieren und auch andere Services finden. Diese müssen es aber auch nicht direkt unterstützen. Projekte wie Spring Cloud Sidecar[5] übernehmen dies für einzelne Services. Die Discovery kann aber auch auf die darüber liegende Abstraktions-Schicht hochgezogen werden. Innerhalb eines Kubernetes[6] Clusters übernimmt die Service-Verwaltung diese Aufgabe selbst und stellt sie über Umgebungsvariablen oder DNS zur Verfügung.

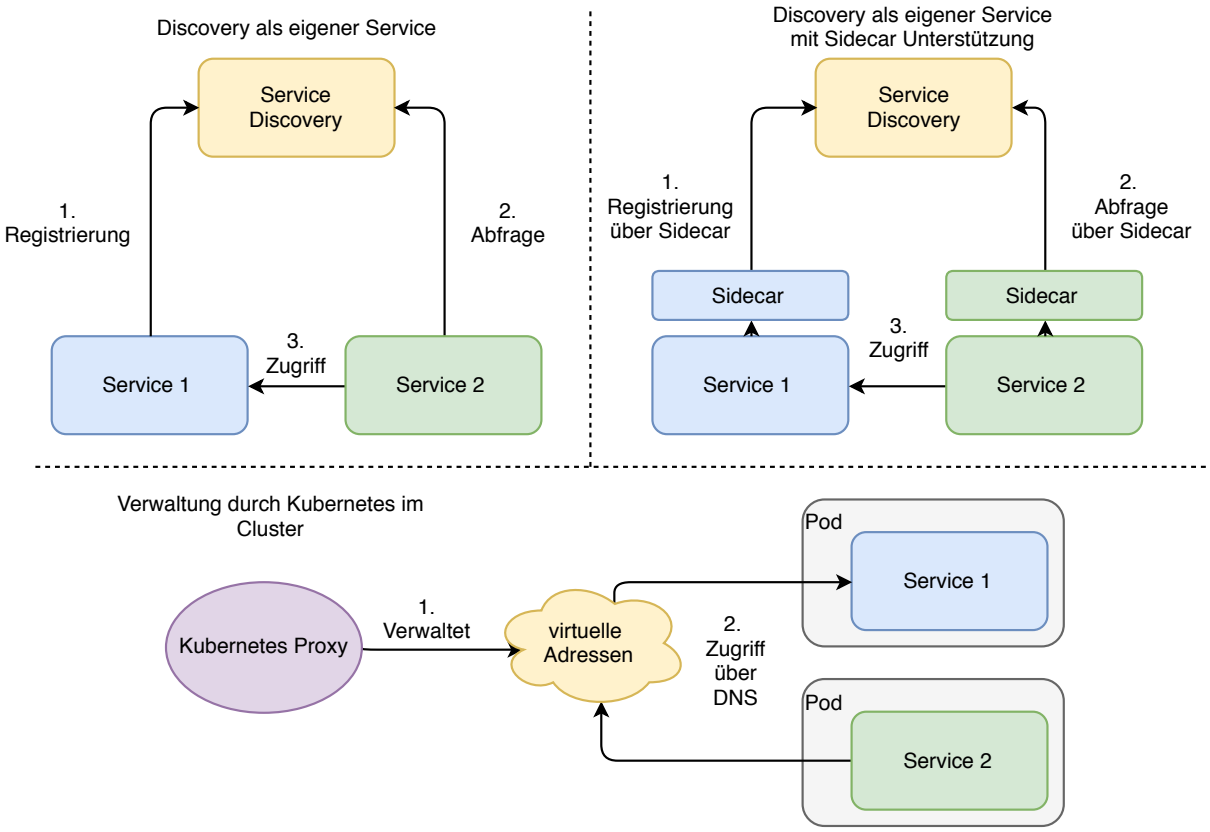

Abbildung 4.3: Beispiele für Ausprägungen der Servicediscovery

Die spezifische Ausprägung der Architektur gibt somit Anforderungen vor, die es zu identifizieren gilt. Diese stellen die funktionalen Serviceanforderungen dar und müssen für die Akzeptanz erfüllt werden. Sollte sich mit einem Framework z. B. nicht die benötigte Authentifizierung umsetzen lassen, kann der Service nicht vor Missbrauch geschützt werden.

[3] https://github.com/Netflix/eureka
[4] https://www.consul.io
[5] http://projects.spring.io/spring-cloud/spring-cloud.html#_polyglot_support_with_sidecar
[6] http://kubernetes.io

Ein Einsatz im produktiven Umfeld wäre demnach undenkbar oder nur mit unverhältnismäßig großem Aufwand zu realisieren.

Nicht-Funktionale Serviceanforderungen

Neben den funktionalen Serviceanforderungen lassen sich auch nicht-funktionale Anforderungen definieren, die zur Erreichung und Zufriedenstellung der benötigen Funktionen notwendig sind. Dies muss dabei nicht auf die durch das Framework vorgegebene Architektur des Services beschränkt sein. An dieser Stelle kann das Framework als Werkzeug aufgefasst werden. Es gilt somit nicht nur die Frage zu klären, ob ein Framework eine benötigte Funktion bereitstellt. Sondern ob es den Entwickler bei der Umsetzung bestmöglich unterstützt und dabei flexibel bleibt. Nach dem KISS-Prinzip[7] wird zum Beispiel die Funktion bevorzugt, die möglichst einfach und „dumm“ erscheint. So brauch es für die Umsetzung keine anspruchsvollen oder besonders cleveren Lösungen. Eine Einfache lässt sich nicht nur besser Lesen und Verstehen, es erhöht auch die Wartbarkeit.

Basisanforderungen von MFEM

Damit das Bewertungsteam, bei der Findung von funktionalen und nicht-funktionalen Serviceanforderungen, nicht bei null anfangen muss, wird an dieser Stelle eine Orientierungshilfe angeboten. Hier wurde in einem Brainstorming versucht, eine Schnittmenge an Anforderungen zu finden, die für die meisten Microservice Architekturen gelten sollte. Die so gefundenen Anforderungen basieren auf der Erfahrung der Anwendungsentwicklung innerhalb der Landeshauptstadt München. Diese hat sich in den letzten Jahren mit dem produktiven Einsatz von Microservices beschäftigt und diverse erfolgreiche Projekte damit umgesetzt. MFEM enthält somit Basisanforderungen, die als Grundlage für die Bewertung eines Frameworks hergenommen werden können. Eine Verpflichtung zu diesen Anforderungen besteht jedoch nicht. Wie zuvor beschrieben, hängen die Anforderungen stark am jeweiligen Kontext und müssen dahingehend beurteilt werden. Daher ist essentiell, dass die Basisanforderungen nur als Orientierung dienen und beliebig gekürzt aber in jedem Fall um weitere kontext-spezifische Anforderungen erweitert werden müssen.

Um die Analyse und Wahl der Anforderungen zu unterstützen, wird hier ein Quality Utility Tree aufgebaut.

[7] `http://principles-wiki.net/principles:keep_it_simple_stupid`

Entwicklung Quality Utility Tree

Der Quality Utility Tree (QUT) bietet einen effizienten Top-Down Ansatz zur Identifizierung und Verfeinerung von Qualitätsanforderungen. Mittels Brainstorming sollen somit Anforderungen gefundenen und in einzelne Kategorien aufgeteilt werden. Die Anforderungen müssen dabei nicht sofort konkret sein, sondern können anfangs grob erstellt werden. Über den Baum können die groben Anforderungen nach rechts weiter verfeinert werden, bis diese konkret genug sind.

Bei der Kategorisierung kann das Qualitätsmodell aus der ISO/IEC 25010 hergenommen werden. Wobei das Modell nur eine Hilfestellung ist und keine Vorgabe darstellt. Es geht lediglich darum einen Überblick zu schaffen und mehrere Teilaspekte zu betrachten. Auch ermöglichen die Kategorien eine bessere Übersicht über das Endergebnis. Hierzu später mehr im Abschnitt 4.4.1 „Ergebnisse auswerten“.

Für die Basisanforderungen wurden die Kategorien Funktionalität, Performance, Benutzbarkeit, Sicherheit, Wartbarkeit und Zukunftssicherheit definiert. Anschließend wurden diesen die Qualitätsanforderungen zugeordnet und stellen ein Beispiel dar, wie ein entsprechender QUT aufgebaut ist. Bild 4.4 zeigt einen Ausschnitt des gesamten Baumes, wobei der vollständige Baum im Anhang zu finden ist.

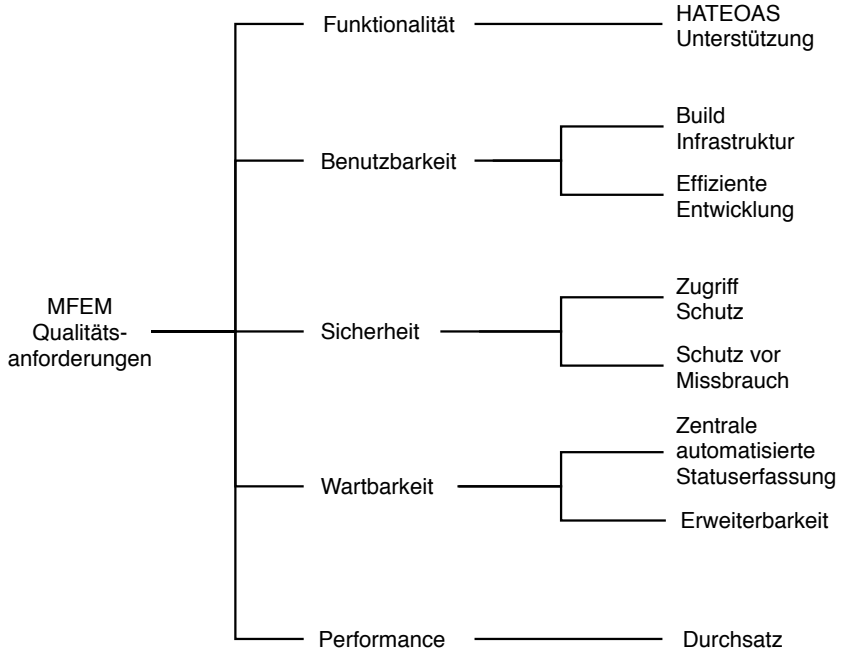

Abbildung 4.4: Ausschnitt aus dem Quality Utility Tree der Basisanforderungen

Damit der Fokus in späteren Phasen auf den wichtigen Anforderungen liegt, werden anschließend alle mit Prioritäten versehen. Hierbei bietet sich eine dreistufige Skala mit A, B und C an. Dies sollte für den Normalfall ausreichen und benötigt keiner weiteren Detailstufe. Die mit A bewerteten Anforderungen können zudem später zu einem schnelleren Abbruch führen, wenn diese nicht erfüllt werden. Auch wirken sie sich beim Gesamtergebnis stärker aus als niedriger bewertete Anforderungen.

Der in Bild 4.4 gezeigte QUT wurde im Bild 4.5 um Prioritäten erweitert. Wobei dies nur ein Beispiel ist und bei jeder Bewertung separat durchgeführt werden sollte.

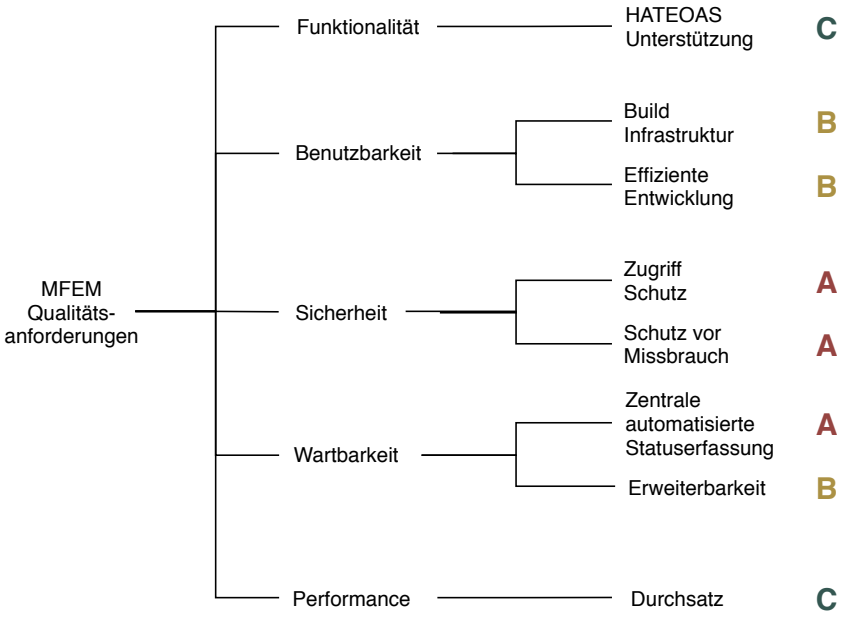

Abbildung 4.5: Beispiel eines Quality Utility Tree der Basisanforderungen mit Prioritäten

4.2.2 Metriken definieren über GQM

In diesem Schritt müssen für die identifizierten Qualitätsanforderungen möglichst aussagekräftig Metriken gefunden werden. Dies können dabei Messung an der Architektur sein, wie das Vorhandensein spezifischer Funktionen oder Komponenten sowie den Einsatz bewährter Entwurfsmuster. Dabei können die möglichen Ergebnisse komplexer als nur ein einfaches „enthalten“ oder „nicht enthalten“ sein. Falls beispielsweise eine spezifische Funktion vom Framework nicht unterstützt wird, heißt dies nicht, dass sich diese nicht umsetzen lässt. Hier bietet sich eine Ordinalskala an. Wobei folgende mögliche Ergebnisse verwendet werden können: enthalten, leicht umsetzbar, schwer umsetzbar, nicht möglich. Dieser Ansatz bietet den Vorteil, dass das Ergebnis quantifizierbar ist.

Diese Metriken bieten sich auch für weitere Anforderungen an, die sich schwer in Zahlen erheben lassen. Die Tabelle 4.1 zeigt ein paar Beispiele aus den Basisanforderungen und die zugehörigen Ordinalskalen mit einer Erläuterung zu den jeweiligen Stufen.

Anforderung (Goal)	**Frage (Question)**	**Skala für Metrik**	**Erläuterung**
Effiziente Entwicklung	Kann das Framework schnell erlernt werden?	sehr gut	Das Framework lässt sich schnell erlernen und ist intuitive zu bedienen. Es nimmt dem Nutzer, wo es geht, Arbeit ab.
		gut	Die Einarbeitungszeit ist etwas länger und das Framework ist nicht sehr intuitive.
		schlecht	Sehr lange Einarbeitungszeit und viele Eigenheiten, die sich dem Benutzer nicht direkt erschließen.
Standard Übertragungsformate	Wird JSON unterstützt?	automatisch	Das Framework de- und serialisiert Objekte automatisch ins JSON Format.
		manuell	Durch eigenes Parsing oder mittels Drittbibliotheken können die Objekte de- und serialisiert werden.
		nicht möglich	JSON wird nicht unterstützt und kann auch nicht nachträglich integriert werden.
Gute Dokumentation	Bietet das Framework eine umfangreiche Dokumentation mit Beispielen?	Sehr Umfangreich mit Beispielen	Die Dokumentation erklärt jeden Teilaspekt des Frameworks und bietet viele Code-Beispiele.
		Umfangreich	Sämtliche Funktionen sind von der Dokumentation erfasst.
		Einfach	Nur die wichtigsten Funktionen werden abgedeckt.
		nicht vorhanden	Es ist keine Dokumentation vorhanden.

Tabelle 4.1: Beispiel: Anforderungen und Ordinalskalen

Zusätzlich können auch Softwaremetriken genutzt werden. Da im Zuge der Evaluation ein Prototyp erstellt wird, können an diesem auch direkt Messungen vorgenommen werden. Damit kann neben der Performance auch z. B. der Aufwand zur Umsetzung bestimmter Funktionen bemessen werden. Falls z. B. die Erstellung eines REST-Endpunktes sehr aufwändig ist und viele Codezeilen sowie Methodenaufrufe benötigt, macht dies den Microservice schnell sehr komplex bei vielen Endpunkten und verschlechtert somit die Wartbarkeit. Um bei der Definition der Metriken stets das Ziel im Auge zu behalten, wird die bereits bekannte GQM Methode genutzt.

Eine um Metriken erweiterter Qualitätsbaum ist in Bild 4.6 dargestellt. Wie zuvor ist dies nur ein Ausschnitt, wobei der gesamte Baum im Anhang zu finden ist.

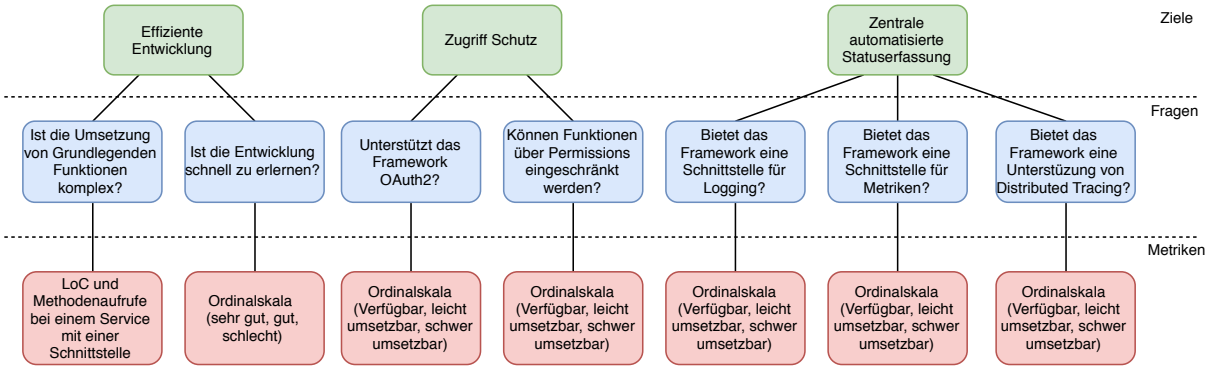

Abbildung 4.6: Ein Ausschnitt aus dem Qualitätsbaum erweitert um Metriken

4.3 Evaluationsphase

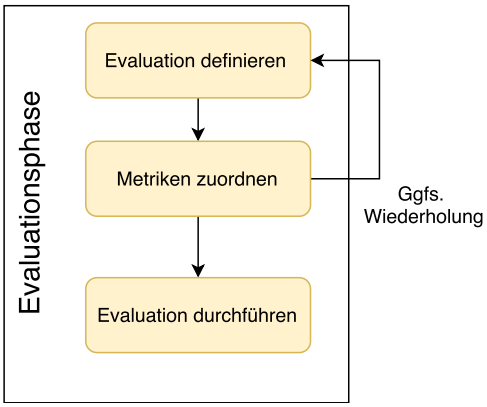

Abbildung 4.7: Evaluationsphase von MFEM

Während der Evaluation wird das Framework auf die Anforderungen mittels der zuvor definierten Metriken untersucht. Dabei wird ein tieferer Blick in das Framework geworfen. Neben der Untersuchung von Dokumentation und bereits vorhandenem Expertenwissen, wird auch ein Prototyp erstellt. Dieser soll als Referenz für zukünftig entwickelte Services gelten und muss somit zielgerichtet entwickelt werden. Es gilt in kürzester Zeit essenzielle Funktionen umzusetzen und dabei Erkenntnisse über das Framework zu gewinnen. Um dies zu Unterstützen muss im ersten Schritt die Evaluation genauer definiert werden.

4.3.1 Evaluation definieren

In der Softwareevaluation wird zwischen der subjektiven und objektiven Evaluation unterschieden [8]. Dabei versucht die subjektive Evaluation eine Beurteilung durch den Be-

nutzer zu finden, wobei der Benutzer im Fall von MFEM der Softwareentwickler ist. Während dieser Evaluation werden sogenannte „weiche“ Daten erhoben. Diese sagen aus, ob die Entwicklung mit dem Framework effizient, einfach, klar oder einsichtig ist. Es wird somit versucht eine Aussage über die Akzeptanz zu treffen. Die subjektiven Eindrücke versucht man bei der objektiven Evaluation möglichst auszuschalten. Das gelingt durch das Anwenden von „harten“ Methoden zur Erhebung von quantitativer, statisch abgesicherter Daten [8]. Sehr gute Beispiele hierfür sind Performancemessungen, Fehlerraten oder Komplexitätsbestimmungen.

Subjektive Evaluation

Während der subjektiven Evaluation soll bei MFEM der Prototyp von einem Experten erstellt werden. Wie eingangs erwähnt muss dies zielgerichtet ablaufen. Um dies zu unterstützen und gleichzeitig eine Aussage über Qualitätsmerkmale wie Lernbarkeit, Analysierbarkeit, Wartbarkeit oder Benutzbarkeit zu treffen, soll der Cognitive Walkthrough (CW) verwendet werden. Dies ist eine aufgabenorientierte Inspektionsmethode [8]. Dabei versetzt sich der Prüfer in einen hypothetischen Benutzer und analysiert konkrete Handlungsabläufe. Dies kann z. B. das Lösen eines Problems oder das Umsetzen einer einfachen Funktion sein. So kann er feststellen, woran die Entwicklung eines Microservices mit Hilfe des Frameworks vermutlich scheitern wird.

Der CW verläuft in 3 Schritten [8]:

1. Vorbereitung In diesem Schritt werden die Beispielszenarien definiert. D. h. es werden für den Experten Aufgaben bzw. Problemstellungen festgelegt, die durchgeführt oder gelöst werden sollen. Dies können grundlegend Aufgaben sein, wie das Erstellen eines REST-Endpunktes. Sie sollten so präzise wie möglich formuliert werden.

2. Analyse Während der Analyse nimmt sich der Experte ein Szenario nach dem anderen zur Hand und führt dieses durch. Für jede Aktivität wird ein Protokoll angefertigt. Durchgeführten Aktionen und damit eventuell zusammenhängenden Probleme oder Feststellungen sind damit festzuhalten.

3. Follow Up Die durch die Analyse gewonnenen Erkenntnisse werden zusammengefasst und ausgewertet. Falls nötig, können Maßnahmen bestimmt werden.

Der Vorteil des CW ist die schnelle und einfache Anwendung. Zudem kann es, wie von MFEM benötigt, in einem frühen Stadium der Entwicklung verwendet werden. [20] Im derzeitigen Definitionsschritt müssen in erster Linie Szenarien definiert werden. Die Analyse und das Follow Up erfolgen in den nächsten Schritten.

Definition von Szenarien

Für die Definition der Szenarien muss ein Standard-Anwendungsfall gefunden werden. Hierzu sollten sich die Prüfer überlegen, wie ein herkömmlicher Service, der mit dem Framework erstellt werden soll, aussehen könnte und was dieser für Eigenschaften haben sollte. Da der Einsatzzweck stark an die zuvor definierten Anforderungen gebunden ist, können diese für die Definition genutzt werden. Soll der Service z. B. eine eigene Datenbank verwalten und diese mittels REST nach außen anbieten oder wird Event Sourcing eingesetzt? Vielleicht soll der Service aber auch keine eigenen Daten verwalten, sondern diese von anderen Services einholen und zusammenführen.

Die Definition des Standard-Anwendungsfalls kann erst einmal grob erfolgen und wird im Folgenden weiter verfeinert. Hierzu sollte der gefundene Anwendungsfall in Entwicklungsstufen aufgeteilt werden. Diese stellen anschließend die Szenarien dar. Die Anzahl sollte dabei gering gehalten werden (z. B. 3 Szenarien), damit der Evaluationsaufwand nicht zu groß wird. Um die Szenarien weiter zu verfeinern und einen Rückschluss auf die Anforderungen zu gewährleisten, werden anschließend den Szenarien die zu messenden Metriken zugeordnet. Dabei kann es vorkommen, dass sich einzelne Metriken nicht zuordnen lassen, da die Szenarien anfangs nur grob definiert sind. In diesem Fall müssen die Szenarien in Hinblick auf die fehlenden Metriken weiter verfeinert werden. Der dadurch entstehende Kreislauf ist im Bild 4.8 abgebildet.

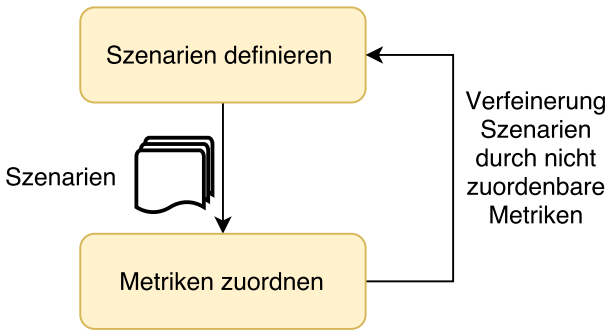

Abbildung 4.8: Kreislauf zur Definition und Verfeinerung der Szenarien

Bei der Zuordnung der Metriken muss zwischen objektiver und subjektiver Evaluation unterschieden werden. Aus diesem Grund wird dieser Vorgang erst im Abschnitt 4.3.2 genauer erläutert, nachdem auch die objektive Evaluation definiert wurde.

Beispiel-Szenarien

Im Folgenden werden 3 Beispiel-Szenarien definiert, die den Anwendungsfall eines einfachen Daten-Service darstellen sollen. Dieser hat eine Anbindung an eine Datenbank und stellt das enthaltene Datenmodell mittels REST zur Verfügung. Mit dieser groben Definition werden die 3 Szenarien gebildet.

Szenario 1 Installation: Die Grundlage für den Einsatz eines Frameworks ist die Installation der benötigten Komponenten. Da MFEM sprachunabhängig ist, kann nicht davon ausgegangen werden, dass alle beteiligten Entwickler bereits die benötigen Compiler bzw. Interpreter und zugehörige Bibliotheken installiert haben. Zudem werden in Zeiten von Continuous Integration[8] und immer kürzer werdenden Time-to-Market Zyklen automatische Build-Tools benötigt. Die Einrichtung dieser Komponenten ist somit die Basis einer jeden Entwicklung und stellt das erste Szenario dar.

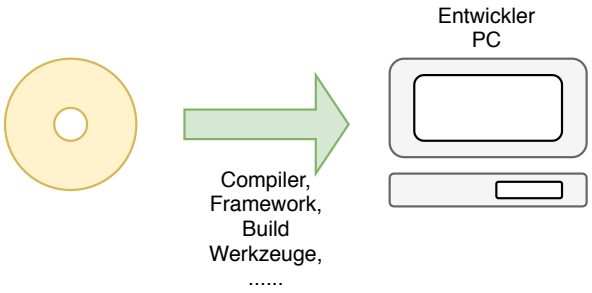

Abbildung 4.9: Szenario 1: Installation des Frameworks mit allen benötigten Komponenten.

Szenario 2 einfacher Service: Die ersten Schritte, gerade in einem ungewohnten Umfeld, sollten nicht zu groß gewählt werden, damit ein erster Eindruck zur Struktur und Syntax erfolgen kann. So bildet das zweite Szenario die Erstellung eines einfachen Hello-World-Services. Dieser soll einen Endpunkt enthalten, der auf jegliche Anfrage „HELLO World!“ antwortet. Als Ergebnis werden so erste Erkenntnisse über die Funktionsweise des Frameworks und der Erstellung von grundlegenden Elementen eines Microservices gewon-

[8] `https://de.wikipedia.org/wiki/Kontinuierliche_Integration`

nen. Da die Absicherung des Services ein wesentlicher Bestandteil ist, soll anschließend der Endpunkt mit einer Authentifizierung abgesichert werden. Die genaue Umsetzung hängt dabei von den zuvor definierten Anforderungen ab. Mit den Basis-Anforderungen von MFEM wird eine Authentifizierung mittels OAuth2-Token gefordert. Zur Vereinfachung werden JSON Web Token (JWT)[9] eingesetzt, da diese signiert sind und vom Service, ohne weitere Kommunikation mit anderen Services, selbst validiert werden können.

Abbildung 4.10: Szenario 2: Erstellung eines einfachen Hello-World-Services.

Szenario 3 erweiterter Service: Das dritte Szenario baut auf dem zweiten auf und erweitert es um ein Datenmodell. Hier soll neben der Anbindung an eine Datenbank auch die Erstellung einer komplexeren API durchgeführt werden. Das Modell muss dabei als solches nicht komplex sein. Hier wird eine einfach Personal- und Aufgabenverwaltung umgesetzt. Dieses teilt Mitarbeiter in Abteilungen auf, welche wiederum Mitarbeiter als Abteilungsleiter haben. Zusätzlich können Mitarbeitern Aufgaben zugeordnet werden. Das vollständige Datenmodell ist im Bild 4.12 dargestellt.

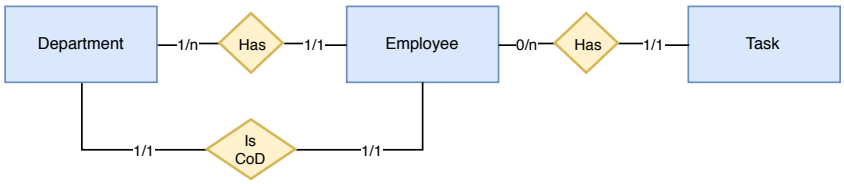

Abbildung 4.11: Szenario 3: Datenmodell der Aufgabenverwaltung

Auf diesem Modell soll auch eine einfache Geschäftslogik implementiert werden, wie die Auflistung aller offenen Aufgaben nach Abteilungen. Die tatsächliche Geschäftslogik spielt dabei weniger eine Rolle. Viel mehr soll der Umgang mit Datenmodell und eigener Logik erprobt werden.

[9] `https://jwt.io`

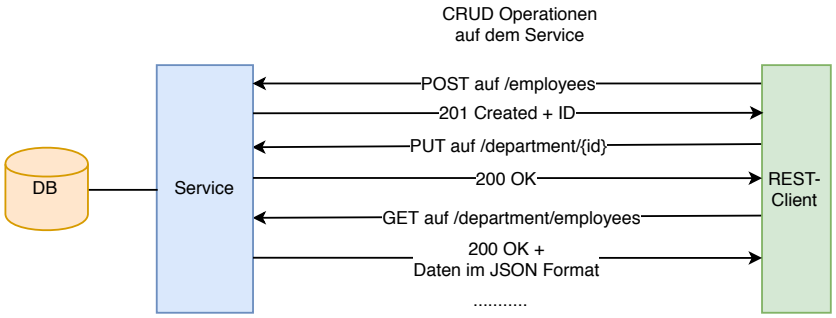

Abbildung 4.12: Szenario 3: Erstellung eines erweiterten Services.

Objektive Evaluation

Die objektive Evaluation dient zur Findung der **harten** Daten. Hierzu werden die Artefakte aus der subjektiven Evaluation als Grundlage der Messungen genutzt. So können einzelne Funktionen im Code auf Komplexität untersucht werden, damit z. B. eine Aussage über Umsetzungsaufwand oder Wartbarkeit getroffen werden kann. Neben diesen Messungen direkt an den Artefakten kann auch eine Performance Messung und Recherche zur Ermittlung der harten Daten herangezogen werden.

Performance Messung

Der aus der subjektiven Evaluation gewonnene lauffähige Prototyp kann für Performance-Messungen herangezogen werden. Hierzu sollen ein oder mehrere Anforderungsprofile definiert werden, die eine reguläre Nutzung des Services unter verschiedenen Situationen betrachtet. Damit werden die Voraussetzungen und Umstände für die Messungen genauer definiert. Die Tabelle 4.2 zeigt hierzu ein paar mögliche Beispiele.

Das erste Anforderungsprofil-Beispiel stellt einen einfachen Datenservice dar. Dieser ist mit einer Datenbank verbunden und führt diese über die REST-Schnittstelle nach außen. Durch die hohen Anfragen wird dieser Service stark unter Last gestellt, um eine Aussage über Qualitätsmerkmale wie Stabilität, Fehlerrate und Durchsatz zu treffen.

Um möglichst limitierende Faktoren, wie die Datenbank, auszuschließen, wurden die Profile zwei bis fünf definiert. Diese bauen auf einer Geschäftslogik im Service auf. Dabei spielt es keine Rolle, ob ein sinnvolles Ergebnis berechnet wird. Mit der wenig rechenintensiven Logik steht der Overhead des Frameworks im Vordergrund. Ist dieser besonders groß, um z. B. die übertragenen Daten zu de- und serialisieren, beeinflusst es jede Anfra-

Nr.	Typ	Parallele Verbindungen	Dauer	Besonderheit
1	Datenservice	255	30s	CRUD Operationen auf Datenbank
2	Einfacher Service	255	30s	Wenig rechenintensive Geschäftslogik
3	Einfacher Service	10	30s	Wenig rechenintensive Geschäftslogik
4	Komplexer Service	255	30s	Stark rechenintensive Geschäftslogik
5	Komplexer Service	10	30s	Stark rechenintensive Geschäftslogik

Tabelle 4.2: Beispiel: Anforderungsprofile für die objektiven Evaluation am Prototyp

ge am Service und vermindert den Durchsatz. Die stark rechenintensive Logik zielt dabei mehr auf einen Vergleich zu anderen Programmiersprachen ab. Aus diesem Grund sollte die enthaltene Logik einheitlich sein. So kann sie darin bestehen Testdaten zu generieren und diese zu sortieren.

Neben dem reinen Zeitverhalten können an den Profilen noch weitere Messwerte erhoben werden. Dies kann z. B. Speicherbedarf in Ruhe sowie bei Vollauslastung sein. Dabei stellen die zuvor genannten Punkte nur Beispiele für Messungen an den Profilen dar. Welche genau durchgeführt werden, wurde bereits mit den Anforderungen über GQM definiert. So bleiben die tatsächlichen Messungen sinnvoll.

Recherche

Auch durch eine Recherche können harte Daten gewonnen werden. So lassen sich Fragen beantworten, die in erster Linie nicht direkt mit der Entwicklung eines Microservices zusammenhängen. Bei einem Open-Source-Framework stellt sich z. B. die Frage, ob dieses eine aktive Entwicklergemeinschaft hat und regelmäßig Fehler behoben werden. Sollte z. B. die Behebung eines Fehlers viel Zeit in Anspruch nehmen, kann dies im produktiven Umfeld schnell zu Problemen führen. Auch ein kommerzieller Support kann gewünscht sein, wenn eigenes Fachwissen fehlt oder tiefgreifende Fehler schnell behoben werden sollen. Solche Punkte sind häufig mit ausschlaggebend bei der Entscheidung für ein Framework und sollten mit berücksichtigt werden.

4.3.2 Metriken zuordnen

Die in der Analysephase gefunden Metriken müssen den genauen Messpunkten zugeordnet werden. Wie in Abschnitt 4.3.1 definiert, wird die Evaluation in objektiv und subjektiv unterschieden. Aus diesem Grund müssen die Metriken dahingehend differenziert werden. Die Objektiven lassen sich dabei am einfachsten erkennen, da sie harte Daten erheben. Diese können am Prototyp oder durch eine Recherche gewonnen werden. Die Subjektiven basieren auf der Einschätzung des Prüfers und sollen in den Szenarien der subjektiven Evaluation ermittelt werden. Bild 4.13 zeigt diese Aufteilung.

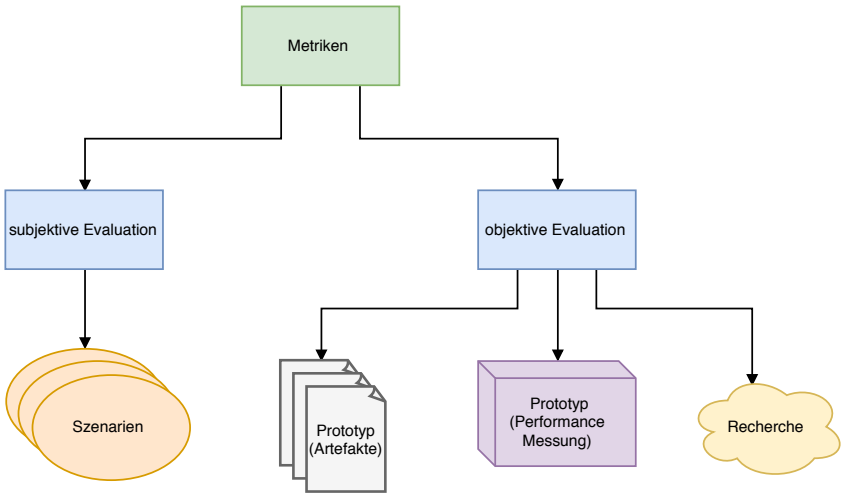

Abbildung 4.13: Aufteilung der Metriken in subjektiv und objektiv sowie Zuordnung zu einzelnen Messpunkten

Wie bereits in Abschnitt 4.3.1 erwähnt, kann es vorkommen, dass sich einzelne subjektive Metriken nicht zuordnen lassen. In diesem Fall müssen die Szenarien angepasst und verfeinert werden.

4.3.3 Evaluation durchführen

An dieser Stelle werden die zuvor definierten Evaluationsprozesse durchgeführt. Dabei sollte der Fokus auf die Anforderungen mit hoher Priorität stehen. So kann das frühzeitig nicht Erfüllen von wichtigen Anforderungen direkt zum Abbruch führen.

Zu aller erst werden die Szenarien des CW umgesetzt. Dabei müssen die gewonnenen Erkenntnisse negativ als auch positiv protokolliert werden, damit diese sich später besser auswerten lassen. Sobald ein Szenario durchgeführt wurde, wird es durch die zugeordneten Metriken bewertet.

Nachdem das letzte Szenario de CW durchgeführt wurde, steht ein Prototyp für die objek-

tive Evaluation bereit. An diesem werden nun über den Programmcode die Softwaremetriken gemessen. Zusätzlich wird am laufenden Prototyp mittels Tools, wie z. B. JMeter[10], die Leistung sowie der Durchsatz gemessen.

4.4 Abschlussphase

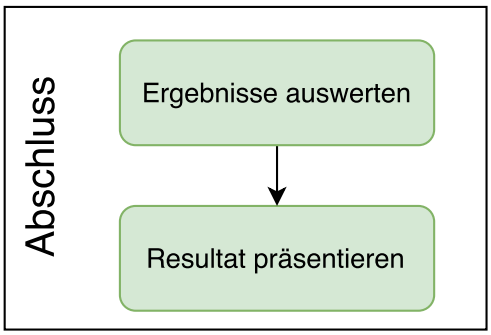

Abbildung 4.14: Abschlussphase von MFEM

In der Abschlussphase geht es um die Aufbereitung und Präsentation der Daten. Die zuvor durchgeführten Phasen bringen viele Anforderungen und zugehörige Metriken hervor. Da dies einen schlechten Überblick über das Gesamtergebnis liefert, werden im nächsten Schritt die Daten aufbereitet.

4.4.1 Ergebnisse auswerten

Da Entscheidungsträger häufig keine Zeit für Detailanalysen haben, müssen die Daten so aufbereitet werden, dass mit einem Blick ein Gesamtergebnis sichtbar wird. Detailfragen lassen sich weiterhin durch die Anforderungen und Ergebnisse der Metriken klären. Ein sehr guten Überblick lässt sich durch ein Netzdiagramm erreichen. Hierzu können die Qualitätskategorien aus dem im Abschnitt 4.2.1 erstellten Quality Utility Tree für die Achsen genommen werden. Um einen Eindruck für das Ergebnis der Auswertung zu erhalten, ist ein Beispiel in Bild 4.15 dargestellt.

[10] http://jmeter.apache.org

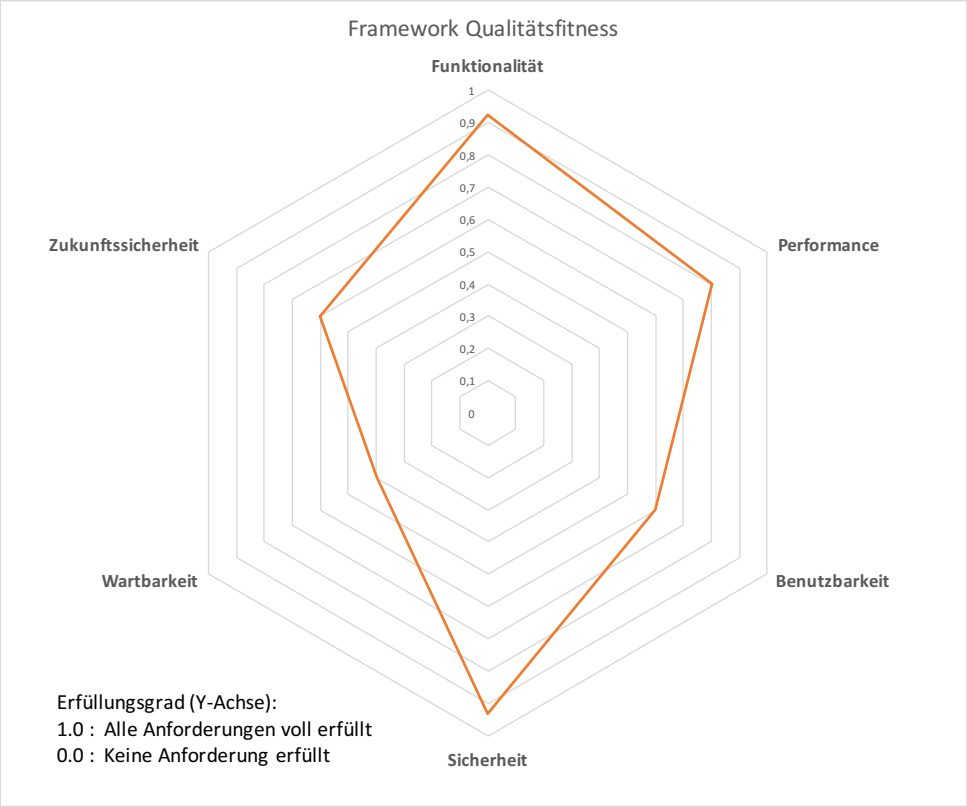

Abbildung 4.15: Beispiel für aufbereitet Daten als Netzdiagramm mit Qualitätskategorien als Achsen.

Die Achsen der einzelnen Qualitätskategorien stellen dabei den Gesamt-Erfüllungsgrad in Prozent dar. Wurden alle Anforderungen der Kategorie mit vollster Zufriedenheit erfüllt, ergibt sich ein Vollausschlag auf der jeweiligen Achse. Die Nichterfüllung einzelner Anforderungen mindert den Ausschlag entsprechend.

Wie stark einzelne Anforderungen in die zugehörige Kategorie einfließen, hängt von der Priorisierung dieser ab. Wurde eine Anforderung mit A bewertet, zählt das Ergebnis zu 100 Prozent. Entsprechend wird der Einfluss bei Priorität B und C auf 50 bzw. 25 Prozent gesenkt. Dies stellt sicher, dass die Nichterfüllung kleiner Anforderungen das Gesamtergebnis nicht zu stark nach unten ziehen.

Für eine einfache Auswertung reicht die Unterscheidung zwischen Erfüllen (1) und Nichterfüllen (0) der Anforderungen. Somit reicht es die Gewichte der erfüllten Anforderungen pro Kategorie zu kumulieren, den Prozentsatz zu bilden und in die jeweilige Achse einzutragen. Dies hat jedoch den Nachteil, dass gerade bei den qualitativ erhobenen Daten viel Informationsgehalt verloren geht. Wird z. B. die Lernbarkeit über einer Ordinalskala mit 3 möglichen Werten erhoben (sehr gut, gut, schlecht), geht die Differenzierung zwischen

„sehr gut“ und „gut“ bzw. „gut“ und „schlecht“ verloren, je nachdem wo die Akzeptanzschwelle liegt.

Um dies zu verhindern, kann auch eine komplexere Auswertung erfolgen. Diese betrachtet jede erhobene Metrik und stellt einen Erfüllungsgrad pro Anforderung zusammen. Hierzu wird das Ergebnis der Metrik auf einen Wert zwischen 0 und 1 normalisiert. Dies kann bei der zuvor genannten Ordinalskala mit 3 Werten folgende Ergebnisse enthalten:

Ergebnisse	**Normalisierung**
sehr gut	1
gut	0.5
schlecht	0

Tabelle 4.3: Beispiel: Normalisierung einer Ordinalskala mit 3 Werten.

Für quantitativ erhobene Daten kann weiterhin die dyadische Darstellung gewählt werden. Alternativ hierzu kann auch eine Abstufung mittels prozentualer Abweichung zum Zielwert ermittelt werden. Soll z. B. die Antwortzeit eines ausgelasteten Services unter 10ms im Durchschnitt betragen, könnte folgende Normalisierung gewählt werden:

Ergebnisse	**Normalisierung**
$\leq 10ms$	1
$12ms$	0.8
$14ms$	0.6
$16ms$	0.4
$18ms$	0.2
$\geq 20ms$	0

Tabelle 4.4: Beispiel: Normalisierung von quantitativ erhobener Latenzzeit

Wie die erhobenen Daten normalisiert werden, hängt von den Anforderungen ab und welches Ziel diese verfolgen. Aus diesem Grund ist dies eine Einzelfallentscheidung und obliegt dem Prüfer.

Für den Erfüllungsgrad einer Anforderung können nun die einzelnen normalisierten Ergebnisse kumuliert und der Prozentsatz gebildet werden. Anschließend erfolgt die Multiplikation mit den Gewichten und das Eintragen in die jeweilige Achse analog zur einfachen Darstellung.

Ein einfaches Beispiel für die Auswertung einer Kategorie findet sich in Bild 4.16

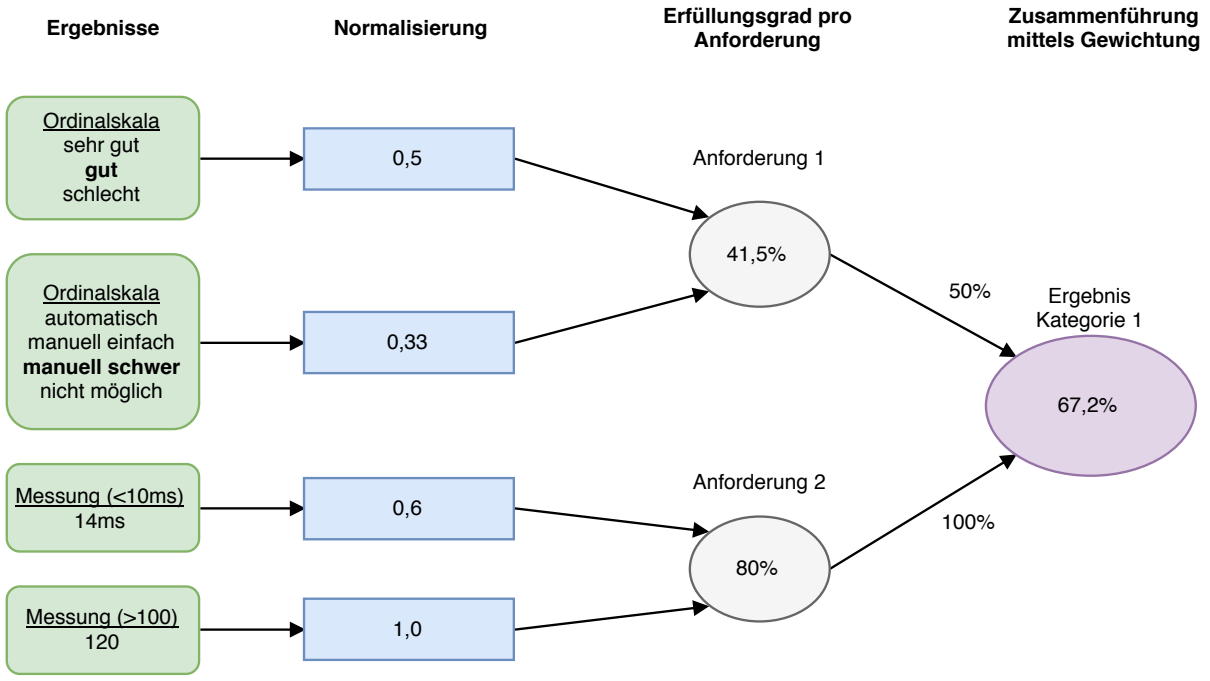

Abbildung 4.16: Beispiel für die Auswertung einer Kategorie.

Die so ausgewerteten und in ein Netzdiagramm eingetragenen Daten bieten einen guten Überblick über die Qualität eines Frameworks. Sie lassen aber auch einen Vergleich mehrerer Frameworks zu, indem verschiedene Ergebnisse in das Diagramm eingetragen werden.

4.4.2 Resultat präsentieren

Im letzten Schritt werden die Ergebnisse festgehalten und möglichen Stakeholdern präsentiert. Dies muss dabei nicht zwangsläufig eine Präsentation sein. Viel mehr geht es darum, aus dem Gesamtergebnis ein Resultat zu ziehen. Dies kann in einem abschließendem Dokument erfolgen. Dabei lassen sich wichtige Anforderungen, die nicht erfüllt wurden hervorheben. Sollten z. B. Funktionen fehlen, kann bei einem späteren Release des Frameworks die Bewertung verkürzt wiederholt und nur auf diese spezifischen Anforderungen untersucht werden. Auch kann anhand des zuvor gewonnen Gesamtüberblicks ein Vergleich mit bereits bewerteten Frameworks erfolgen. Dies kann anschließend die Grundlage für oder gegen die weitere Entwicklung mit dem Framework sein.

5 Evaluation der Methode an Beispielen

In diesem Kapitel soll die Anwendbarkeit und Wirksamkeit von MFEM untersucht werden. Hierzu wird die Methode an 2 Beispielen durchgeführt und anschließend ausgewertet. Es werden sämtliche Phase exemplarisch durchlaufen, dokumentiert und kurz resümiert. Zudem wird für die Durchführung der 2 Beispiele der Zeitaufwand gemessen, um auch eine Aussage über die Anwendungsdauer zu treffen. Diese spielt eine Rolle für die Akzeptanz der Methode, da eine lange Laufzeit dem Nutzen entgegensteht.
Als Ergebnis der Evaluation wird sich zeigen, ob die Methode möglichst viele Seiten eines Frameworks betrachtet und dabei repräsentative Resultate liefert. Um dies zu gewährleisten, müssen die richtigen Kandidaten gewählt werden.

5.1 Wahl der Kandidaten

Um mit der Evaluation von MFEM möglichst vertretbare Ergebnisse zu erhalten, wurden folgenden Anforderungen an die Wahl der Kandidaten gestellt:

Heterogenität Da MFEM eine sprachunabhängige Bewertungsmethode ist, sollten die Programmiersprachen andersartig sein.

Diskrepanz Um Unterschiede in einzelnen Aspekten zu erhalten, sollte der Reifegrad bzw. Entwicklungsstand auseinandergehen.

Popularität Die Frameworks sollten aktuell und relevant sein.

Opportunität Der Zweck sollte nachgewiesenermaßen für den Einsatz in einer Microservice Architektur sein.

Diesen Anforderungen folgend wurde als Erstes das auf Java basierende Spring Framework[11] gewählt. Es befindet sich seit 2002 in Entwicklung und hat seit dem mehrere Preise gewonnen [6, S. 1]. Dabei entwickelt es die sehr aktive Open-Source-Community ständig weiter und integriert die neuesten Technologien. Beispiele hierfür sind WebFlux, welches reaktive Webapplikationen ermöglicht, oder Cloud-Contract, um die REST-Schnittstelle mit einem Consumer-Driven Contract abzusichern. Zudem findet jährlich die SpringOne Konferenz statt, welche mit über 2000 Teilnehmern und 30 großen Sponsoren für den Erfolg von Spring sprechen [14].
2014 wurde die Erweiterung Spring Boot offiziell veröffentlicht und stellte eine große Evolution des Frameworks dar. Es nimmt dem Entwickler so viel Arbeit wie möglich ab, sodass dieser sich auf die Geschäftslogik konzentrieren kann. Dabei stellt es die Konvention über die Konfiguration und erstellt so automatisch robuste, erweiterbare und skalierbare Spring

[11] `spring.io`

Applikationen [6, S. 1]. Dies spricht für den hohen Reifegrad und die Akzeptanz des Frameworks.
Nach [21] ist das Spring Framework eine sehr gute Wahl für den Einsatz in einer Microservice Architektur und lässt dabei kaum ein Problem offen, für das es keine Lösung vorhält.

Die Wahl des zweiten Frameworks fiel auf Go-Kit[12]. Es wird seit 2016 entwickelt und ist somit, im Gegensatz zu Spring, ein sehr junges Framework. Dabei setzen die Entwickler auf die Programmiersprache Go, welche selbst noch relativ jung ist.
Go wurde ursprünglich 2007 von Mitarbeitern des Unternehmens Google [12] entwickelt und ist seit 2012 in einer stabilen Version verfügbar. Seit dem ist das Interesse für diese Sprache immens gestiegen, was der Tiobe Index deutlich zeigt. Dies ist ein Popularitäts-Ranking von Programmiersprachen, in dem Go bereits Platz 14 (Stand: Februar 2017) einnimmt [3]. Aufgrund der starken Steigerung gegenüber dem Vorjahr wurde Go von Tiobe auch zur Programmiersprache 2016 gewählt.
Da Go bereits im Hinblick auf skalierbare Netzwerkdienste, Cluster- und Cloud Computing entwickelt wurde, bringt die Sprache bereits sehr viel für die Erstellung von verteilten Systemen mit. In Bezug auf eine Microservice Architektur fehlen jedoch einige Funktionen wie z. B. Infrastrukturintegration oder Logging. Diese Lücke versucht Go-Kit zu schließen, damit der Entwickler sich auf die Geschäftslogik konzentrieren kann.

Mit Spring und Go-Kit für die Evaluation von MFEM werden die Anforderungen nach Heterogenität, Diskrepanz, Popularität und Opportunität erfüllt. Neben den unterschiedlichen Programmiersprachen geht der Reifegrad der Frameworks stark auseinander und sollte somit für unterschiedliche Ergebnisse sorgen. Nichtsdestotrotz erfahren beide aktuell einen hohen Zuspruch für den Einsatz in einer Microservice Architektur und stellen somit interessante Kandidaten dar.

5.2 Kickoff-Phase

Neben der Einführung der Methode sieht diese Phase eine Vorstellung der vorhandenen Microservice Architektur vor. Da ein Teil der Evaluation auf den in MFEM enthaltenen Basisanforderungen und deren Wirkung liegen soll, wurde ein möglichst einfacher Entwurf der zugrundeliegenden Architektur gewählt. So ergeben sich in der Analysephase nur wenige spezifische Anforderungen, die für die Evaluation der Methode nur eine geringe Rolle spielen. Weitere Phasen sind von der Microservice Architektur nicht direkt betroffen und bleiben daher unberührt. Abbildung 5.1 zeigt die gewählte Architektur.

[12] `gokit.io`

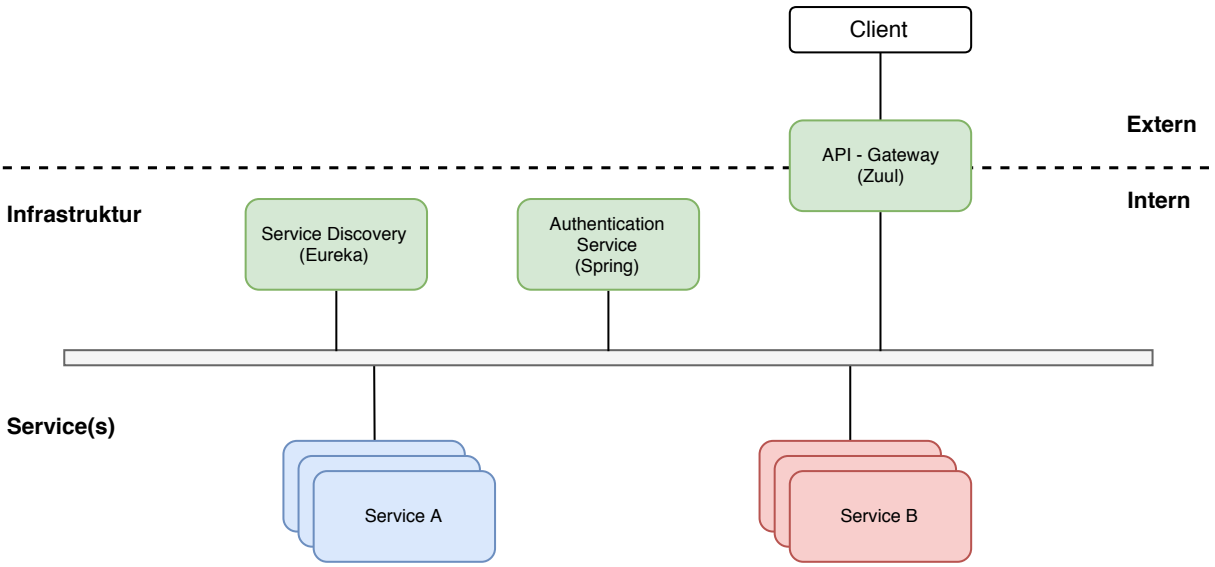

Abbildung 5.1: Vorgegebene Microservice Architektur für die Evaluation

Die Architektur teilt sich in Infrastruktur und Services auf. Services enthalten die Geschäftslogik sowie Datenhaltung. Dabei kann es verschiedene Services geben, die in mehrere Instanzen vorhanden sein können. Die genaue Form ist für die Evaluation irrelevant und wird daher nicht weiter berücksichtigt. Des Weiteren kümmert sich die Infrastruktur um den reibungslosen Ablauf und ist in Service-Discovery, Authentication-Service und API-Gateway aufgeteilt.

Das API-Gateway stellt eine einheitliche Schnittstelle des Gesamtsystems den Clients zur Verfügung. So müssen diese keine Kenntnis über die interne Struktur haben und können sämtliche Funktionen über einen zentralen Punkt nutzen. Hierzu reicht es die externen Anfragen an die jeweiligen Services weiter.

Die Service-Discovery sorgt für die Verwaltung der einzelnen Services und deren Instanzen, damit diese von anderen, z. B. dem API-Gateway, erreicht werden können. Jede Instanz eines Services muss sich an der Discovery beim Start registrieren und einen regelmäßigen Heartbeat an diese senden. Hierzu bietet die mit Netflix Eureka erstellte Discovery eine REST-Schnittstelle zur Verfügung. Der genaue Ablauf ist in Abbildung 5.2 dargestellt.

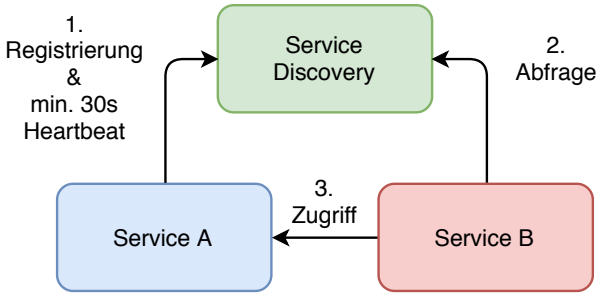

Abbildung 5.2: Ablauf: Zugriff über die Service Discovery

Der Authentication Service kümmert sich um die Authentifizierung der Benutzer. Damit müssen die einzelnen Services keine eigene Benutzerverwaltung integrieren. Mit der Authentifizierung erhält der Client einen begrenzt gültigen JWT, mit dem er auf die Services zugreifen kann. So lassen sich auch kaskadierte Anfragen realisieren, bei dem ein Service mittels des JWT einen anderen aufruft ohne die Anmeldeinformationen des Clients zu kennen. Dies ist auch vereinfacht in Abbildung 5.3 dargestellt, wobei die Zwischenschritte über das API-Gateway und die Service-Discovery vernachlässigt wurden.

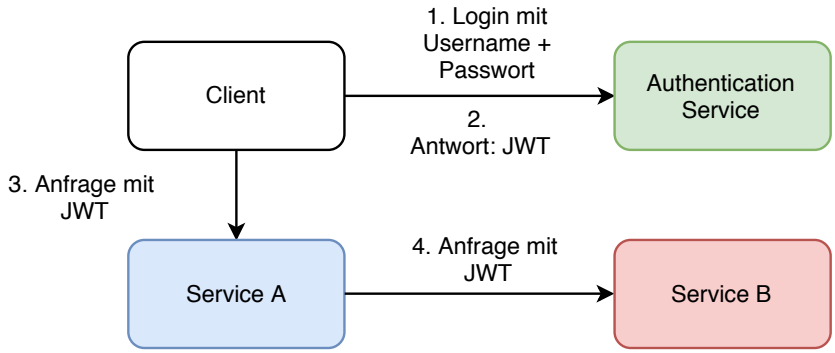

Abbildung 5.3: Vereinfachte Darstellung einer kaskadierten Anfrage eines Clients über mehrere Services

Weitere Vorgaben über die Architektur werden nicht gemacht, da diese im Sinne der Evaluation nicht zielführend wären. Wie eingangs geschildert, würde sich dies nur in zusätzlichen Anforderungen widerspiegeln.

5.2.1 Resümee Kickoff

Da nur der Autor die Evaluation durchführt, gab es keine Informationen, die mit anderen Teilnehmern ausgetauscht werden mussten. Aus diesem Grund wäre diese Phase bei einer

realen Anwendung übersprungen worden. Sie ist somit zu Recht nach MFEM nur optional durchzuführen. Eine Einführung bei mehreren Teilnehmern mach weiterhin durchaus Sinn.

5.3 Analysephase

In der Analysephase sollen nun mit Hilfe der gegebenen Architektur die Anforderungen gewählt, priorisiert und um Metriken erweitert werden. Hierzu wird im nächsten Schritt ein Quality Utility Tree aufgebaut.

5.3.1 Architektur Analyse und Wahl der Anforderungen

Mit MFEM kommen bereits viele Anforderungen, die Basisanforderungen, mit. Da die Architektur möglichst einfach gehalten wurde, gibt es keinen Widerspruch zu diesen Anforderungen. Auch gibt es unter den Basisanforderungen keine Punkte die sich Widersprechen. Sie lassen sich somit vollständig übernehmen.

Zusätzlich ergeben sich aus der Architektur funktionale Serviceanforderungen. Dies betrifft hauptsächlich die Service-Discovery. Ein Service muss sich an diesem An- und Abmelden sowie regelmäßig einen Heartbeat senden. Auch für die Inter-Service-Kommunikation muss ein Service an der Discovery die Einträge abfragen und verarbeiten können. (Abbildung 5.2) Somit wurde die Anforderung zur Unterstützung einer mit Eureka aufgebauten Service-Discovery aufgenommen.

In den Basisanforderungen ist bereits unter der Kategorie Sicherheit die Anforderung nach einer Absicherung gefordert. Dies beinhaltet die Unterstützung eines JWT und wird daher nicht als eigener Punkt aufgenommen. Zusätzlich soll auf eine Autorisierung durch Benutzer zugehörige Rechte verzichtet werden. Aus diesem Grund ergeben sich durch den Authentifizierungs-Service keine weiteren Anforderungen.
Auch durch das Gateway ergeben sich keine Anforderungen, da die Beziehung zu den Services einseitig ist. Im vorliegenden Entwurf leitet das API-Gateway Anfragen an die Services weiter. Jedoch stellen die Services keine Anfragen an das Gateway und müssen daher keine zugehörigen Funktionen unterstützen.

Die Basisanforderungen decken bereits eine Vielzahl an nicht-funktionalen Serviceanforderungen ab. Diese beinhalten z. B. die Forderung nach einer aktiven Community, eine effiziente Programmierung oder eine gute Dokumentation. Aus diesem Grund wird auch hier auf die Aufnahme weiterer Anforderungen verzichtet.

Nachdem alle Anforderungen zusammengetragen und in einem Quality Utility Tree eingeordnet wurden, sind diese Priorisiert worden. Abbildung 5.4 zeigt den entstandenen Quality Utility Tree mit allen Anforderungen für die Evaluation.

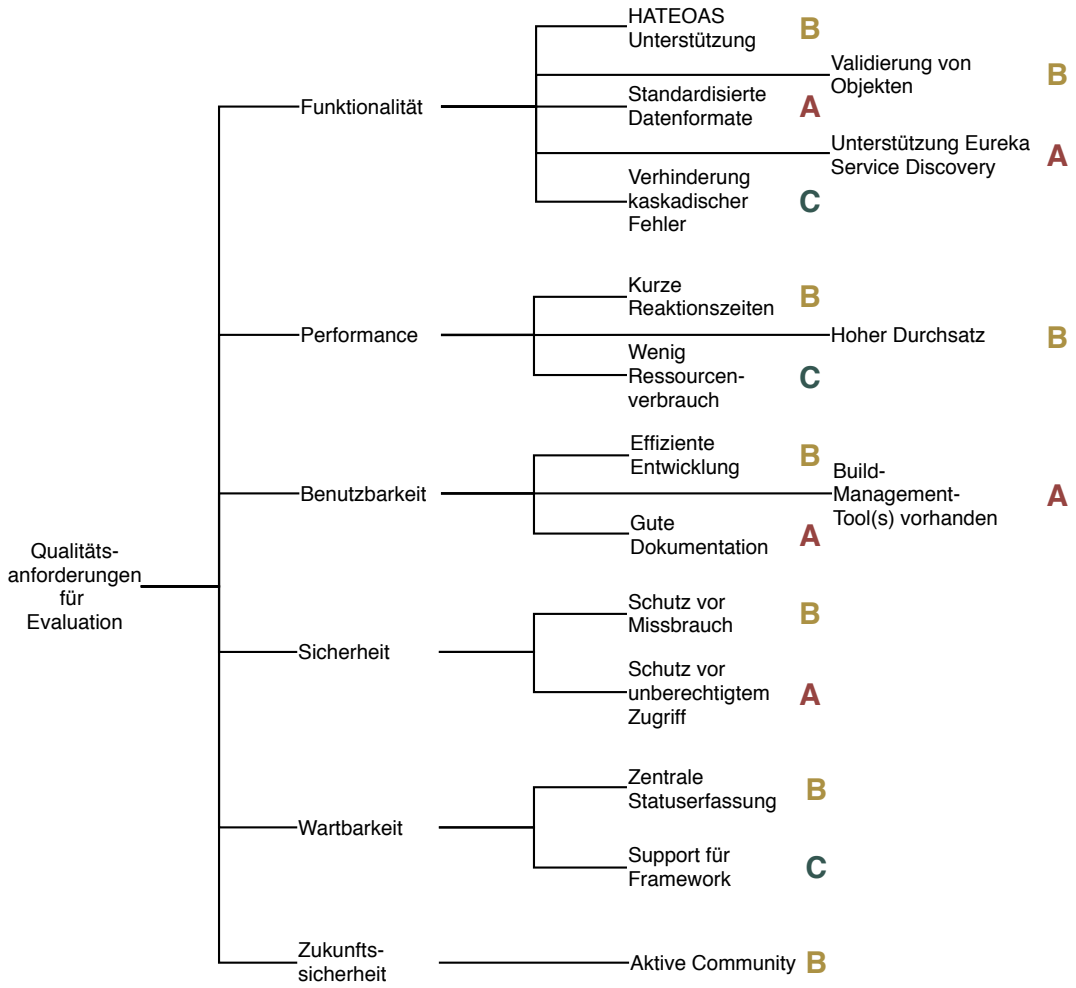

Abbildung 5.4: Priorisierter Quality Utility Tree mit allen Anforderungen für die Evaluation.

5.3.2 Metriken definieren

Mit dem Ergebnis aus dem letzten Schritt können nun die passenden Metriken für die Anforderungen gefunden werden. MFEM sieht hierfür den Top-Down Ansatz der GQM Methode vor.

Das heißt, dass für jede Anforderungen eine oder mehrere Fragen erstellt werden, die zusammen das Ziel wiedergeben. Sie drücken somit aus, was man mit der Messung erfahren will und betrachten die Anforderung von mehreren Seiten.

Für alle Anforderungen wurden so Fragen erstellt und in einer Tabelle festgehalten. Darauf aufbauend wurden Metriken definiert, die die Fragen beantworten sollen.

Die gesamte in diesem Schritt erstellte Tabelle wird in 5.1 dargestellt und enthält das

Ergebnis der Anwendung von GQM auf die Anforderungen mit allen definierten Metriken. 1

Kategorie	Anf. Nr.	Anforderung (Goal)	Priorität	Question	M.-Nr.	Metrik
Funktionalität	1	Eingehende Domänen Objekte müssen validiert werden	B	Unterstützt das Framework eine automatische Validierung von Domänen Objekten?	1	Objekvalidierung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
	2	Die Schnittstelle ist nach dem HATEOAS Modell aufgebaut	B	Fügt das Framework HATEOAS Links in die Antowrt ein?	2	HATEOAS-Links: Ordinalskala (automatisch, manuell, nicht möglich)
				Baut das Framework die Schnittstelle über Ressourcen auf?	3	Schnittstelle nach Ressourcen: Ordinalskala (automatisch, manuell, nicht möglich)
	3	Standardisierte Datenformate	A	Kann das Framework Daten im JSON Format de-/serialisieren	4	JSON-Format: Ordinalskala(automatisch, manuell, nicht möglich)
				Kann das Framework Daten im XML Format de-/serialisieren	5	XML-Format: Ordinalskala(automatisch, manuell, nicht möglich)
	16	Unterstützung Eureka Service Discovery	A	Kann das Framework den Service automatisch an der Discovery An- und Abmelden sowie ein Heartbeat senden?	27	Discovery An- und Abmeldung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
				Kann das Framework automatisch Instanzen eines bestimmten Services an der Discovery abfragen und für die Anfrage nutzen?	28	Instanzen an Discovery erfragen: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
	4	Verhinderung kaskadischer Fehler	C	Wir vom Framework das Circuit Breaker Pattern unterstützt	6	Circuit-Breaker-Pattern: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
Performance	5	Kurze Reakionszeit	B	Ist der Overhead vom Framework minimal?	7	Latenz Antwort bei Messung an Endpunkt ohne Logik
				Lassen sich schnell neue Instanzen hochfahren?	29	Messung Startzeit
	6	Hoher Durchsatz	B	Kann der Service Lastspitzen aushalten und bewältigen?	8	Druchsatz bei 100.000 Anfragen mit 255 echt parallelen Clients in Anfragen/Sek
	7	Der Service verwendet möglichst wenig Speicher	C	Ist der Service leichtgewichtig?	9	Messung Heap Size unter Last: Kleiner 100MB
Benutzbarkeit	8	Build-Management-Tool(s) vorhanden	A	Lässt sich automatisiert ein Build vom Service erstellen	10	Autom. Builds: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
				Werden Dependencies automatisch geladen?	11	Dependencieverwaltung: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	9	Die Entwicklung mit dem Framework ist effizient	B	Kann eine Endpunkt einfach erstellt werden?	12	LoC für einen Endpunkt
					13	Methodenaufrufe für einen Endpunkt
				Lässt sich das Framework einfach und schnell installieren?	14	Installation: Ordinalskala (sehr gut, gut, schlecht)
				Ist die Entwicklung schnell zu erlernen	15	Entwicklung mit Framework: Ordinalskala (sehr gut, gut, schlecht)
	10	Gute Dokumentation	A	Bietet das Framework eine umfangreiche Dokumentation mit Beispielen	16	Dokumentation: Ordinalskala (Sehr Umfangreich mit Beispielen, Umfangreich, Einfach, keine)
Sicherheit	11	Missbrauch vom Service kann unterbunden werden	B	Lassen sich über das Framework Events loggen	17	Logged Events: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	12	Der Zugriff auf den Service ist abgesichert	A	Lässt sich die API mittels JWT absichern?	18	JWT-Auth.: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)

Wartbarkeit	13	Der Status vom Service kann Zentral erfasst werden	B	Bietet das Framework eine Schnittstelle für Logging	19	Logging: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
				Bietet das Framework eine Schnittstelle für Metriken	20	Metriken: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	14	Framework wird supported	C	Wie lange wird ein Support gewährleistet?	21	Länge vom Support in Monaten pro Major Release
				Gibt es vom Hersteller Support	22	Kommerzieller Support vorhanden (j/n)
Zukunftssicherheit	15	Aktive Community	B	Werden Regelmäßig Fehler behoben?	23	Release Zykluslänge bzw. Durchschnittlicher Majorrelease Abstand
				Wie stark ist das Interesse von Firmen und wie sind deren Abhängigkeiten?	24	Businessimpact bei Verlust des Frameworks: Ordinalskala(stark, mittel, schwach)
					25	Große Firmen die auf das Framework setzen: Odrinalskala(viele, wenige, keine)
				Wie viele Benutzer gibt es	26	Github Repository : Star + Watch + Fork

Tabelle 5.1: Goal Question Metrik Ergebnis

5.3.3 Resümee Analysephase

Die Basisanforderungen stellen eine gute Grundlage für die Wahl der Anforderungen da und beschleunigen somit die Analysephase. Dadurch konnte der Quality Utility Tree schnell aufgebaut und priorisiert werden. Es bietet sich dabei an, dass der QUT mit den Basisanforderungen vor dem Brainstorming für die Anforderungen vorbereitet ist und entsprechend angepasst wird.

Bei der Definition von Metriken fällt auf, dass viele qualitative Metriken, die Ordinalskalen, verwendet wurden. Dies lässt den Ursprung von MFEM in der qualitativen Architekturbewertung erkennen. Aus Sicht des Autors bieten die Ordinalskalen einen guten Ansatz, um auch weiche Daten zu quantifizieren. Zusammen mit den weiteren Metriken kann das Framework so von allen Seiten betrachtet werden.

Wie sich der Einsatz der gefunden Metriken auswirkt, kommt aus dem den folgenden Phasen hervor.

5.4 Evaluationsphase

In der Evaluationsphase wird nun der Ablauf der Evaluation geplant und durchgeführt. Hierzu werden im nächsten Schritt Szenarien definiert.

5.4.1 Evaluation definieren

Die Evaluation wird in subjektiv und objektiv aufgeteilt. Für die subjektive Evaluation werden Szenarien definiert, die die Anforderungen bzw. Metriken untersuchen sollen.

MFEM sieht hierzu die Definition eines Standard-Anwendungsfalls vor, der in mehrere Teile, den Szenarien, aufgeteilt wird. Hier wurde die Erstellung eines einfachen Daten-Service definiert, der sich mit dem Beispiel aus Kapitel 4.3.1 deckt. Aus diesem Grund werden auch die Beispiel-Szenarien aus Kapitel 4.3.1 übernommen und hier nur kurz zusammengefasst:

Sz.1 Installation Installation der Sprache sowie des Frameworks und Einrichtung der Build-Tools

Sz.2 Einfacher Service Programmierung eines einfachen „Hello World"-Services, inklusive Security

Sz.3 Erweiterter Service Erweiterung des einfachen Services um ein Datenmodell (Aufgabenverwaltung)

Für die objektive Evaluation stehen nach MFEM bereits Messgruppen fest. Diese umfassen die Messung an Artefakten, wie z. B. dem Code, die Performance-Messung am lauffähigen Prototyp sowie der Recherche.
Um für die Messungen am laufenden Prototyp definierte Umstände zu haben, werden hier noch Anforderungsprofile benötigt. Dabei wurden folgende Profile für die Evaluation festgehalten:

Nr.	Typ	Parallele Verbindungen	Dauer	Besonderheit
1	Datenservice	255	30s	CRUD Operationen auf Datenbank
2	Einfacher Service	255	30s	Wenig rechenintensive Geschäftslogik

Tabelle 5.2: Anforderungsprofile für die objektiven Evaluation am Prototyp

Insgesamt ergeben sich für den nächsten Schritt somit 6 Messgruppen, denen Metriken zugeordnet werden können. Diese sind in Abbildung 5.5 nochmal zusammengefasst.

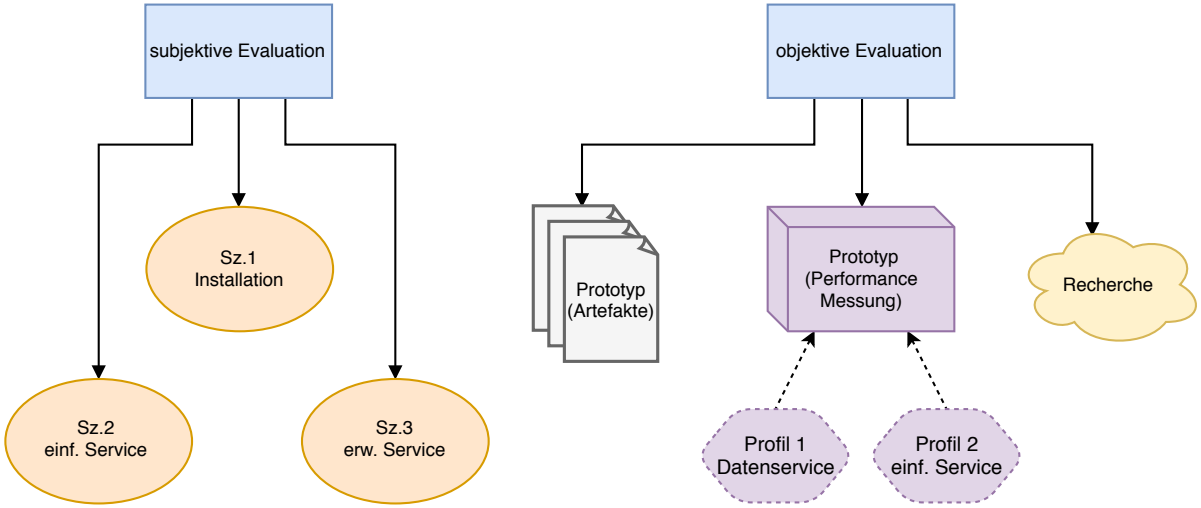

Abbildung 5.5: Zusammenfassung aller Messgruppen für Evaluation

5.4.2 Metriken zuordnen

Die Zuordnung der Metriken beginnt mit den Szenarios. Da diese zur subjektiven Evaluation gehören, werden nur weiche Daten erhoben. Im aktuellen Fall betrifft dies somit die mittels Ordinalskala erhobenen Metriken.

Da das Szenario 1 die Installation und Einrichtung übernimmt, wurden hier z. B. Metriken der Anforderungen Build-Tools oder Dependencieverwaltung zugeordnet. Mit dem Szenario 2, dem einfachen Service, konnten die Standardformate und Security (JWT) zugeordnet werden. Als letzte wurden dem Szenario 3, dem erweiterten Service, die Metriken für die REST-Schnittstelle sowie Objektvalidierung zugeordnet. Zusätzlich wurden diesem Szenario, da es das fortgeschrittenste ist, auch die Dokumentation und effiziente Programmierung zugeteilt.
Somit blieben am Ende des ersten Durchlaufs Metriken für Anforderungen wie z. B. Logging, Service-Discovery und Circuit-Breaker-Pattern übrig. Aus diesem Grund mussten die Szenarios weiter verfeinert werden. Die Wahl fiel dabei auf das Szenario 2. Es sollte somit nicht nur ein „Hello World“-Service entstehen, sondern auch ein Service der bereits in der Infrastruktur funktioniert. Somit ließen sich auch die letzten Metriken zuordnen.

Es ergibt sich somit die in Tabelle 5.3 gezeigte Zuordnung.

Szenario	M. Nr.	Metrik
1: Installation / Einrichtung	10	Autom. Builds: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	11	Dependencieverwaltung: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	14	Installation: Ordinalskala (sehr gut, gut, schlecht)
2: Einfacher Service	4	JSON-Format: Ordinalskala(automatisch, manuell, nicht möglich)
	5	XML-Format: Ordinalskala(automatisch, manuell, nicht möglich)
	6	Circuit-Breaker-Pattern: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
	17	Logged Events: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	18	JWT-Auth.: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	19	Logging: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	20	Metriken: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)
	27	Discovery An- und Abmeldung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
	28	Instanzen an Discovery erfragen: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
3: Erweiterter Service	1	Objekvalidierung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)
	2	HATEOAS-Links: Ordinalskala (automatisch, manuell, nicht möglich)
	3	Schnittstelle nach Ressourcen: Ordinalskala (automatisch, manuell, nicht möglich)
	15	Entwicklung mit Framework: Ordinalskala (sehr gut, gut, schlecht)
	16	Dokumentation: Ordinalskala (Sehr Umfangreich mit Beispielen, Umfangreich, Einfach, keine)

Tabelle 5.3: Szenarios: Zugeordnete Metriken

Da die objektiven Metriken sich nur an ihrem definierten Messpunkt erheben lassen, z. B. kann der Durchsatz nur am lauffähigen Prototyp ermitteln, ließen sich diese einfach auf die 3 verbleibenden Messgruppen verteilen.

Es ergibt sich somit die in Tabelle 5.4 gezeigte Zuordnung.

Evaluations Schritt	Messgruppe	M. Nr.	Metrik
Qualitative Evaluation	Artefakte (z.B. Code)	12	LoC für einen Endpunkt
		13	Methodenaufrufe für einen Endpunkt
	Performance	7	Latenz Antwort bei Messung an Endpunkt ohne Logik
		8	Druchsatz bei 100.000 Anfragen mit 255 echt parallelen Clients in Anfragen/Sek
		9	Messung Heap Size unter Last: Kleiner 100MB
		29	Messung Startzeit
	Recherche	21	Länge vom Support in Monaten pro Major Release
		22	Kommerzieller Support vorhanden (j/n)
		23	Release Zykluslänge bzw. Durchschnittlicher Majorrelease Abstand
		24	Businessimpact bei Verlust des Frameworks: Ordinalskala(stark, mittel, schwach)
		25	Große Firmen die auf das Framework setzen: Odrinalskala(viele, wenige, keine)
		26	Github Repository : Star + Watch + Fork

Tabelle 5.4: Objektive Evaluation: Zugeordnete Metriken

Da nun festgelegt ist, an welchem Punkt welche Daten erhoben werden, kann die Evaluation der Kandidaten durchgeführt werden.

5.4.3 Evaluation durchführen: Spring Boot

An dieser Stelle werden die Szenarien und Messungen an Spring Boot vorgenommen. Da es bei der Evaluation von MFEM weniger um die konkrete Implementierung[13] geht, wird die Durchführung der einzelnen Messgruppen nur kurz beschrieben. Dabei werden markante Punkte hervorgehoben, die die Ergebnisse nachvollziehbar machen.

Szenario 1: Installation

Mit Spring Boot wird Java als Programmiersprache benötigt. Die nötigen Installationsdateien lassen sich für jedes Betriebssystem bei Oracle[14] beziehen.

[13] Die gesamte Implementierung findet sich auf einem Github Repository mit der URL:`https://github.com/darenegade/SimpleSpringService`

[14] `http://www.oracle.com/technetwork/java`

Als Buildmanagement-Tools haben sich Gradle[15] und Maven[16] etabliert. Beide sind für die Verwendung mit Spring gut geeignet und daher fiel die Wahl auf Maven. Die Installation ist sehr einfach und es werden keine Vorkenntnisse benötigt. So konnten alle Metriken mit den bestmöglichen Werten bewertet werden.

M. Nr.	Metrik	Ergebnis
10	Autom. Builds: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	Verfügbar
11	Dependencieverwaltung: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	Verfügbar
14	Installation: Ordinalskala (sehr gut, gut, schlecht)	Sehr gut

Tabelle 5.5: Ergebnis von Szenario 1 für Spring

Szenario 2: Einfacher Service

Für die Erstellung eines neuen Projektes bietet Spring einen Initialisierungsservice[17] an. Dieser lässt einen die nötigen Spring Module auswählen und das Projekt vorkonfigurieren. Als Ergebnis erhält man ein vollständig konfiguriertes Maven Projekt. Dies macht den Einstieg sehr einfach.

Auch die Implementierung eines einfachen „Hello-World“-Service ist mit Spring denkbar einfach. Das Listing 1 zeigt was hierfür nötig ist. Daran sieht man sehr gut, dass die Einstiegshürde sehr gering ist. Mit @SpringBootApplication werden die Standard-Springkonfigurationen aktiviert. So lässt sich anschließend über @RestController und @PostMapping der „Hello-World“-Endpunkt erstellen. Mehr Konfiguration ist dabei nicht nötig, da Spring über die automatische Konfiguration die Parameter sowie den Rückgabewert der Methode erkennt und damit den Endpunkt anlegt. Auch die De- und Serialisierung in die Standardformate erfolgt automatisch und muss nicht manuell erstellt werden. Somit konnten diese Anforderungen mit automatisch bewertet werden.

[15] `https://gradle.org`
[16] `https://maven.apache.org`
[17] `http://start.spring.io`

Listing 1: „Hello-Word“ in Spring

```
@SpringBootApplication
@RestController
@EnableEurekaClient
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @PostMapping("/hello_service")
    public String helloService(@RequestBody Name name){
        return "Hello " + name.getName();
    }
}
```

Ähnlich einfach gestaltet sich die Einrichtung der Service-Discovery Unterstützung. Mit der zusätzlichen Annotation @EnableEurekaClient und einem Eintrag der Discovery-URL in der Datei application.properties ist der Service für die Discovery konfiguriert. Die zugehörige Anforderung konnte somit bestmöglich bewertet werden.
Diese Kombination von Annotationen und einfachen Konfigurationen ist typisch für Spring Boot und konnte auf sämtliche weiteren Anforderungen angewendet werden. Das Ergebnis in Tabelle 5.6 ist dementsprechend positiv.

M. Nr.	Metrik	Ergebnis
4	JSON-Format: Ordinalskala(automatisch, manuell, nicht möglich)	automatisch
5	XML-Format: Ordinalskala(automatisch, manuell, nicht möglich)	automatisch
6	Circuit-Breaker-Pattern: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	enthalten
17	Logged Events: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	leicht umsetzbar
18	JWT-Auth.: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	Verfügbar
19	Logging: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	leicht umsetzbar
20	Metriken: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	Verfügbar
27	Discovery An- und Abmeldung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	enthalten
28	Instanzen an Discovery erfragen: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	enthalten

Tabelle 5.6: Ergebnis von Szenario 2 für Spring

Szenario 3: Erweiterter Service

Das Szenario 3 sieht vor das Datenmodell für die Aufgabenverwaltung (Abbildung 4.11) zu implementieren und über eine REST-Schnittstelle nach außen anzubieten. Das Datenmodell wird dabei über die Java Persistence API (JPA) für Spring aufgebaut und die Konfiguration besteht somit auch aus Annotationen. Listing 2 zeigt dies als Beispiel für die Entität „Department“.

Listing 2: Umsetzung der Entität „Department“ mittels JPA

```
@Entity
public class Department extends BaseEntity{
String name;

@OneToOne
@NotNull
Employee head;

@OneToMany
Set<Employee> employees;

//... Getter + Setter
}
```

Die Besonderheit von Spring ist nun, dass mittels Spring-Data-Rest und dem in Listing 3 gezeigten Code, die komplette REST-Schnittstelle fertig konfiguriert ist. Das Repository-Interface muss lediglich als solches mit der Annotation @RepositoryRestResource gekennzeichnet und von CrudRepository<T, ID> abgeleitet werden, sodass Spring die Schnittstelle aufbauen kann. Diese ist vollständig nach Ressoucen aufgebaut und integriert in die Antworten automatisch Hypermedia As The Engine Of Application State (HATEOAS) Links.

Listing 3: Erstellung der REST-Schnittstelle für das „Department“ über ein Interface

```
@RepositoryRestResource
public interface DepartmentRepository extends
                                    CrudRepository<Department, UUID> {
}
```

Die Entwicklung eines Datenservice könnte somit kaum effizienter sein. Die automatische Konfiguration lässt sich auch anpassen. Hierzu bietet die Dokumentation detaillierte Beschreibungen und Code-Beispiele.
Somit konnte auch im 3. Szenario Spring sehr gut bewertet werden. Das Ergebnis ist in Tabelle 5.7 zu sehen.

M. Nr.	Metrik	Ergebnis
1	Objekvalidierung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	enthalten
2	HATEOAS-Links: Ordinalskala (automatisch, manuell, nicht möglich)	automatisch
3	Schnittstelle nach Ressourcen: Ordinalskala (automatisch, manuell, nicht möglich)	automatisch
15	Entwicklung mit Framework: Ordinalskala (sehr gut, gut, schlecht)	sehr gut
16	Dokumentation: Ordinalskala (Sehr Umfangreich mit Beispielen, Umfangreich, Einfach, keine)	Sehr Umfangreich

Tabelle 5.7: Ergebnis von Szenario 3 für Spring

Quantitative Evaluation

Der aus dem 3. Szenario entstandene Prototyp kann nun für die Erhebung der harten Daten herangezogen werden. So wurden die Lines of Code (LOC) und Methodenaufrufe für einen Endpunkt gemessen. Diese fallen, wie in Listing 1 gezeigt, sehr gering aus. So bleibt die Codebasis auch bei vielen Endpunkten schlank und gut wartbar.
Des Weiteren wurden am lauffähigen Prototyp Performance Messungen durchgeführt. Hierzu wurde das Anforderungsprofil 2 aus der Tabelle 5.2 auf den Prototyp angewendet und die entsprechenden Metriken, wie z. B. Latenz, Durchsatz und Speicherverbrauch unter Last, gemessen[18]. Die Anwendung des Anforderungsprofils 1 aus der Tabelle 5.2 wurde für die Bestimmung der Metriken nicht benötigt und daher nicht weiter beachtet.

Die Recherche hat den Erwartungen entsprochen. Da Spring ein sehr etabliertes Framework ist, ist auch die Community sehr aktiv. Es werden regelmäßig Fehler behoben und neue Major-Releases veröffentlicht. Dies spiegelt auch die Akzeptanz bei großen Firmen wider. So setzen nach Pivotal[19], der Organisation hinter dem Spring Framework, die 7 größten Banken oder die 9 größten Automanufakturen das Framework produktiv ein.

Die Ergebnisse der quantitativen Evaluation sind in der Tabelle 5.8 festgehalten. Mit diesem Schritt wurde die Evaluation von dem Spring Framework beendet.

[18] Für die Performance Messung wurde das Tool JMeter(`http://jmeter.apache.org`) eingesetzt und auf einem Laptop mit i7 2,3GHz Prozessor und 16GB DDR3 RAM durchgeführt.
[19] `https://pivotal.io`

Messgruppe	M. Nr.	Metrik	Ergebnis
Artefakte (z.B. Code)	12	LoC für einen Endpunkt	6
	13	Methodenaufrufe für einen Endpunkt	2
Performance	7	Latenz Antwort bei Messung an Endpunkt ohne Logik	40ms(Avg)
	8	Druchsatz bei 100.000 Anfragen mit 255 echt parallelen Clients in Anfragen/Sek	8624 Anfr/sec
	9	Messung Heap Size unter Last: Kleiner 100MB	718MB
	29	Messung Startzeit	5,3 Sekunden
Recherche	21	Länge vom Support in Monaten pro Major Release	10 Monate
	22	Kommerzieller Support vorhanden (j/n)	ja
	23	Release Zykluslänge bzw. Durchschnittlicher Majorrelease Abstand	8 Monate
	24	Businessimpact bei Verlust des Frameworks: Ordinalskala(stark, mittel, schwach)	stark
	25	Große Firmen die auf das Framework setzen: Odrinalskala(viele, wenige, keine)	viele
	26	Github Repository : Star + Watch + Fork	23958

Tabelle 5.8: Ergebnis der quantitativen Evaluation von Spring

5.4.4 Evaluation durchführen: Go-Kit

Der zweite Kandidat für die Evaluation ist Go-Kit. Auch hier wird nicht auf die komplette Implementierung[20] eingegangen, sondern nur markante Punkte hervorgehoben. Dabei beginnt die Evaluation mit dem 1. Szenario.

Szenario 1: Installation

Go-Kit wird, wie der Name vermuten lässt, mit Go entwickelt. Daher ist die Installation von Go der erste Schritt. Hierzu lassen sich die Installationsdateien von der Webseite `golang.org` herunterladen und installieren.
Damit sind auch die essentiellen Tools fürs Buildmanagement installiert. Diese sind z. B. `go get` für das Auflösen und Herunterladen von Abhängigkeiten (Dependencies) sowie `go test` für das Ausführen von Unit-Tests.
Diese Befehle ersetzen aber kein vollständiges Buildmanagement-Tool, welches den kompletten Prozess steuert und die richtigen Abhängigkeiten auflöst [2]. So ist es notwendig

[20] Die gesamte Implementierung findet sich auf einem Github Repository mit der URL:`https://github.com/darenegade/SimpleGoKitService`

den kompletten Prozess mittels Skript zu steuern und dabei Paketmanagement-Tools, wie Glide[21], zu nutzen.

Aus diesem Grund gab es für die Evaluation Abzüge bezüglich der Build-Tools. Die vollständige Bewertung ist in Tabelle 5.9 festgehalten.

Nr.	Metrik	Ergebnis
10	Autom. Builds: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	leicht umsetzbar
11	Dependencieverwaltung: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	Verfügbar
14	Installation: Ordinalskala (sehr gut, gut, schlecht)	Sehr gut

Tabelle 5.9: Ergebnis von Szenario 1 für GoKIt

Szenario 2: Einfacher Service

Im 2. Szenario wird ein einfacher „Hello World“-Service implementiert. Hierzu muss ein neues Projekt angelegt werden, welches im Workspace von Go, dem GoPath, erfolgen muss. Dabei ist das main.go File der Einstiegspunkt einer jeden Go Applikation.
Die Implementierung des einfachen „Hello-Word“-Endpunktes ist in Listing 4 dargestellt.

Listing 4: „Hello-Word“ in Go-Kit

```
func main() {
    logger := LOG.NewLogfmtLogger(os.Stdout)
    ctx := context.Background()
    svc := helloWorldService{}

    var helloWorldEndpoint endpoint.Endpoint
    helloWorldEndpoint = makeHelloWorldEndpoint(svc)

    helloWorldHandler := httptransport.NewServer(
        ctx, helloWorldEndpoint,
        decodeHelloWorldRequest, encodeResponse,
        httptransport.ServerErrorLogger(logger),
    )

    http.Handle("/hello_service", helloWorldHandler)
```

[21] https://github.com/Masterminds/glide

```
16          log.Fatal(http.ListenAndServe(":8080", nil))
        }
18
        type HelloWorldService interface {
20          helloService(string) (string, error)
        }
22
        type helloWorldService struct{}
24
        func (helloWorldService) helloService(name string) (string, error) {
26          return "Hello " + name, nil
        }
```

Dies zeigt sehr deutlich, dass die Erstellung eines Endpunktes relativ komplex ist. Es erfordert mehrere Konfigurationsschritte und muss vollständig manuell erfolgen, wobei der Ausschnitt nur ein Teil der gesamt nötigen Implementierung darstellt.
Auch ist die Verbindung zur Service Discovery an der Stelle noch nicht eingebaut. Go-Kit bietet für Eureka keine Unterstützung, sodass diese komplett eigenständig entwickelt werden muss. Da diese Anforderung mit „A“ bewertet wurde und auch weitere Metriken ein mittelmäßiges Ergebnis aufweisen, könnte an dieser Stelle bereits über ein Abbruch der Evaluation diskutiert werden.
Im Sinne der Evaluation von MFEM wurde entschieden, die Durchführung nicht abzubrechen. Das dabei entstanden Ergebnis ist in Tabelle 5.10 zusammengefasst.

Nr.	Metrik	Ergebnis
4	JSON-Format: Ordinalskala(automatisch, manuell, nicht möglich)	manuell
5	XML-Format: Ordinalskala(automatisch, manuell, nicht möglich)	manuell
6	Circuit-Breaker-Pattern: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	enthalten
17	Logged Events: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	schwer umsetzbar
18	JWT-Auth.: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	leicht umsetzbar
19	Logging: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	leicht umsetzbar
20	Metriken: Ordinalskala (Verfügbar, leicht umsetzbar, schwer umsetzbar)	leicht umsetzbar
27	Discovery An- und Abmeldung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	schwer umsetzbar
28	Instanzen an Discovery erfragen: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	schwer umsetzbar

Tabelle 5.10: Ergebnis von Szenario 2 für GoKIt

Szenario 3: Erweiterter Service

Auch beim erweiterten Service stößt man mit Go-Kit schnell an die Grenzen. Es bietet selber keine Unterstützung für eine Verbindung zu einer MySQL Datenbank. Um dies zu ermöglichen, wurde die Gorm[22] Bibliothek mit aufgenommen. Es unterstützt die gängigsten Operationen auf einer Datenbank und verwaltet dabei das Datenschema.
Da Go-Kit somit nichts über die Datenbank weiß, kann es die REST-Schnittstelle auch nicht nach Ressourcen aufbauen. Dies muss somit manuell erfolgen und für fast jede Operation muss ein eigener Endpunkt erstellt werden. Da dies analog zum Listing 4 erfolgt, entsteht sehr viel „Boilerplate“ Code, der die Entwicklung ineffizient macht. So wird der Code mit nur wenigen Endpunkten schnell unübersichtlich und schwer wartbar.
Bei der Umsetzung hilft die Dokumentation nur wenig. Sie enthält lediglich die Klassen sowie Packages und wenige einfache Beispiele. Tiefer greifende Erläuterungen sucht man vergebens.
Aus diesen Gründen ist die in Tabelle 5.11 dargestellte Bewertung von Go-Kit für das 3. Szenario ernüchternd ausgefallen.

Nr.	Metrik	Ergebnis
1	Objekvalidierung: Ordinalskala (enthalten, leicht umsetzbar, schwer umsetzbar)	schwer umsetzbar
2	HATEOAS-Links: Ordinalskala (automatisch, manuell, nicht möglich)	manuell
3	Schnittstelle nach Ressourcen: Ordinalskala (automatisch, manuell, nicht möglich)	manuell
15	Entwicklung mit Framework: Ordinalskala (sehr gut, gut, schlecht)	schlecht
16	Dokumentation: Ordinalskala (Sehr Umfangreich mit Beispielen, Umfangreich, Einfach, keine)	Einfach

Tabelle 5.11: Ergebnis von Szenario 3 für GoKIt

Quantitative Evaluation

Bei den Performance-Messungen konnte Go-Kit seine Stärken zeigen. Mit durchschnittlichen 9 Millisekunden Latenz kann der Service schnell Anfragen verarbeiten. So konnte er unter Last beachtliche 27.625 Anfragen/Sekunde beantworten und blieb dabei mit 31MB

[22] `http://jinzhu.me/gorm/`

im Speicher sehr effizient[23].

Die Recherche bestätigte jedoch das bisher relativ schlechte Ergebnis. Obwohl das Framework auf der offiziellen Webseite als Produktionsreif vermarktet wird [1], steht es derzeit noch in Entwicklung und ist aktuell in der Version 0.3 veröffentlicht. Somit gibt es zur Zeit weder einen offiziellen Support noch setzen zu Recht große Firmen auf dieses Framework.

Das Ergebnis der Quantitativen Evaluation ist in Tabelle 5.12 dargestellt.

Messgruppe	Nr.	Metrik	Ergebnis
Artefakte (z.B. Code)	12	LoC für einen Endpunkt	45
	13	Methodenaufrufe für einen Endpunkt	13
Performance	7	Latenz Antwort bei Messung an Endpunkt ohne Logik	9ms
	8	Druchsatz bei 100.000 Anfragen mit 255 echt parallelen Clients in Anfragen/Sek	27.625,2 Anfragen/sec
	9	Messung Speicherverbrauch (RSS) unter Last: Kleiner 100MB	31MB
	29	Messung Startzeit	18,4ms
Recherche	21	Länge vom Support in Monaten pro Major Release	unklar
	22	Kommerzieller Support vorhanden (j/n)	nein
	23	Release Zykluslänge bzw. Durchschnittlicher Majorrelease Abstand	Noch nicht v1
	24	Businessimpact bei Verlust des Frameworks: Ordinalskala(stark, mittel, schwach)	schwach
	25	Große Firmen die auf das Framework setzen: Ordinalskala(viele, wenige, keine)	keine
	26	Github Repository : Star + Watch + Fork	6850

Tabelle 5.12: Ergebnis der quantitativen Evaluation von Go-Kit

5.4.5 Resümee Evaluationsphase

Die Evaluationsphase ist die längste Phase von MFEM. So hat die praktische Umsetzung der Szenarien und Messungen sowie die Einarbeitung in das Framework bei beiden Kandidaten eine Woche betragen. Aus diesem Grund ist es wichtig, bei der Ermittlung der Metriken auf die Prioritäten der Anforderungen zu achten. Wie bei der Evaluation von Go-Kit zu sehen war, hätte schon das Ergebnis des 2. Szenarios zu einem Abbruch führen können.

[23]Für die Performance Messung wurde das Tool JMeter(`http://jmeter.apache.org`) eingesetzt und auf einem Laptop mit i7 2,3GHz Prozessor und 16GB DDR3 RAM durchgeführt.

Durch die genaue Definition der Szenarien mit den zugehörigen Metriken erfolgt die Durchführung der Evaluation zudem sehr zielgerichtet. Mit etwas Disziplin kann so eine ausschweifende Entwicklung verhindert und somit der Zeitaufwand weitestgehend verkürzt werden. Hierzu sollten, wie von MFEM vorgesehen, die Szenarien möglichst genau definiert werden.
Hierzu hat sich der Kreislauf zur Definition von Szenarien und der Zuordnung von Metriken in dieser Phase bewährt. So können keine Metriken übersehen werden und die Szenarien verfeinern sich zusehend.

5.5 Abschlussphase

In der Abschlussphase werden die Ergebnisse aus der Evaluation aufbereitet und präsentiert. Aufgrund der Fülle von Anforderungen und erhobene Daten, wird dies hier nur exemplarisch an einer Qualitätskategorie für ein Framework durchgeführt. Die Auswertung der Restlichen erfolgt analog und findet sich als Excel Tabelle in den jeweiligen Github-Repositories sowie auf der beigelegten Daten-CD.

Die Aufbereitung der erhobenen Daten wird mit der komplexen Auswertung erfolgen. Die Abbildung 5.6 zeigt den kompletten Prozess.

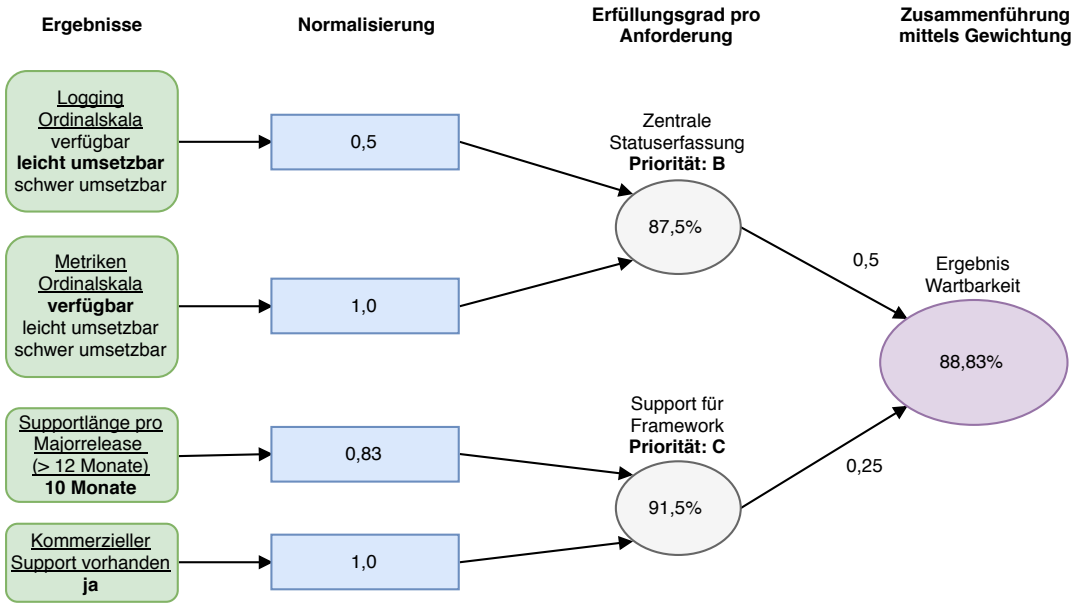

Abbildung 5.6: Auswertung und Zusammenführung aller Daten für die Qualitätskategorie „Wartbarkeit“ am Beispiel Spring

So wurden im ersten Schritt die ermittelten Ergebnisse auf einen Wert zwischen 0 und 1

normalisiert. Für die dreistufigen Ordinalskalen ergeben sich 3 mögliche Werte: 1 für den höchsten, $0,5$ für den mittleren und 0 für den niedrigsten Wert.
Bei den quantitativen Daten wird die prozentuale Abweichung zum Zielwert für die Normalisierung hergenommen. So ergab sich für die Supportlänge, mit einem Ergebnis von 10 und einem Zielwert von 12, ein Wert von $0,83$. Die Erfüllung einer Ja/Nein-Frage kann wie eine zweistufige Ordinalskala angesehen werden. So wurde der vorhandene Support mit 1 normalisiert.
Nachdem die einzelnen Werte für eine Anforderung aufsummiert wurden, kann der prozentuale Erfüllungsgrad bestimmt werden. Bei 2 Anforderungen würde ein Wert von 2 einer vollen Erfüllung (100%) entsprechen. Mit einem Wert von $1,5$ ergibt sich so ein Erfüllungsgrad von $87,5\%$.
Anhand der Prioritäten werden die Gewichte für die Zusammenführung der Anforderungen ermittelt. Eine mit „A“ bewertete Anforderung geht vollständig in das Ergebnis über. Die Priorität „B“ zählt zur Hälfte und „C“ nur ein Viertel. So ergibt sich das Gesamtergebnis von $88,83\%$ für die Wartbarkeit von Spring.

Nachdem alle Ergebnisse aller Qualitätskategorien für Spring und Go-Kit mit diesem Schema ausgewertet wurden, konnte das Netzdiagramm erstellt werden. Abbildung 5.7 zeigt das Gesamtergebnis von Spring und Go-Kit in einer Grafik.

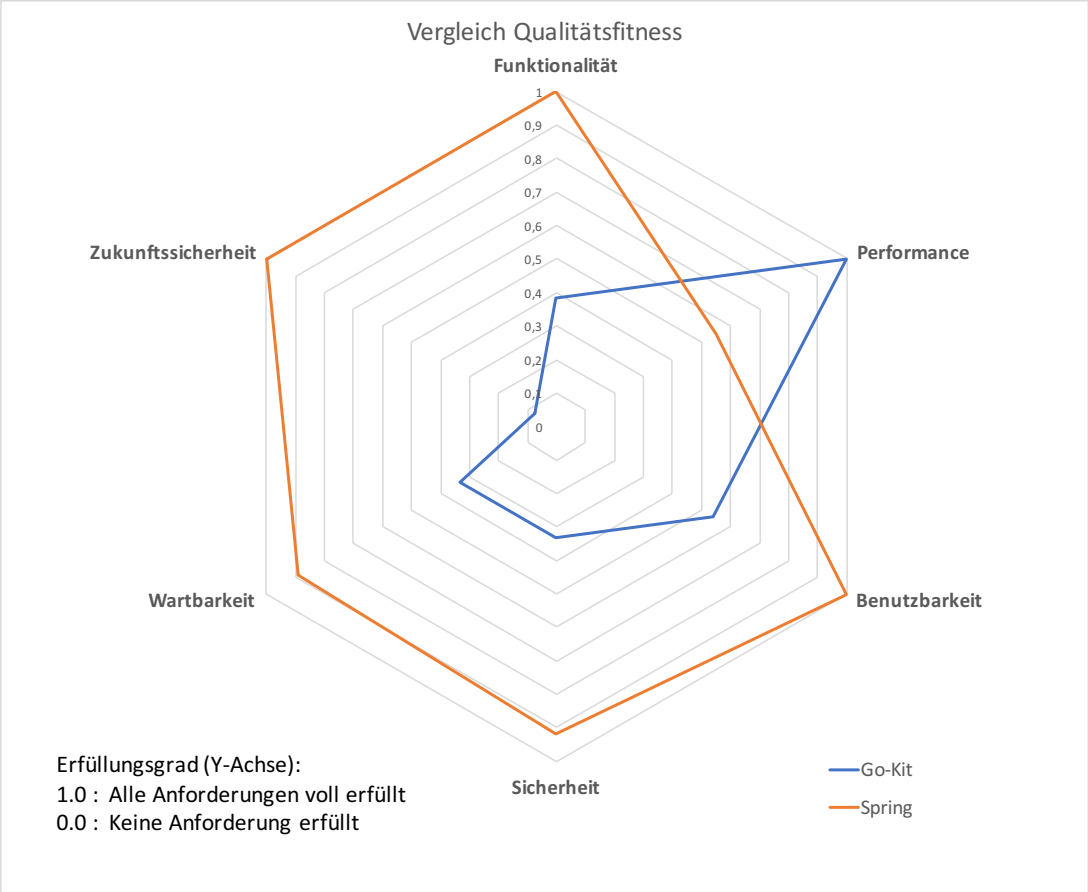

Abbildung 5.7: Gesamtergebnis von Spring und Go-Kit als Netzdiagramm

Das Gesamtergebnis macht die Überlegenheit von Spring deutlich. In nahezu jeder Kategorie hat es einen Vollausschlag. Lediglich bei der Performance kann Go-Kit Punkten. Aufgrund des miserablen restlichen Ergebnisses, wird jedoch auch dann von einem Einsatz im produktiven Umfeld abgeraten, falls die Geschwindigkeit ein maßgeblicher Faktor ist. Der Einsatz von Spring ist dagegen sehr empfehlenswert, was der Erfolg und die weite Verbreitung des Frameworks zeigt.

5.5.1 Resümee Abschlussphase

Die Auswertung der Daten ist sehr einfach und liefert ein ansehnliches Gesamtergebnis. Es kann dabei nur für sich stehen oder auch im Vergleich zu anderen Frameworks gut hergenommen werden, was die Abbildung **??** sehr gut zeigt. Auf dieser Basis kann schnell ein guter Überblick gewonnen werden, mittels dessen weitere Diskussionen durchgeführt oder Entscheidungen getroffen werden können.

5.6 Gesamtauswertung MFEM

Zusammenfassend kann gesagt werden, dass MFEM seinen Zweck erfüllt. Die Diskrepanz, die bei der Wahl der Kandidaten eine Anforderung war, hat sich in den Ergebnissen widergespiegelt. So wurde trivialerweise ein gutes Framework äußerst positiv und ein unausgereiftes Framework negativ bewertet. Hierzu wurden, bis auf wenige Ausnahmen, nur die Basisanforderungen genutzt. So kann auch gesagt werden, dass diese bereits eine gute Grundlage für eine Bewertung darstellen.
Sämtliche Anforderungen lassen sich durch die Analysephase an die jeweilige Situation anpassen oder erweitern, sodass die Methode flexibel bleibt. Wie hier gezeigt wurde, entsteht so schnell eine Vielzahl an Anforderungen. Dazu bietet der Quality Utility Tree einen guten Ansatz, um den Überblick zu behalten und sich nicht in Feinheiten zu verlieren.
Die erste Hürde stellt die Wahl der passenden Metriken dar. Dies war auch bei dieser Evaluation durchaus eine Herausforderung. Auch wenn hier bis zuletzt kein Patentrezept für die Wahl der bestmöglichen Metriken ausgestellt werden kann, stellt die GQM Methode einen guten Top-Down Ansatz dar, um dies zu unterstützen.

Bis zur Evaluationsphase, und selbst dort erst im letzten Schritt, spielt das eigentlich zu testende Framework keine Rolle. Dadurch bleibt die Methode sprachunabhängig und verkürzt sich bei mehrfacher Anwendung an verschiedenen Frameworks. Dieser Gewinn ist jedoch nur marginal, da die Durchführung der Evaluation am Framework den größten Zeitaufwand einnimmt. Hierzu bietet die Methode mit den Szenarien einen guten Ansatz, um den verhältnismäßig langen Prozess zumindest zielgerichtet zu gestalten. Wie sich gezeigt hat, kann durch die Priorisierung auch ein schneller Abbruch zustande kommen. So wird wenigstens nicht unnötig Zeit in einen negativen Kandidaten gesteckt.
Einzig die Abbruchbedingung wird von MFEM nicht vorgegeben und sollte vor der Durchführung definiert werden. Eine Bedingung im Zusammenhang mit den Prioritäten ist hierfür prädestiniert.

Das mittels Netzdiagramm dargestellte Gesamtergebnis bietet einen sehr guten und schnellen Überblick über das Ergebnis. So kann der Architekt auch vor nicht allzu technisch versierten Entscheidungsträgern seine Wahl anschaulich begründen, ohne zu tief ins Detail gehen zu müssen.

6 Fazit und Ausblick

Mit MFEM wurde eine Methode vorgestellt, die ein Framework von allen Seiten zielgerichtet auf eine Microservice Architektur hin untersucht. Sie bietet viele Ansätze, um den Prozess so effizient wie möglich zu gestalten. Dabei verliert sie nicht an Flexibilität und kann an die jeweilige Situation angepasst werden. Dies geht dabei über die Wahl der passenden Anforderungen und Szenarien hinaus.

6.1 MFEM Anpassung/Erweiterung

Der wohl größte Kritikpunkt an MFEM ist der hohe Zeitaufwand für die Evaluation. Auch wenn für die voll umfängliche Bewertung eines Frameworks, ohne dabei Vorkenntnisse zu verlangen, Zeit benötigt wird, kann diese verkürzt werden. Anstatt sämtliche Anforderungen in die Evaluationsphase zu überführen, kann anhand der Prioritäten eine Auswahl getroffen werden.
So können z. B. nur Anforderungen mit der Priorität „A“ untersucht werden. Dies hat den Vorteil, dass sich die Evaluation erheblich verkürzt. Im Gegenzug ergeben sich so jedoch blinde Flecken, die ein größeres Restrisiko hinterlassen.

Des Weiteren könnte MFEM auch durch eine Langzeitsicht erweitert werden. Eine regelmäßige Bewertung, z. B. pro Major Release, würde einen Trend aufzeigen. So werden vielleicht in späteren Versionen benötigte Features nicht mehr unterstützt oder ein anfänglich schlechtes Framework entwickelt sich zur positiven Alternative.

6.2 Ausblick

In jedem Fall wäre eine Anwendung von MFEM auf die aktuell relevanten Frameworks interessant. Anhand der allgemeinen Basisanforderungen ließe sich sogar ein Ranking aufstellen. Viele derzeit publizierte Rankings, wie [18] und [15], betrachten nur einen Aspekt oder beziehen sich rein auf Umfragen.
Mit MFEM würde hingegen eine detailliertere Bewertung erfolgen, aufgrund dessen fundierte Technologieeinscheidungen getroffen werden können.

7 Quellenverzeichnis

[1] Peter Bourgon. *FAQ: Is GoKit production ready?* Zugriff: 23.02.2017. URL: https://gokit.io/faq/#is-go-kit-production-ready.

[2] Sam Boyer. *The Saga of Go Dependency Management.* Zugriff: 23.02.2017. URL: https://blog.gopheracademy.com/advent-2016/saga-go-dependency-management/.

[3] TIOBE software BV. *TIOBE Index.* Zugriff: 18.02.2017. URL: http://www.tiobe.com/tiobe-index/.

[4] Tom DeMarco. „Software Engineering: An Idea Whose Time Has Come and Gone?“ In: *IEEE Software* 26 (2009), S. 96, 95. ISSN: 0740-7459.

[5] L. Dobrica und E. Niemela. „A survey on software architecture analysis methods“. In: *IEEE Transactions on Software Engineering* 28.7 (Juli 2002), S. 638–653. ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1019479.

[6] Felipe Gutierrez. „Pro Spring Boot“. In: 1st. Berkely, CA, USA: Apress, 2016. ISBN: 9781484214329.

[7] Sascha-Ulf Habenicht. *Das GQM-Paradigma.* Zugriff: 22.10.2016. URL: www.sascha.habenicht.name/publikationen/GQM-Paradigma.pdf.

[8] Marcus Hegner. *Methoden zur Evaluation von Software.* Zugriff: 22.10.2016. URL: http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/iz_arbeitsberichte/ab_29.pdf.

[9] Software Engineering Institute. *Architecture Tradeoff Analysis Method.* Zugriff: 22.10.2016. URL: http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm.

[10] ISO/IEC. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.* Techn. Ber. 2010.

[11] William L. de Oliveira und Juan A. de la Puente Juan C. Dueñas. „A Software Architecture Evaluation Model“. In: *Development and Evolution of Software Architectures for Product Families: Second International ESPRIT ARES Workshop Las Palmas de Gran Canaria, Spain February 26–27, 1998 Proceedings.* Springer Berlin Heidelberg, 1998, S. 148–157. ISBN: 978-3-540-68383-4.

[12] Jason Kincaid. *Google's Go: A New Programming Language That's Python Meets C++.* Zugriff: 18.02.2017. URL: https://techcrunch.com/2009/11/10/google-go-language/.

[13] Marek Kirejczyk. *Hype Driven Development.* Zugriff: 25.02.2017. URL: https://blog.daftcode.pl/hype-driven-development-3469fc2e9b22.

[14] Josh Long. *SpringOne Platform 2016 Recap: Day 1.* Zugriff: 18.02.2017. URL: https://spring.io/blog/2016/08/03/springone-platform-2016-recap-day-1.

[15] Bruz Marzolf. *Hot Web Framework*. Zugriff: 23.02.2017. URL: `http://hotframeworks.com`.

[16] Rick Kazman und Mark Klein Paul Clements. *ATAM: Method for Architecture Evaluation*. Zugriff: 22.10.2016. URL: `http://www.sei.cmu.edu/reports/00tr004.pdf`.

[17] Suzanne Robertson und James Robertson. *Mastering the Requirements Process (2Nd Edition)*. Addison-Wesley Professional, 2006. ISBN: 0321419499.

[18] Hartmut Schlosser. *Technologie-Trends 2017: Das sind die Top-Frameworks*. Zugriff: 25.02.2017. URL: `https://jaxenter.de/framework-trends-2017-53153`.

[19] Gernot Starke. *Effektive Softwarearchitekturen - Ein praktischer Leitfaden*. M: Carl Hanser Verlag GmbH Co KG, 2015. ISBN: 978-3-446-44406-5.

[20] Cathleen Wharton u. a. „Usability Inspection Methods“. In: John Wiley & Sons, Inc., 1994. Kap. The Cognitive Walkthrough Method: A Practitioner's Guide, S. 105–140. ISBN: 0-471-01877-5.

[21] Eberhard Wolff. *Eine frühlingshafte Lösung für Microservices: Spring Boot*. Zugriff: 18.02.2017. URL: `https://jaxenter.de/eine-fruehlingshafte-loesung-fuer-microservices-spring-boot-42028`.

[22] Eberhard Wolff. *Microservices Grundlagen flexibler Softwarearchitekturen*. Heidelberg, Neckar: dpunkt, 2015. ISBN: 978-3-8649-0313-7.

Todo list

Selbstständigkeitserklärung

Zarwel, René
(Familienname, Vorname)

München, 13.03.2017
(Ort, Datum)

21.10.1987
(Geburtsdatum)

Informatik - 7B / WS - 2013
(Studiengruppe / WS/SS)

Erklärung

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, den 13.03.2017

René Zarwel

Anhang

A MFEM - Vollständiger Quality Utility Tree der Basisanforderungen