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: XX.XX.XX

I Kurzfassung

Insert Abstract Here

Abstract

Insert Abstract Here

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

V Listing-Verzeichnis

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
MFEM Microservice Framework Evaluation Method
REST Representational State Transfer
CW Cognitive Walkthrough
JWT JSON Web Token

1 Einleitung

TODO 1: Einführung + Motivation für Erstellung Methode

2 Microservices

TODO 2: Kurz Microservices Basics + Motivation einfügen und beschreiben

Die Auswahl des richtigen Frameworks für einen einzelnen Service hängt von vielen Faktoren ab. Damit neben den Kundenanforderungen nicht Anforderungen von der Microservice Architektur vergessen und objektiv in den Auswahlprozess mit einbezogen werden, sollte es eine Methode zur Bewertung des Frameworks geben. Dabei kann die durch das Framework vorgegebene Architektur und die bereits enthaltenen Funktionen bewertet werden. So kann, schon während der Entwurfsphase, eine fundierte Aussage über die zu erwartenden Risiken und Trade-Offs getroffen werden. Sollte z. B. eine wesentliche Funktion, wie eine Authentifizierung am Service, fehlen, müsste diese selbstständig entwickelt werden. Die kann, je nach Architektur, sehr aufwendig sein. Damit verbunden wäre ein hoher Zeitaufwand, der die Projektkosten immens steigern würde. Um eine passende Bewertungsmethode zu finden, können bereits etablierte Methoden zur Architekturbewertung herangezogen werden. Der Vorteil hierbei ist, dass diese bereits getestet, weiterentwickelt und in vielen Unternehmen verwendet werden. So ist der Aufwand für die Integration der neuen Methode sehr gering, da sie in bereits bestehende Bewertungsprozesse integriert werden kann. In diesem Zusammenhang stellt sich die grundsätzliche Frage, warum Architekturen in der professionellen Softwareentwicklung bewertet werden? Und ob es dafür bereits etablierte Methoden gibt?

3 Qualitätsbewertung von Softwarearchitektur

> „*You cannot control what you cannot measure*“
> - Tom DeMarco (Controlling Software Projects, 1982)

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 [10]. Auch wenn DeMarco 2009 seine Aussage relativiert hat [1], 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 [10]. 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 1)

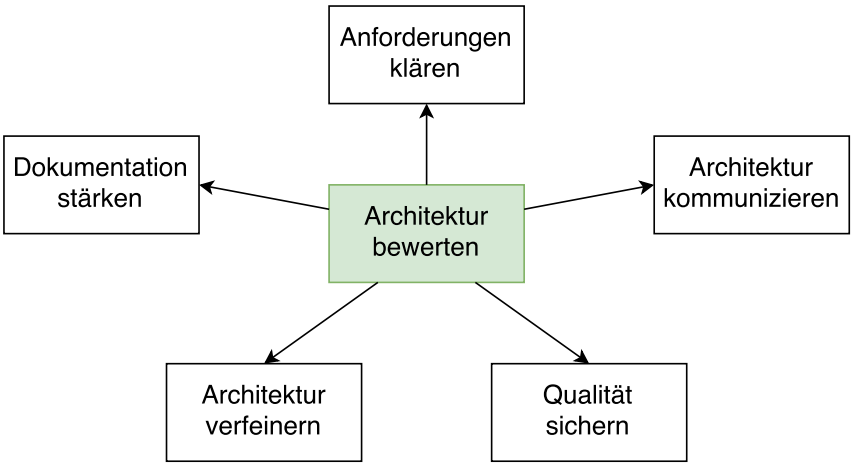

Abbildung 1: Allgemeine Ziele von Architekturbewertung.

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.

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

3.1 Anforderungen, Merkmale und Ziele

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 [9]. Ein Beispiel:

„Der Kunde soll im Onlineshop Artikel in einen Warenkorb legen können."

Nicht-funktionalen Anforderungen beschreiben *in welcher Qualität* die zu erbringenden Leistung erfolgen soll [9]. Ein Beispiel:

„Der Bestellvorgang soll nicht länger als 2 Sekunden dauern."

In der Regel werden diese Anforderungen durch Verträge, Normen oder Spezifikationen festgelegt. Diese begründen sich meistens auf die Ziele des Produktes. So will der Onlineshop-Betreiber Frust beim Kunden verhindern und Spontankäufe begünstigen, indem er den Bestellvorgang möglichst kurz hält. Eine Untersuchung der Produktmerkmale, also der charakteristischen Kennzeichen eines Produktes, gibt dabei Aufschluss über die Erfüllung der Anforderungen. So kann der vorher genannte Bestellvorgang bei der Messung 5 Sekunden benötigen und damit die Anforderung nicht erfüllen.

3.2 Qualitätsmerkmale der Softwarearchitektur

Auch bei der Architekturbewertung sind die Anforderungen das Fundament. Sie charakterisieren die gewünschten Eigenschaften eines Systems [10]. Bei der Softwarearchitektur gibt es eine Vielzahl an Qualitätsmerkmale anhand derer Anforderungen bestimmt und definiert werden können [10, 8]:

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 umzusetzen 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.

3.3 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 [10]:

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 ATAM wurde die SAAM von der SEI weiterentwickelt und stellt heute die führende Methode zur Architekturbewertung dar [5].

3.4 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 [8]. 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 [8] 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 beziehen 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 [8]. 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 2 gibt hierzu einen Überblick.

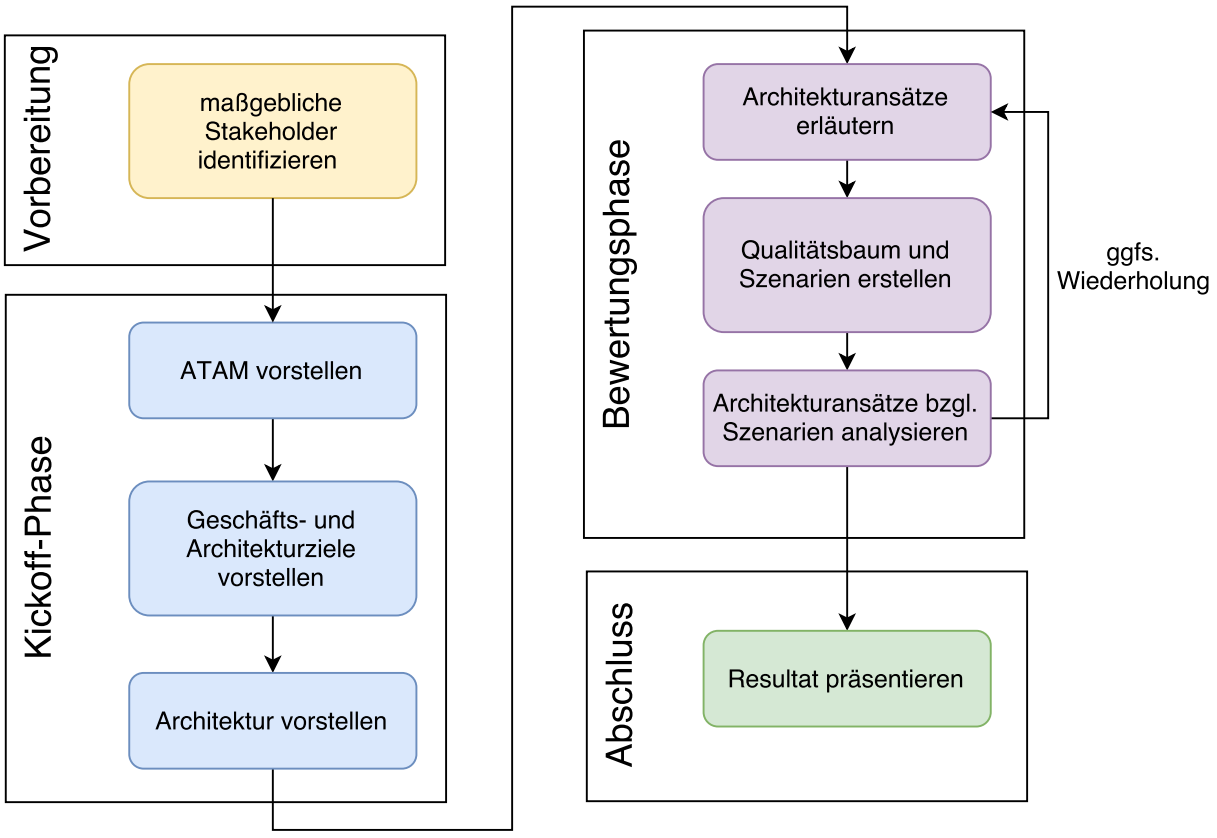

Abbildung 2: Phasen der Architekturbewertung nach ATAM [10]

3.4.1 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 [10], direkt zur Architekturbewertung eingeladen. Zu viele Personen würden den Prozess, durch Diskussionen und Abschweifungen, nur verlangsamen und ineffektiv gestalten.

3.4.2 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 [10]. 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 [8]:

- 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 [10]. 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 [10]. 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 [8], 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 [8, 10]:

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

3.4.3 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 [10]. 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 [8]. 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 stellt ein Beispiel für einen solchen Qualitätsbaum dar.

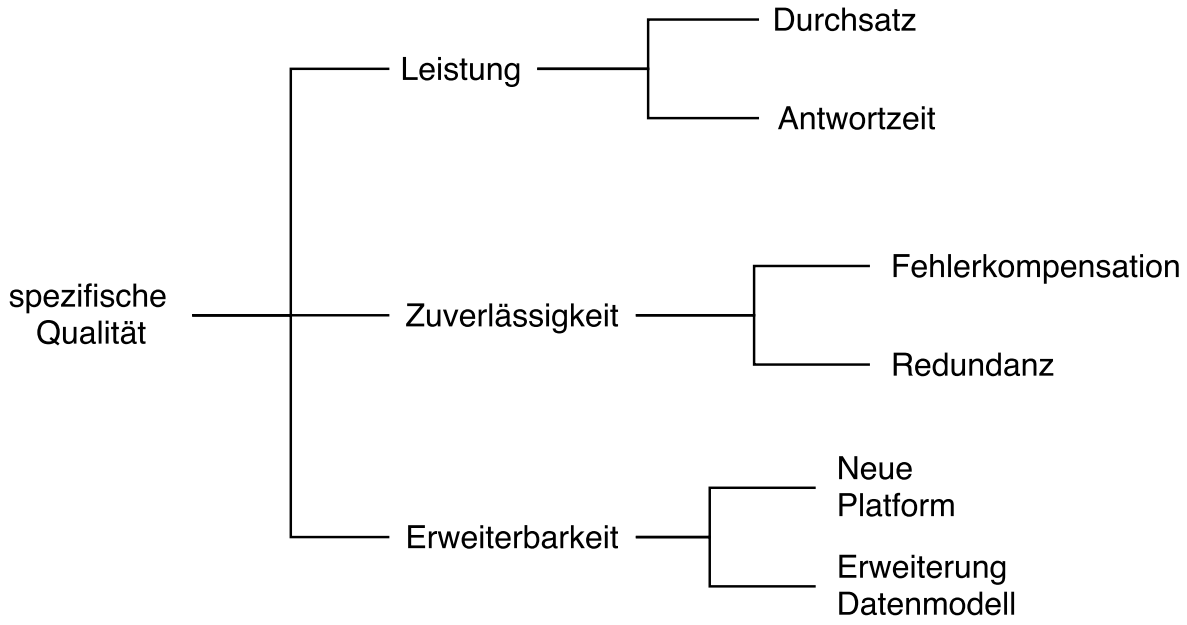

Abbildung 3: Beispiel eines Qualitätsbaums.

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 [10]:

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 4 dargestellt.

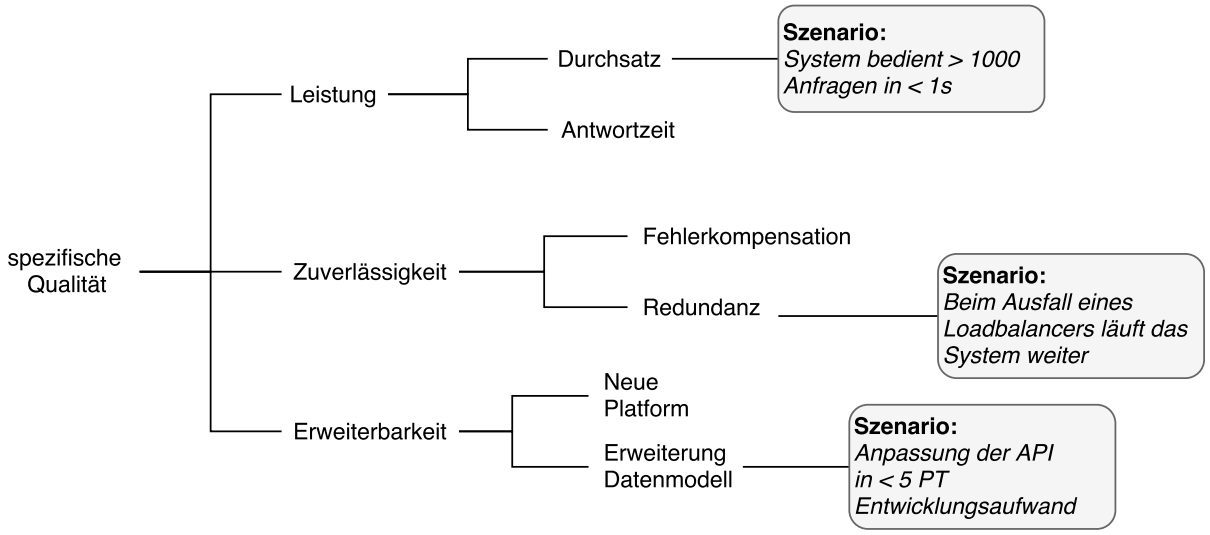

Abbildung 4: Qualitätsbaum mit einzelnen Szenarien erweitert

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 [10]. 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.

3.4.4 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 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.

3.5 Software Architecture Evaluation Model (SAEM)

Im Gegensatz zu ATAM ist SAEM weniger bekannt und erprobt [2]. 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

[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.

Qualität und Kapazitäten des Systems haben [7]. Dies trifft besonders auf die nicht-funktionalen Anforderungen, wie Wartbarkeit, Skalierbarkeit oder Effizienz, zu.

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 [7]. SAEM versucht dies mittels eines Qualitätsmodells auf die Softwarearchitektur abzubilden und stellt Methoden sowie Metriken für die Evaluation bereit.

3.5.1 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 [6]. Anhand derer kann die Qualität der Softwarearchitektur spezifiziert werden, indem man diesen Architekturanforderungen zuordnet.

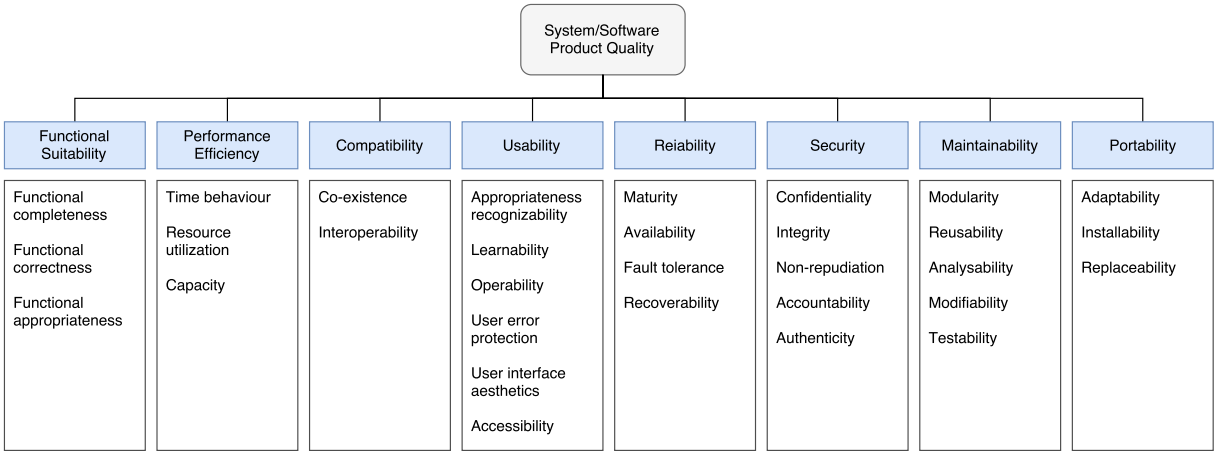

Abbildung 5: Qualitätsmodell der ISO/IEC 25010 [6]

Hierzu wird im ersten Schritt die externe Qualität bestimmt. Diese basiert auf messbaren Metriken unter Nutzungsbedingungen und werden vom Nutzer sowie den Architekten definiert [7]. 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 [7]. Die Bestimmung dieser basiert

meist auf Expertenwissen oder Firmen interne Richtlinien.

3.5.2 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 [3]. 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 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 [3] 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 6 dargestellt.

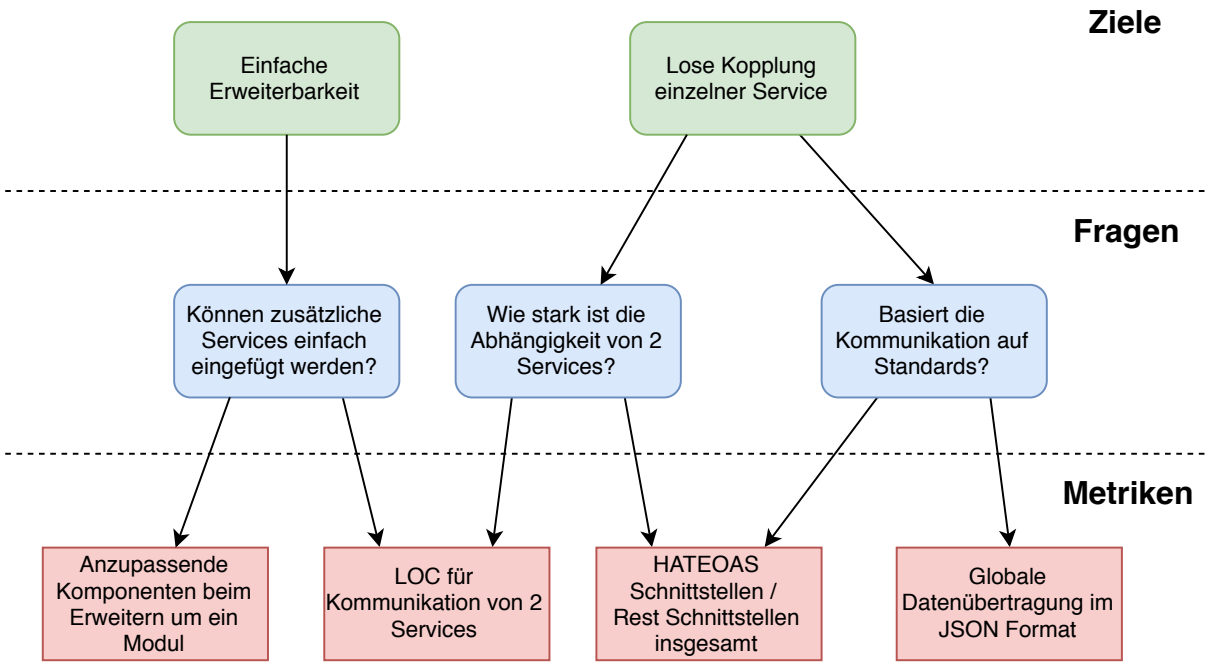

Abbildung 6: Beispiel für die Anwendung von GQM an einer Microservice Architektur

3.5.3 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 [2], die auf Erfahrung und Expertenwissen basiert.

3.6 Architekturbewertungsmethoden und Microservices

TODO 3: Resümee der beiden Methoden und Klärung: „Warum schlecht für MS-Frameworks?“

Mit dem Wissen über verschiedene Ansätze zur Qualitätsbewertung von Softwarearchitektur kann nun versucht werden, eine Methode zu finden, die es zur Bewertung von Microservice Frameworks und die daraus vorgegebenen Architektur des Services nutzt.

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 MFEM keine reine Architekturbewertungs-Methode. Sie betrachtet einerseits die durch das Framework vorgegebene Servicearchitektur und versucht dabei Risiken aufzudecken sowie ein tieferes Verständnis für die Architektur zu schaffen. Auf der anderen Seite wird das Framework als Produkt aufgefasst und versucht, eine Aussage über die Qualität dessen zu treffen. 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 7 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.

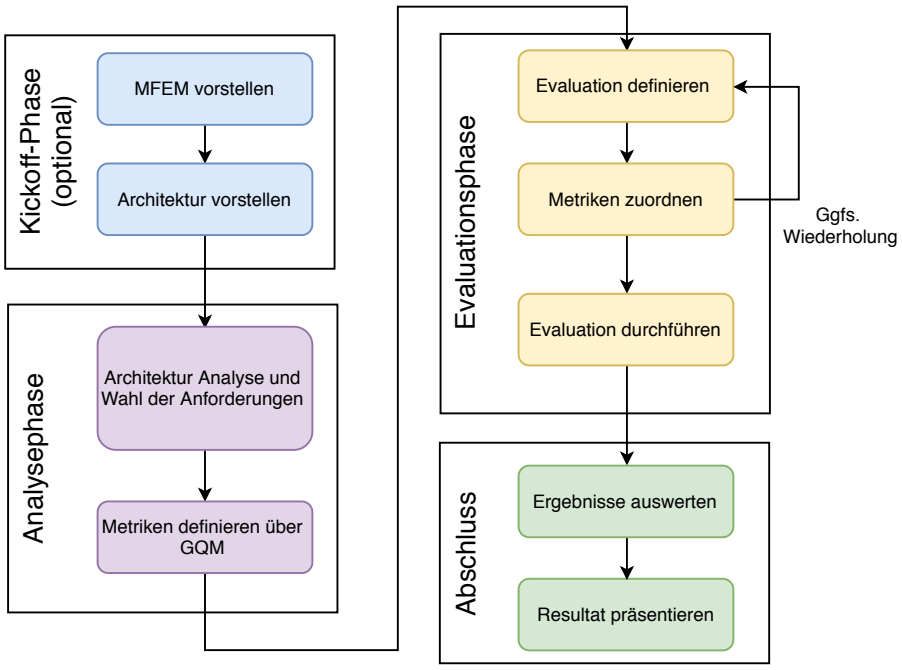

Abbildung 7: Schematische Darstellung vom Ablauf von MFEM

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

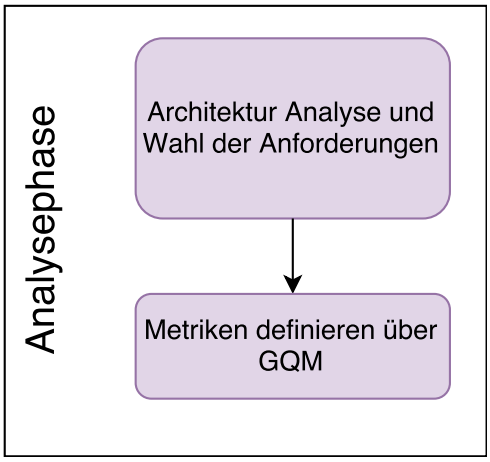

Abbildung 8: Analysephase von MFEM

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.

[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

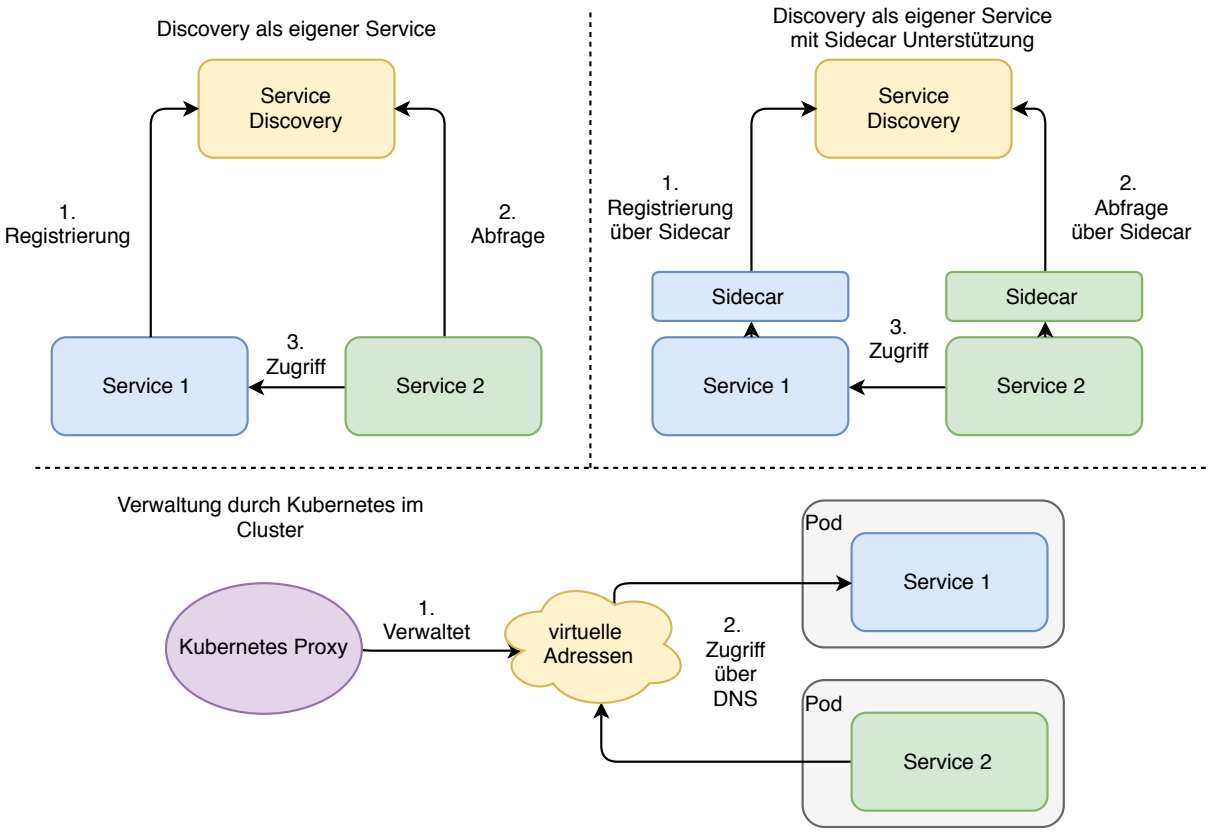

Abbildung 9: Beispiele für Ausprägungen der Servicediscovery

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. 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.

TODO 4: anders Formulieren und ausschmücken

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

Entwicklung Quality Utility Tree

TODO 5: Priorisierung von Anforderungen

Der Quality Utility Tree bietet einen effizienten Top-Down Ansatz zur Identifizierung und Verfeinerung von Qualitätsanforderungen. Die mittels Brainstorming gefundenen Anforderungen werden in einzelne Kategorien aufgeteilt und bis zu den konkreten Zielen weiter verfeinert. Bei der Kategorisierung kann das Qualitätsmodell aus der ISO/IEC 25010 hergenommen werden. Wobei das Modell nur eine Hilfestellung ist und keine Vorgabe darstellt. Hier wurden die Kategorien Funktionalität, Performance, Benutzbarkeit, Sicherheit und Wartbarkeit definiert. Anschließend wurden diesen die Qualitätsanforderungen zugeordnet. Bild 10 zeigt einen Ausschnitt des gesamten Baumes, wobei der vollständige Baum im Anhang zu finden ist.

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

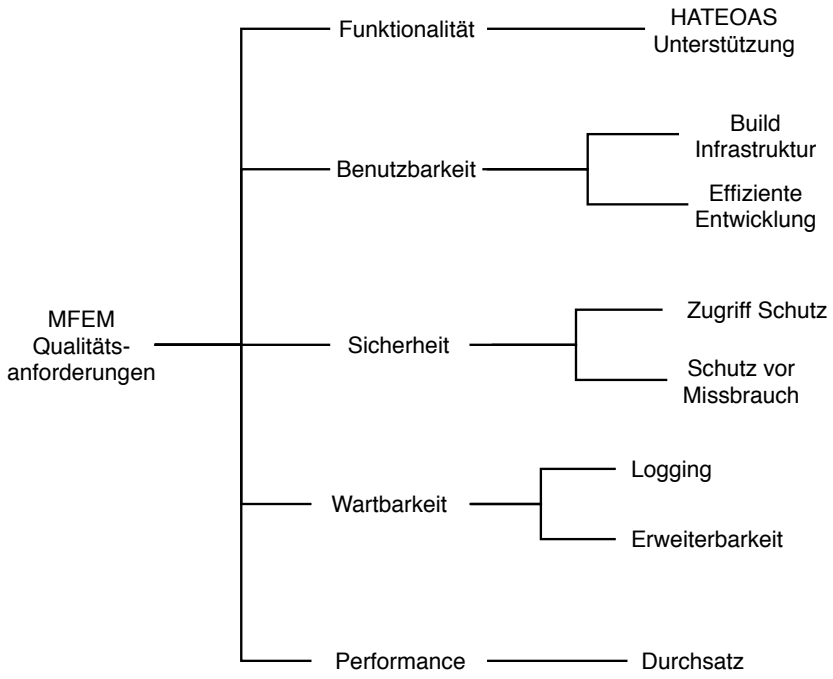

Abbildung 10: Ausschnitt aus dem Quality Utility Tree der Basisanforderungen

4.2.2 Metriken definieren über GQM

TODO 6: Ausführliche Ablauf erklären

Für die identifizierten Qualitätsanforderungen müssen 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. 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 Representational State Transfer (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 11 dargestellt. Wie zuvor ist dies nur ein Ausschnitt, wobei der gesamte Baum im Anhang zu finden ist.

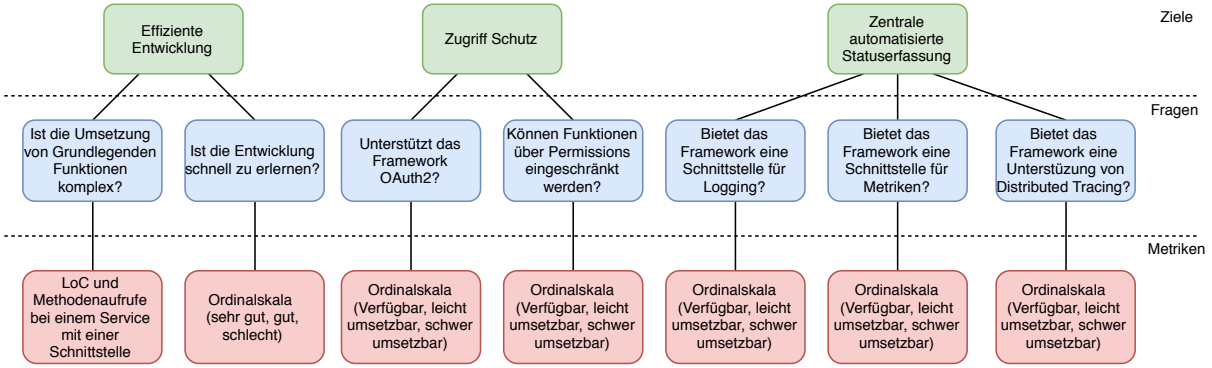

Abbildung 11: Ein Ausschnitt aus dem Qualitätsbaum erweitert um Metriken

4.3 Evaluationsphase

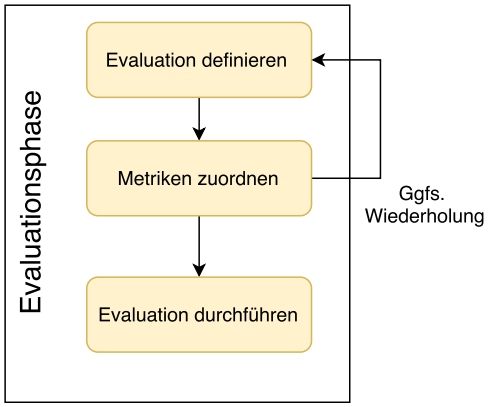

Abbildung 12: Evaluationsphase von MFEM

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 [4]. Dabei versucht die subjektive Evaluation eine Beurteilung durch den Benutzer 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 [4]. 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 [4]. 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 [4]:

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 zusam-

mengefasst 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. [11] 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 13 abgebildet.

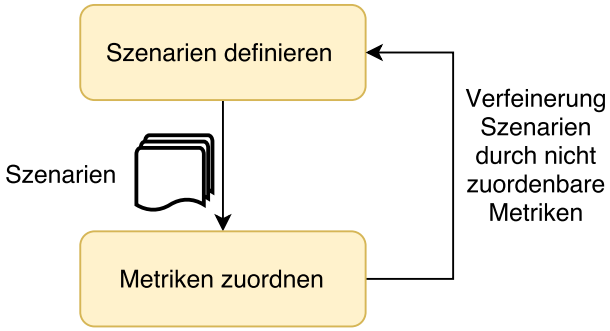

Abbildung 13: Kreislauf zur Definition und Verfeinerung der Szenarien

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.

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

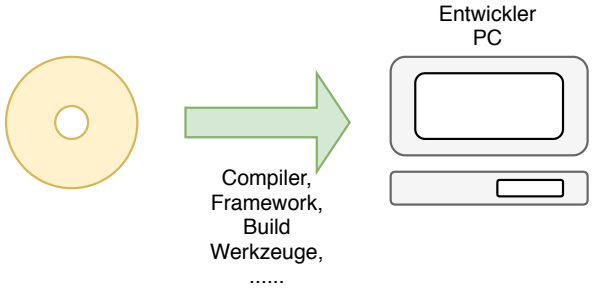

Abbildung 14: Szenario 1: Installation des Frameworks mit allen benötigten Komponenten.

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 gewonnen. 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.

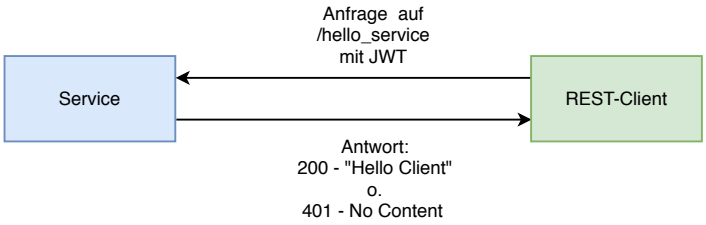

Abbildung 15: Szenario 2: Erstellung eines einfachen Hello-World-Services.

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

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

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 17 dargestellt.

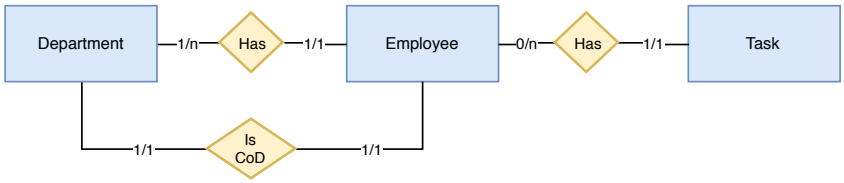

Abbildung 16: Szenario 3: Datenmodell der Aufgabenverwaltung

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.

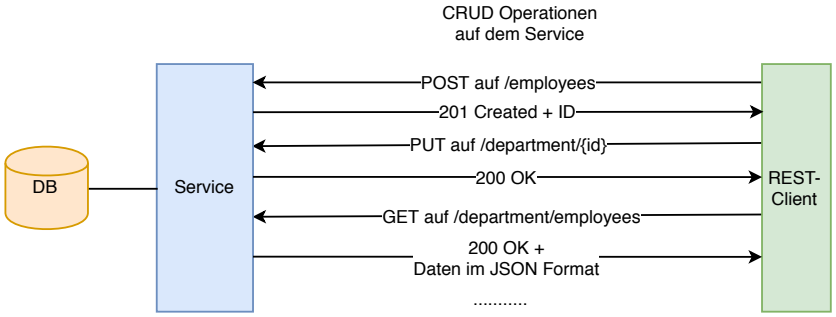

Abbildung 17: Szenario 3: Erstellung eines erweiterten Services.

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.

Nr.	Name	parallele Anfragen	Wieder-holungen	Besonderheit
1	Datenservice	255	500	CRUD Operationen auf Datenbank
2	Einfacher Service	255	500	Wenig rechenintensive Geschäftslogik
3	Komplexer Service	255	500	Stark rechenintensive Geschäftslogik

Tabelle 1: Beispiel: Anforderungsprofile für die objektiven Evaluation am Prototyp

Das erste Anforderungsprofil 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 das Profil zwei definiert. Dieses baut 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 dieses besonders groß, um z. B. die übertragenen Daten zu de- und serialisieren, beeinflusst es jede Anfrage 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. Qualitätsmerkmale wie Lernbarkeit und Effizienz lassen sich dabei besser in der subjektiven und Performance sowie Wartbarkeit eher in der objektiven Evaluation bestimmen. Welche Metriken genau in welcher Evaluationsphase ermittelt werden sollen, wird in diesem Schritt definiert.

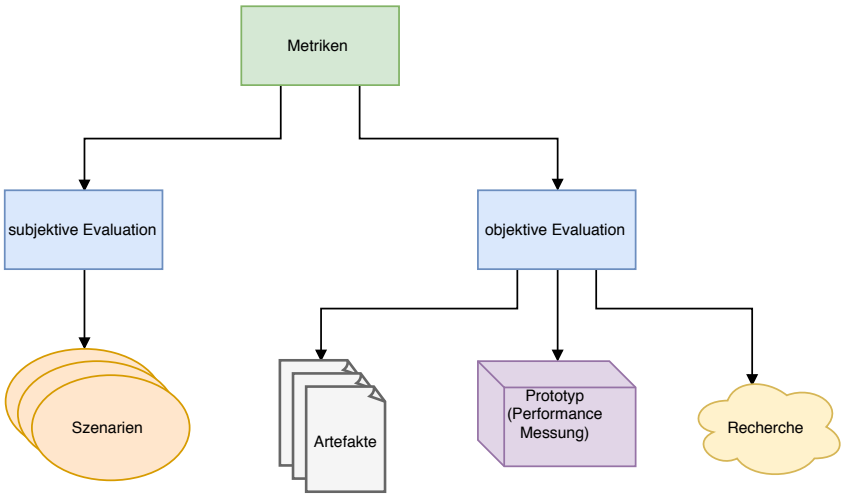

Abbildung 18: Aufteilung der Metriken in subjektiv und objektiv sowie Zuordnung zu einzelnen Messpunkten

Sollten sich einzelne Metriken nicht zuordnen lassen, können die Szenarien und Profile des vorangegangenen Schrittes erweitert oder verfeinert werden. Dabei sind die Anforderungen absteigend der Prioritäten zu verteilen.

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 objektive 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

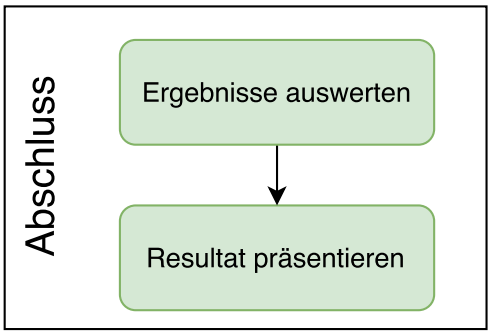

Abbildung 19: Abschlussphase von MFEM

4.4.1 Ergebnisse auswerten

TODO 7: Auswerten usw..

4.4.2 Resultat präsentieren

TODO 8: Komplett überarbeiten

In diesem Schritt werden die Ergebnisse präsentiert und . Dies muss dabei nicht zwangsläufig eine Präsentation sein. Viel mehr geht es um das Aufbereiten der Daten. 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.

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

Anhand der Ergebnisse kann eine eindeutige und fundierte Aussage über die Akzeptanz des Frameworks getroffen werden. Und das Risiko zur Erfüllung des geforderten Qualitätsziels minimiert sich.

5 Evaluation der Methode an Beispielen

6 Vergleich und Ausblick

6.1 Methoden Anpassung/Erweiterung

6.2 Ausblick

7 Quellenverzeichnis

[1] Tom DeMarco. „Software Engineering: An Idea Whose Time Has Come and Gone?“ In: *IEEE Software* 26 (2009), S. 96, 95. ISSN: 0740-7459. DOI: doi.ieeecomputersociety.org/10.1109/MS.2009.101.

[2] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

Todo list

Anhang

A MFEM - Vollständiger Quality Utility Tree

TODO 9: Quality Utility Tree vollständig mit allen Kategorien (mit GQM)

B CODE

TODO 10: Nothing Here