MASTERARBEIT

IST NEU IMMER BESSER?

Vergleich der Qualität von Microservice Architekturen und SOA.

ausgeführt an der

am Studiengang

Software Engineering Leadership

Von: Alexander Goldbeck

Personenkennzeichen: 1440030003

Papenburg, am … .....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

Ich möchte meinem Arbeitgeber für die gute Unterstützung des Studiums danken. Außerdem geht mein Dank an den Betreuer meiner Masterarbeit Felix Heppner, der mich stets kompetent bei der Erarbeitung dieser Arbeit unterstützt hat.

KURZFASSUNG

Durch die Verbreitung des Internets und damit der verteilten Anwendungen wurden serviceorientierte Architekturen zu einem festen Begriff der Softwareindustrie. In jüngerer Zukunft stieg das Interesse an Microservice Architekturen an und diese werden unter anderem auch mit „service orientation done right“ beschrieben.

Diese Arbeit beschäftigt sich mit der Frage „Ist neu immer besser?“ und damit mit der Frage, ob Microservice Architekturen grundsätzlich besser sind als klassische serviceorientierte Architekturen.

Zunächst werden die beiden Architekturen definiert und mit ihren Merkmalen beschrieben. Anschließend findet mit dem Architecture Tradeoff Analysis Model eine Definition einer Architekturbewertungsmethode statt. Außerdem werden Qualitätsattribute der ISO/IEC 25010 beschrieben die zur Bewertung der Softwarequalität verwendet werden kann.

In dem nächsten Schritt werden die Fähigkeiten der beiden, in dieser Arbeit untersuchten, Architekturen beschrieben, die Qualitätskriterien der ISO/IEC 25010 zu erfüllen. Nachfolgend findet eine beispielhafte Auswertung von den Anforderungen von Webanwendungstypen an diese Qualitätskriterien statt mit anschließender Gegenüberstellung zu den Eigenschaften der Architekturen.

Das Ergebnis dieser Arbeit ist, dass Microservice Architekturen grundsätzlich nicht immer besser sind als klassische serviceorientierte Architekturen und somit die Forschungsfrage mit „Nein“ beantwortet werden kann. Vielmehr sorgen die Merkmale der beiden Architekturen dafür, dass diese sich für unterschiedliche Anforderungen besonders eignen. Serviceorientierte Architekturen haben ihre Stärken beispielsweise in der Konnektivität zu anderen Systemen und den ausgereiften Technologien was vor allem die Sicherheit angeht. Microservices hingegen eignen sich sehr gut für Software mit sehr starken Skalierungs- und Performanceanforderungen.

ABSTRACT

Through the spread of the internet and thus the distributed applications, service-oriented architectures became a fixed concept of the software industry. In the recent past, interest in microservice architectures has grown as these are also described as "service orientation done right".

This thesis deals with the question "Is new always better?" and thus with the question whether microservice architectures are fundamentally better than traditional service-oriented architectures.

First, the two architectures are defined and described with their corresponding characteristics. The Architecture Tradeoff Analysis Model is then used to define a architecture evaluation method. Furthermore, quality attributes of the ISO / IEC 25010 are described which can be used for the evaluation of the software quality.

In the next step the capabilties of the two architectures to meet the quality criteria of ISO / IEC 25010 is described. Afterwards an evaluation of the requirements of sample web application types for these quality criteria is done.Then a comparison of this evaluation with the capibilities of the architectures is done.

The result of this thesis is that microservice architectures are not always better than traditional service-oriented architectures and the accordingly research question can be answered with "no".

Rather, the characteristics of the two architectures ensure that these are particularly suitable for different requirements. Service-oriented architectures for example have their strengths in the connectivity with other systems and mature technologies, especially with regard to security. Microservices, on the other hand, are very suitable for software with very strong scalability and performance requirements.

GLEICHHEITSGRUNDSATZ

Aus Gründen der Lesbarkeit wurde in dieser Arbeit darauf verzichtet, geschlechtsspezifische Formulierungen zu verwenden. Jedoch möchte ich ausdrücklich festhalten, dass die bei Personen verwendeten maskulinen Formen für beide Geschlechter zu verstehen sind.

INHALTSVERZEICHNIS

1 EINLEITUNG

Durch die stark voranschreitende Verbreitung des Internets sind immer mehr verteilte Anwendungen entstanden, die über das Internet kommunizieren. Nach (Josuttis, 2007) wurde bereits 1994 der Begriff serviceorientierte Architekturen durch Alexander Pasik verwendet, da der etablierte Begriff von Server/Client Architekturen einen zu großen Fokus auf die Hardwarekomponenten gelegt hat.

Im Jahr 2005 hat (Cearley, Fenn, & Plummer, 2005) folgende Aussage bezüglich der Verbreitung von serviceorientierten Architekturen getätigt:

„By 2008, SOA will provide the basis for 80 percent of development projects.“

(Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) beschreibt, dass über die Jahre sehr viele fehlgeschlagene Implementierungen von serviceorientierten Architekturen durchgeführt wurden und Microservices auf die ursprünglichen Prinzipien von serviceorientierten Architekturen aufbaut und unter anderem auch als „service orientation done right“ bezeichnet werden kann.

Abbildung 1 zeigt anhand der Anzahl von Suchanfragen bei Google, dass in den letzten drei Jahren das Interesse an Microservices stark angestiegen ist.

Abbildung 1: Google Trends für Begriff Microservice (Google, 2017)

Da der Begriff Microservices in der IT Industrie aktuell ein Trendthema ist befasst sich diese Arbeit mit der Frage „Ist neu immer besser?“ und vergleicht in diesem Zusammenhang die Eigenschaften von serviceorientierten Architekturen und Microservice Architekturen.

Das Ziel dieser Arbeit ist es zu evaluieren, ob Microservice Architekturen wirklich die besseren serviceorientierten Architekturen sind und generell vorzuziehen sind oder dies abhängig von den umzusetzenden Anforderungen ist.

Um diese Frage beantworten zu können werden diese beiden Architekturen anhand von Qualitätsattributen miteinander verglichen und üblichen Anforderungen von bestimmten

Anwendungstypen gegenübergestellt. So soll es möglich sein eine Entscheidungshilfe zu geben für welchen Anwendungstypen entweder serviceorientierte Architekturen oder Microservice Architekturen der bessere Ansatz sind.

In dieser Arbeit werden zunächst serviceorientierte Architekturen definiert, deren grundlegende Konzepte, Merkmale und Prinzipien beschreiben sowie die Stärken und Schwächen herausgestellt. Nachfolgend findet analog eine Beschreibung von Microservice Architekturen statt.

Anschließend wird mit der Architecture Tradeoff Analysis Method und den Attribute Based Architectural Styles eine Methode zur Bewertung von Architekturen sowie die Analyse der Auswirkung von Architekturentscheidungen auf das Gesamtsystem beschrieben.

Im Anschluss findet eine Beschreibung von Qualitätskriterien anhand der ISO/IEC 25010 statt, die die Grundlage für die anschließende Qualitätskriterienbewertung der beiden Architekturen bildet. In dieser Qualitätskriterienbewertung werden die beiden Architekturen anhand der in der ISO/IEC 25010 definierten Qualitätsattribute bewertet. Nachfolgend werden anhand von beispielhaften Anwendungstypen typische Anforderungen an die Qualitätsattribute dieser Anwendungstypen festgestellt.

Abschließend findet eine Bewertung der Eignung der beiden, in dieser Arbeit untersuchten, Softwarearchitekturen zur Erfüllung von typischen Anforderungen der Anwendungstypen statt. Zusätzlich wird im Fazit die Forschungsfrage, ob neu immer besser ist und Microservice Architekturen die besseren serviceorientierten Architekturen sind beantwortet.

2 SERVICEORIENTIERTE ARCHITEKTUR

Dieses Kapitel gibt eine allgemeine Übersicht über die grundlegenden Konzepte, den Merkmalen und Prinzipien, den erwarteten Vorteilen und den Herausforderungen von serviceorientierten Architekturen.

In Kapitel 2.1 werden zunächst die grundlegenden Konzepte und Begriffe einer SOA beschrieben und definiert. Anschließend wird in Kapitel 2.2 auf die zentralen Merkmale und Prinzipien einer SOA eingegangen, gefolgt in Kapitel 2.3 mit daraus entstandenen Erwartungen an mögliche Vorteile beim Einsatz einer SOA. Abschließend werden in Kapitel 2.4 die auftretenden Herausforderungen im Rahmen von SOA beschrieben.

2.1 Grundlegende Konzepte

Serviceorientierung wird durch (Erl, 2005) so beschrieben, dass unterschiedliche Unternehmen, oder im Zusammenhang mit der IT z.B. Teams, eigenständige Dienste anbieten, die durch eine Vielzahl an Konsumenten verwendet werden können.

Nach (Erl, 2005) beschreibt eine Architektur, die auf Serviceorientierung aufbaut, ein Modell in dem kleine, eigenständige Logikbausteine (Services) durch Automatisierungslogik zusammengesetzt ein größeres Stück Geschäftslogik abbilden können. Ein wichtiger Punkt hierbei ist, dass die einzelnen Service unabhängig voneinander existieren, und sich somit auch unabhängig voneinander weiterentwickeln, können.

Des Weiteren beschreibt (Erl, 2005), dass die Services in einer SOA über Servicebeschreibungen miteinander verbunden werden können und über Nachrichten kommunizieren.

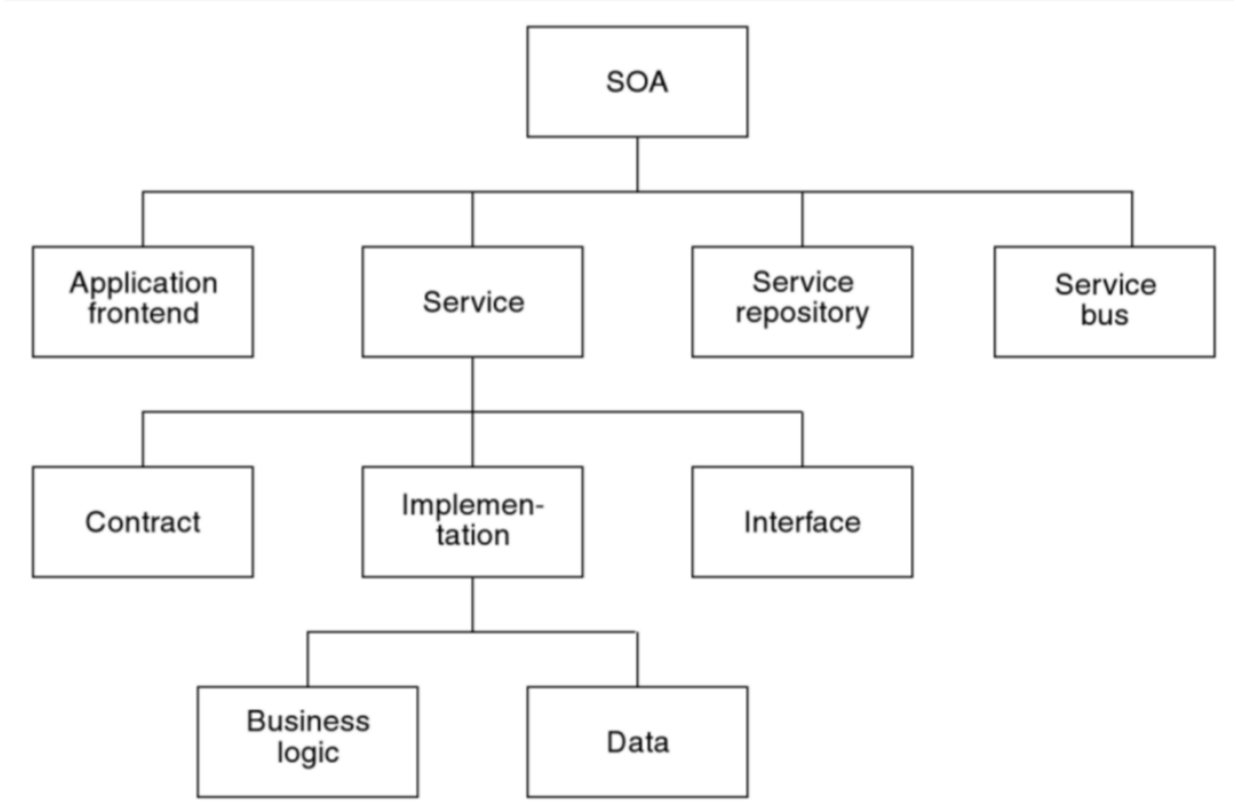

Abbildung 2: Artefakte einer SOA (Krafzig, Banke, & Slama, 2004) (S. 57)

Im Nachfolgenden werden die Kernkomponenten einer SOA detaillierter beschrieben.

2.1.1 Service

In der Literatur wird der Begriff des Services häufig sehr ähnlich definiert. (Erl, 2005) (S. 33) beschreibt Services damit, dass diese Logik in einem abgegrenzten Kontext kapseln, um unabhängig voneinander bleiben. Als mögliche Kontexte beschreibt (Erl, 2005) eine Geschäftsaufgabe, eine Geschäftseinheit oder andere logische Gruppierungseinheiten.

Nach (Krafzig, Banke, & Slama, 2004) (S. 59) wird ein Service folgendermaßen beschrieben:

> *A service is a software component auf distinctive functional meaning that typically encapsulates a high-level business concept.*

Die starke Gemeinsamkeit der beiden Definitionen ist der Fokus auf die Kapselung von Geschäftslogik. Außerdem treffen beide Autoren keine Aussagen über den Umfang oder die Größe von einzelnen Services. Die Größe von Services ist nach (Erl, 2005) davon abhängig welchen Umfang der Kontext hat, den dieser Service adressiert.

Wie in Abbildung 2 zu sehen unterteilt (Krafzig, Banke, & Slama, 2004) einen Service noch weiter auf in die Bestandteile Vertrag, Implementierung und Schnittstelle. Auf den Vertrag und die Schnittstelle eines Services wird in Kapitel 2.1.2 genauer eingegangen.

Bei der Implementierung unterscheidet (Krafzig, Banke, & Slama, 2004) noch zwischen Services die primär Geschäftslogik anbieten und solchen, die vorrangig reine Daten bereitstellen.

2.1.2 Servicebeschreibung

Um die angebotenen Funktionalitäten für Servicekonsumenten bereitzustellen, muss der Serviceprovider hierzu eine Beschreibung liefern.

Den Minimalumfang einer Servicebeschreibung definiert (Erl, 2005) folgendermaßen:

> *A service description in its most basic format establishes the name of the service and the data expected and returned by the service.*

Im Gegensatz hierzu beschreibt (Krafzig, Banke, & Slama, 2004) eine Servicebeschreibung mit den Begriffen Vertrag und Schnittstelle.

Als Vertrag nach (Krafzig, Banke, & Slama, 2004) wird eine Servicespezifikation bezeichnet, die den Verwendungszweck, die Funktionalität, die Randbedingungen sowie Informationen zur Verwendung des Services enthält.

Laut (Krafzig, Banke, & Slama, 2004) hat diese Spezifikation keine vorgegebene Form, da sich diese je nach Art des Services unterscheidet. Als optionale, aber mit Vorteilen behaftet, Form der Spezifikation sieht (Krafzig, Banke, & Slama, 2004) die Verwendung von formalen Beschreibungssprachen wie der Web Service Definition Language (WSDL). Standardisierte Spezifikationen wie die Beschreibung durch die WSDL bieten eine zusätzliche Abstraktionsschicht sowie eine Unabhängigkeit bei der Wahl der Implementierungswerkzeuge.

2.1.3 Servicekommunikation

Wie in Kapitel 2.1 beschrieben, kann über das Zusammenspiel von unabhängigen Services „höhere“ Geschäftslogik abgebildet werden. Um dieses Zusammenspiel zu ermöglichen muss eine Kommunikation der Services untereinander stattfinden.

Laut (Erl, 2005) wird für die Kommunikation ein Framework benötigt, das die lose Kopplung und damit die Unabhängigkeit der Services gewährleistet. Messaging ist ein mögliches Framework, um diese Eigenschaften zu bewahren. (Erl, 2005) beschreibt die Kommunikation innerhalb einer SOA so, dass der Service, sobald er eine Nachricht versendet hat, jegliche Kontrolle über diese verliert. Aus diesem Grund müssen Nachrichten so ausgestaltet sein, dass diese eigenständig ihren angedachten Weiterverarbeitungszweck erfüllen können. Nachrichten die diese Eigenschaften aufweisen bezeichnet (Erl, 2005) als „independent units of communication“.

Als Kommunikationsplattform bzw. Integrationsplattform beschreibt (Wolff, 2016) eine Plattform die dazu dient, dass sich die verschiedenen Services und Systeme gegenseitig aufrufen können. Die Hauptaufgabe dieser Integrationsplattform ist die sogenannte Service Orchestrierung. Dies bedeutet, dass der Nachrichtenfluss zwischen den Services koordiniert wird, um „höhere“ Geschäftslogik anzubieten. Eine konkrete Ausprägung einer solchen

Integrationsplattform beschreibt (Krafzig, Banke, & Slama, 2004) (S. 65) als Service Bus, der unter anderem folgende Fähigkeiten mitbringen sollte:

- Konnektivität: ein Service Bus bietet Fähigkeiten, um den an einer SOA beteiligten Systemen es zu ermöglichen die Funktionalität der Services aufzurufen. Dies bezeichnet (Krafzig, Banke, & Slama, 2004) als „primary purpose“ eines Services Bus.
- Heterogenität von Technologien: die Technologien der in Unternehmen eingesetzten Software gestalten sich in der Regel sehr heterogen. Aus diesem Grund muss ein Service Bus die technischen Fähigkeiten besitzen, um Systeme verschiedenster Programmiersprachen und Protokolle anbinden zu können.
- Heterogenität von Kommunikation: neben heterogenen Technologien gibt es auch unterschiedlichste Anforderungen an die Kommunikationskonzepte in einem Unternehmen. Je nach den Anforderungen der anzubindenden Anwendungen muss ein Service Bus mindestens synchrone sowie asynchrone Kommunikation unterstützten.
- Technische Dienste: abgesehen vom Herstellen der Kommunikation verschiedener Anwendungen muss ein Service Bus nach (Krafzig, Banke, & Slama, 2004) ebenfalls noch technische Dienste wie Logging, Auditing, Security, Nachrichtentransformationen und Transaktionen unterstützen.

2.1.4 Servicerepository

Dieses Kapitel beschreibt generell die Fähigkeiten sowie den Zweck und Nutzen durch die sich ein Servicerepository im Rahmen einer SOA auszeichnet. Die Inhalte dieses Kapitels richten sich grob nach (Krafzig, Banke, & Slama, 2004).

Ein Servicerepository bietet Servicekonsumenten Fähigkeiten an, um Services aufzufinden und alle Informationen bereitzustellen, die für die Verwendung der Services notwendig sind. Neben den Informationen über den in Kapitel 2.1.2 beschriebenen Servicevertrag können auch weiterführende Informationen angeboten werden. Als Beispiele hierfür nennt (Krafzig, Banke, & Slama, 2004) zum Beispiel den Ansprechpartner für einen Service, eventuelle Benutzungsgebühren oder Informationen über die verfügbaren Service Level.

Das Servicerepository ist hauptsächlich für die Nutzung innerhalb eines Unternehmens vorgesehen, da es für die Nutzung über die Unternehmensgrenzen hinaus weiterführende Anforderungen gibt. Als Beispiele hierfür nennt (Krafzig, Banke, & Slama, 2004) Anforderungen wie spezielle Sicherheitsanforderungen, Userregistrierung, Abrechnungen und Versionierung.

(Krafzig, Banke, & Slama, 2004) sieht ein Servicerepository als ein „sehr nützliches Element einer SOA“ aber nicht als eine Pflicht bei jeder SOA an. Ob der Einsatz eines Servicerepositories Sinn macht hängt von der Größe der SOA sowie der Anzahl an Teams die in der Entwicklung involviert sind ab.

Die letztendliche Implementierung eines Servicerepositories kann nahezu beliebig aussehen. Das Ausmaß der Implementierung beginnt nach (Krafzig, Banke, & Slama, 2004) bei dem einfachen Ausdrucken von Serviceverträgen bis hin zu eigens entwickelten Repositories, die

Servicedokumentationen automatisiert generieren, die den Anforderungen des Unternehmens entsprechen. Folgende Beispiele nennt (Krafzig, Banke, & Slama, 2004) als Informationen die in einem Servicerepository vorhanden sein sollten:

- Serviceinformationen wie verfügbare Methoden und deren Parameter. Diese können beispielsweise als WSDL und XSD Schema vorliegen.
- Serviceeigner auf den verschiedenen Ebenen des Unternehmens:
  - Business: Zuständig für Fragen und Änderungen an die fachliche Funktionalität.
  - Entwicklung: Zuständig für technische Fragen und Änderungen.
  - Betrieb: Zuständig für Fragen bezüglich des Servicezugriffs und Betriebsproblemen.
- Zugriffsrechte und wie diese für den Service erlangt werden können. Zudem sind auch noch die zugrundeliegenden Sicherheitsmechanismen ein Teil der Dokumentation.
- Performance- und Skalierbarkeitsinformationen in Form eines Service Level Agreement Templates.
- Transaktionale Informationen zum Service und den angebotenen Methoden. Hierzu gehören vor allem Informationen über die Idempotenz und eventuelle Kompensationsverfahren im Fehlerfall.

2.2 Merkmale / Prinzipien

Dieses Kapitel beschreibt die Designprinzipien, die nach (Erl, 2005) für das Design von Komponenten in einer SOA zu beachten sind.

- **Loose Coupling:** die Beziehung zwischen Services wird bei der losen Kopplung so beschrieben, dass das Ziel ist die Abhängigkeiten untereinander zu minimieren. Nach (Krafzig, Banke, & Slama, 2004) (S. 47) bleiben die Services, während sie die Funktionalität anderer Services verwenden, unabhängig voneinander. Dies kann durch dynamische Aufrufe der Services erreicht werden, die durch die Informationen aus den in Kapiteln 2.1.2 und 2.1.3 beschriebenen Konzepten der Servicebeschreibung- und repositories ermöglich werden.
- **Service Contract:** Services halten den in Kapitel 2.1.2 als Spezifikation beschriebenen Servicevertrag ein, um die vereinbarte Kommunikation nicht zu stören.
- **Autonomy:** die Voraussetzung für die Eigenständigkeit beschreibt (Erl, 2005) folgendermaßen: „*Autonomy requires that the range of logic exposed by a service exist within an explicit boundary.“*
- Dies hat als Folge, dass jeder Service selbstbestimmend ist und keine Abhängigkeiten zu anderen Services hat, die das Deployment oder die Weiterentwicklung des Services beeinflussen können.

- **Abstraction:** eine Abstraktion eines Services wird über die Servicebeschreibung erreicht. Abgesehen von dieser Servicebeschreibung werden keine Informationen über die Service-interne Logik veröffentlich.
- **Reusability:** die Geschäftslogik wird so in Services aufgeteilt, dass diese eine Wiederverwendung der entstehenden Services fördert.
- **Composability:** es kann eine Koordinierung und Orchestrierung von mehreren Services stattfinden, sodass hieraus ein „höherwertiger" Service entsteht. (Erl, 2005) bezeichnet solche Services als „composite services".
- **Statelessness:** (Erl, 2005) beschreibt Zustandslosigkeit so, dass ein Service die Menge der Zustandsinformationen sowie deren Vorhaltedauer minimieren sollte. Außerdem sind die Zustandsinformationen Aufruf-spezifisch was bedeutet, dass die Zustandsinformationen nur für die Dauer eines Aufrufs von einem Konsumenten vorgehalten werden.
- **Discoverability:** Services lassen sich von potentiellen Konsumenten auffinden. Dies geschieht über Mechanismen wie dem, in Kapitel 2.1.4 beschriebenen, Servicerepository.

2.3 Erwartete Vorteile

2.3.1 Bessere Integrierbarkeit

Durch den Einsatz von eigenständigen Services in einer SOA und der Verwendung dieser zur Integration von Anwendungen entsteht nach (Erl, 2005) eine Serviceorientierte Integrationsarchitektur. Ebenso wird durch den Einsatz von standardisierten Kommunikationstechnologien der Aufwand für die Integration verschiedener Anwendungen nach (Erl, 2005) deutlich reduziert.

2.3.2 Wiederverwendbarkeit

In einer SOA wird nach (Erl, 2005) das Entwerfen und Entwickeln von Services die wiederverwendbare Geschäftslogik anbieten in den Mittelpunkt gestellt. Dies bedeutet, dass bei zukünftigen Anforderungen bereits geschaffene Geschäftslogik wiederverwendet werden kann und somit nach (Erl, 2005) die Kosten für Neuentwicklungen in einer SOA immer weiter sinken.

2.3.3 Standardisierte Lösungsarchitektur

Bei der Implementierung wird nach (Erl, 2005) darauf geachtet, dass nur die benötigten Technologien Teil der Architektur werden und hier verstärkt auf Standardisierung geachtet wird. Als Beispiele für standardisierte Technologien nennt (Erl, 2005) die Verwendung der WS-* Web

Service Standards sowie der weitreichende Einsatz von XML als Datenrepräsentation in einer SOA.

Dies führt nach (Erl, 2005) zu folgenden Vorteilen:

- Die Anforderungen an die Anzahl der Fähigkeiten der Mitarbeiter sinkt.
- Die eingesetzten Technologien und Formate sind leicht verständlich.
- Es kann die Herstellerneutralität gewahrt werden, wodurch keine künstlichen Technologieeinschränkungen entstehen.

Dies führt zu einem reduzierten Aufwand und reduzierten Kosten bei der Entwicklung von Anwendungen im Umfeld einer SOA.

2.3.4 Legacy Anwendungen nutzbar machen

Bestehende Anwendungen in einem Unternehmen, sogenannte Legacy Anwendungen, können nach (Erl, 2005) befähigt werden in einer SOA verwendet werden zu können, in denen für die zu verwendende Geschäftslogik Services entwickelt werden.

Dies führt nach (Erl, 2005) dazu, dass die Kosten für die Integration dieser Anwendungen sinken und diese seltener komplett abgelöst werden müssen.

2.3.5 Zentralisierte Kommunikationsinfrastruktur

Durch den Einsatz einer zentralen Infrastrukturkomponente für die Kommunikation wie einen, in Kapitel 2.1.3 beschriebenen, Service Bus können nach (Erl, 2005) die Kosten für die Kommunikationsinfrastruktur gesenkt werden, da diese für den gesamten Unternehmenskontext verwendbar ist.

2.3.6 Organisatorische Flexibilität

Den Hauptgrund für die Einführung einer SOA beschreibt (Krafzig, Banke, & Slama, 2004) folgendermaßen:

> *„[…] the main motivation for creating an SOA is the desire to increase agility of the enterprise IT systems.“*

Diesem Grund stimmt (Erl, 2005) weitestgehend zu indem er es als einen der größten Vorteile einer SOA, dass diese durch die Annahme, dass alle entwickelten Lösungen sich weiterentwickeln werden, das Unternehmen vor unerwarteten Folgen dieser Weiterentwicklung schützt.

2.4 Herausforderungen

In diesem Kapitel werden die Herausforderungen, die bei der Entwicklung von Services im Rahmen der Einführung und Implementierung einer SOA auftreten beschrieben. In diesem Rahmen hat (Nadhan, 2004) acht Schlüsselherausforderungen identifiziert, auf die näher eingegangen wird.

2.4.1 Service Identifikation

Bei der Herausforderung der Service Identifikation geht es nach (Nadhan, 2004) um die Frage, was ein Service ist und welche Geschäftsfähigkeiten durch diesen angeboten werden. Außerdem ist die Frage der richtigen Granularität für einen Service zu klären.

Die Frage was ein Service ist beantwortet (Krafzig, Banke, & Slama, 2004) mit *„every service is an entity of distinctive functional meaning that typically encapsulates a high-level business entity.“*

Bezüglich der Granularität von Services trifft (Erl, 2005) die Aussage, dass je nach Wahl der Granularität der Services diese verschiedene Eigenschaften aufweisen. Grobgranulare Services neigen so zum Beispiel dazu eine bessere Interoperabilität zu bieten wohingegen die Wiederverwendbarkeit von feingranularen Services höher ist.

Um Services zu identifizieren und die dazugehörigen Serviceprovier zu bestimmen bietet (Nadhan, 2004) zwei Vorgehensweisen an. Zum einen lässt sich dieses Ziel über die Rationalisierung von Services erreichen und zum anderen über die Konsolidierung von Services. Die Grundannahme die (Nadhan, 2004) trifft ist, dass die benötigte Funktionalität in einem Unternehmen durch verschiedene Systeme angeboten werden kann.

Bei der **Service Rationalisierung** werden weniger häufig verwendete Services komplett durch die Funktionalitäten von häufiger benutzten Services ersetzt und somit die Gesamtzahl der Services in der SOA reduziert.

Bei der **Service Konsolidierung** werden alle Funktionalitäten der einzelnen Services zusammengefasst und über einen Service als Katalog angeboten. Der Client gibt in seinen Anfragen an welche Teilfunktionalität aus dem Katalog er verwendet möchte.

2.4.2 Service Verortung

Bei dem Einsatz von verteilten Architekturansätzen kann es nach (Nadhan, 2004) dazu kommen, dass die gleiche Geschäftsentität in verschiedenen Quellsystemen verteilt ist. Eine Datensynchronisation ist in diesen Fällen notwendig, um die Systemfunktionalität über die gesamte Unternehmensarchitektur sicherzustellen.

Auf die Frage wo Services, die diese Datensynchronisation anbieten, verortet sind stellt (Nadhan, 2004) drei Ansäte vor:

- **Content-based routing:** beim Content-based routing werden die am Service ankommenden Anfragen abhängig vom Inhalt der Anfrage, transparent für den Client, an das zutreffende Zielsystem weitergeleitet. In diesem Szenario sind die Services bei jedem Zielsystem verortet und werden dort für die jeweiligen Clients des Zielsystem angeboten.
- **Servicerepository-based routing:** analog zum Content-based routing werden bei dem Servicerepository-based routing die Anfragen abhängig vom Inhalt an die zutreffenden Zielsysteme weitergeleitet. Die Geschäftsregeln in welchen Fällen an welche Zielsysteme weitergeleitet werden soll sind im Servicerepository hinterlegt, was es ermöglicht diese einfacher und an zentraler Stelle zu ändern. Dieser Service ist zentral in der SOA verortet und bietet diese Funktionalität für alle Clients aller Zielsysteme an.
- **Back-end replication:** bei dem von (Nadhan, 2004) als „Back-end replication" bezeichneten Ansatz wird sich darauf verlassen, dass alle Zielsysteme die Services für eine Geschäftsentität anbieten für diese Entität eine eingebaute Datenreplikation anbieten. Greift ein Service auf eine Entität zu deren Daten in einem physisch fremden System liegen sorgt die eingebaute Datenreplikation des Systems dafür, dass ein Zugriff auf diese Daten möglich ist. Somit ist der Standort der Daten für die jeweiligen Services transparent. In diesem Ansatz ist kein expliziter Service für die Datenreplikation vorhanden, da sich auf die Replikationsfähigkeiten der Datenhaltungssysteme verlassen wird.

2.4.3 Service Domain Definition

Die Gruppierung von Services in logische Domänen vereinfacht durch die Reduzierung der zu beachtenden Komponenten nach (Nadhan, 2004) die Architektur eines Systems. Nach (Nadhan, 2004) gibt es drei Möglichkeiten logische Domänen zu erstellen:

- **Funktionale Domänen:** in funktionalen Domänen werden die Services über die Geschäftsfunktionen die sie anbieten gruppiert. Die Geschäftsprozessverantwortlichen können diese Domänen definieren und haben dadurch die Kontrolle über alle Services in ihrer fachlichen Domäne.
- **Technologiebasierte Domänen:** bei der Einordnung in funktionale Domänen kann es zu der Herausforderung kommen, dass sich die Technologie in den Systemen der Domäne unterschiedlich schnell weiterentwickelt und somit die Gewährleistung der Kommunikation untereinander schwieriger wird. Aus diesem Grund gibt es den Ansatz, dass Services die die gleiche Technologie verwenden in einer Domäne gruppiert werden, um nach (Nadhan, 2004) die Technologie effizient nutzen zu können.
- **Anwendungsbasierte Domänen:** der dritte Ansatz ist die Gruppierung von Services abhängig von der Anwendung, zu der die Services gehören. Dies vereinfacht nach (Nadhan, 2004) die Verwaltung und Wartung der Services, da diese alle dasselbe System verwenden.

2.4.4 Service Paketierung

Damit die Funktionen von Unternehmensanwendungen innerhalb einer SOA verwendet werden können müssen diese durch die Systeme als Service angeboten werden.

Nach (Nadhan, 2004) stellen vor allem Altsysteme, die auf Mainframe-Systemen aufbauen eine Herausforderung dar, da es sich um tief untereinander verzahnte Anwendungsteile handelt.

(Nadhan, 2004) bietet einen dreiteiligen Ansatz an, der es ermöglichen soll Altanwendungen so zu paketieren, um eigenständige Services anbieten zu können.

1. **Geschäftsbereiche definieren**: es müssen die logischen Bereiche in denen sich die Geschäftsfunktionalitäten aufteilt identifiziert werden.
2. **Programme zuweisen:** die einzelnen Programme und Anwendungsteile des Altsystems werden den identifizieren Geschäftsbereichen zugewiesen. Lässt sich keine eindeutige Zuordnung schlägt (Nadhan, 2004) vor die Programme so anzupassen, dass diese sich eindeutig zuordnen lassen.
3. **Lose Kopplung herstellen:** die Programme innerhalb der identifizieren Geschäftsbereiche sind weiterhin eng untereinander verknüpft. Hier soll durch den Einsatz von standardisierten Schnittstellen für die Kommunikation der Programme untereinander eine lose Kopplung erreicht werden. Diese lose gekoppelten Programmteile lassen sich nach (Nadhan, 2004) besser in einem serviceorientierten Architekturansatz verwenden.

2.4.5 Service Orchestrierung

Um die Anfrage eines Service Konsumenten zu bedienen, kann es nach (Nadhan, 2004) notwendig sein, dass ein Service die Funktionalitäten von anderen Services aufruft. Je nach Szenario kann es sich hierbei um ein einfaches Weiterleiten der Anfrage an einen anderen Service handeln oder aber um komplexe Anfragen mit Abhängigkeiten der Serviceaufrufe untereinander, die bis zu einem Deadlock führen können.

Damit die einzelnen involvierten Services keine Abhängigkeiten untereinander haben und somit unnötig komplex werden bietet (Nadhan, 2004) den „Business Process Management“ Ansatz an. Bei diesem Ansatz ist die Logik für die Orchestrierung der Serviceaufrufe nicht in den einzelnen Services angesiedelt, sondern auf der darüber liegenden Ebene der Geschäftsprozesse. Der Geschäftsprozess bietet den Konsumenten eine Schnittstelle an, um die fachliche Funktionalität zu nutzen und kümmert sich um die Orchestrierung der Serviceaufrufe.

2.4.6 Service Routing

Nach (Nadhan, 2004) müssen die physischen Standorte und die Verortung von Services in Domänen für den Service in einer SOA Konsumenten transparent sein. Da eine Anfrage bei

einem Servicerepository für jeden Serviceaufruf sehr zeitintensiv werden kann, bietet (Nadhan, 2004) zwei Ansätze wie trotzdem eine akzeptable Performance erzielt werden kann:

Intelligente Services: es werden alle Standortinformationen für alle Services in jeden Service mit eingebaut. Dies gewährleistet eine hochperformante Abarbeitung der Serviceanfragen, ist jedoch auch sehr wartungsintensiv und fördert eine, in einer SOA nicht gewünschte, enge Kopplung.

Router: bei dem Einsatz von Routern wird die Logik zum Bestimmen des richtigen Standortes eines Services an eine eigens dafür geschaffene Router Komponente delegiert. Nach (Nadhan, 2004) finden Router auf zwei Ebenen ihren Einsatz:

1. **Service Domäne:** auf der Ebene der Service Domäne ist es die Aufgabe der Router Komponenten bestimmen zu können welche Service Domäne die Anfrage bearbeiten kann. Kann die Anfrage in der Service Domäne der Router Komponente verarbeitet werden leitet diese die Anfrage weiter an die Router Komponente auf Service Ebene. Ist dies nicht der Fall wird die Anfrage an die Router Komponente der zuständigen Service Domäne weitergeleitet.
2. **Service:** ein Router auf Service Ebene kennt die Standorte aller Services innerhalb seiner Service Domäne und kann alle Anfragen an die entsprechenden Services weiterleiten.

2.4.7 Service Governance

In Unternehmenskontext taucht nach (Nadhan, 2004) immer die Frage auf wer Neuentwicklung oder Änderung von Services überwacht, definiert und genehmigt. Um diese Entscheidungen zu ermöglichen können interne Governance Stellen geschafften werden. (Nadhan, 2004) nennt als die zwei Ausprägungen dieser Governance Stellen zum einen die zentrale Governance Stelle und zum anderen die verteilte Governance Stelle.

Bei der **zentralen Governance Stelle** sind Mitglieder aller Service Domänen sowie Fachexperten aus den Geschäftsbereichen und Technologieexperten vertreten. Diese Governance Stelle ist als Ganzes dafür zuständig Änderungen an der Unternehmens-SOA zu genehmigen und Standards für deren Umsetzung zu definieren und zu kommunizieren.

Die **verteilte Governance Stelle** hat nach (Nadhan, 2004) als Voraussetzung, dass die in Kapitel 2.4.3 beschriebenen funktionalen Domänen als Ansatz zur Domänendefinition gewählt wurde. Bei einer verteilten Governance Stelle verantwortet jede Service Domäne die Änderungen an ihren Services selbst. Es kann bei diesem Ansatz weiterhin eine zentrale Stelle für architekturelle Vorgaben geben, diese ist jedoch nicht dafür verantwortlich Änderungen an Services zu genehmigen.

2.4.8 Service Nachrichtenformat Standard

Durch Industriestandards werden nach (Nadhan, 2004) standardisierte Nachrichtenformate vorgegeben, die das grundlegende Datenformat definieren. Auf der inhaltlichen Ebene erlauben diese Standards es jedoch so angepasst zu werden, dass sie den Anforderungen der verschiedenen Unternehmen entsprechen.

Um inhaltlich im gesamten Unternehmen ein standardisiertes Format einsetzen zu können schlägt (Nadhan, 2004) den Ansatz der „Metadata Governance“ vor. Bei diesem Ansatz werden die wichtigsten Geschäftsentitäten in einem Metadaten Repository verwaltet und gepflegt. Für die Pflege dieser Metadaten ist eine zentrale Governance Stelle zuständig, die die für das gesamte Unternehmen gültigen Metadaten einer Entität vorgibt.

3 MICROSERVICE ARCHITEKTUR

Dieses Kapitel gibt eine allgemeine Übersicht über die grundlegenden Konzepte, den Merkmalen und Prinzipien, den erwarteten Vorteilen und den Herausforderungen von Microservice Architekturen.

Es gibt nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) und (Wolff, 2016) für Microservices noch keine einheitliche oder allgemeingültige Definition, wohl jedoch grundlegende Konzepte und Merkmale die diese ausmachen.

3.1 Grundlegende Konzepte

Dieses Kapitel beschreibt die nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) grundlegenden Konzepte die Microservice Architekturen zugrunde liegen. (Wolff, 2016) sieht die Konzepte ähnlich zu (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014), wird hier jedoch nicht genauer betrachtet, da die Definition von (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) deutlich früher entstanden ist.

3.1.1 Komponentisierung durch Services

Komponenten in der Softwareentwicklung beschreibt (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) als ein Stück Software das „unabhängig ersetzt oder aktualisiert werden kann. Im Kontext von Microservices sind Programmbibliotheken Komponenten, die mit einem Programm zusammen kompiliert werden und im selben Prozess ausgeführt werden.

Eine Komponente wird durch (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) als Service bezeichnet, wenn diese in einem eigenen Prozess ausgeführt wird und die Kommunikation mit anderen Komponenten über das Netzwerk mit zum Beispiel Webservice Aufrufen stattfindet.

Services als Komponenten anzusehen haben nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) folgende Vorteile:

- **Unabhängiges Deployment:** da Services eigenständig in eignen Prozess ausgeführt werden lassen sich diese bei Aktualisierung im Gegensatz zu Programmbibliotheken unabhängig der anderen Komponenten deployen. Die Ausnahme hierzu ist eine Anpassung der angebotenen Schnittstellen, bei der jeweils auch die konsumierenden Komponenten aktualisiert werden müssen.

- **Explizite Komponentenschnittstellen:** durch die Verwendung von Netzwerkkommunikation zwischen den verschiedenen Komponenten einer Software wird ist es eine Voraussetzung, dass Schnittstellen explizit gemachen werden müssen, was nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) zwangsweise zu einer loseren Kopplung führt.

3.1.2 Organized around business capabilites

Nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) wird bei der Aufteilung von Organisationen in einzelnen Teams der Fokus durch das Management hauptsächlich auf die technischen Fähigkeiten. Die hierdurch entstehenden Teams spezialisieren sich auf die technischen Fähigkeiten ihres Teams und versuchen soweit es geht Anforderungen an die Software mit ihren technischen Lösungsmitteln umzusetzen. (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) bezeichnet dies als ein klassisches Beispiel für die Anwendung von Conway's Gesetz, das dazu führt, dass die Anwendungsarchitektur der der Organisationsstruktur entspricht.

Das Gesetz nach Conway lautet folgendermaßen:

> *Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure. (Conway, 1968)*

In Abbildung 3 ist eine Organisationsstruktur mit der dazu entstandenen Anwendungsarchitektur zu sehen.

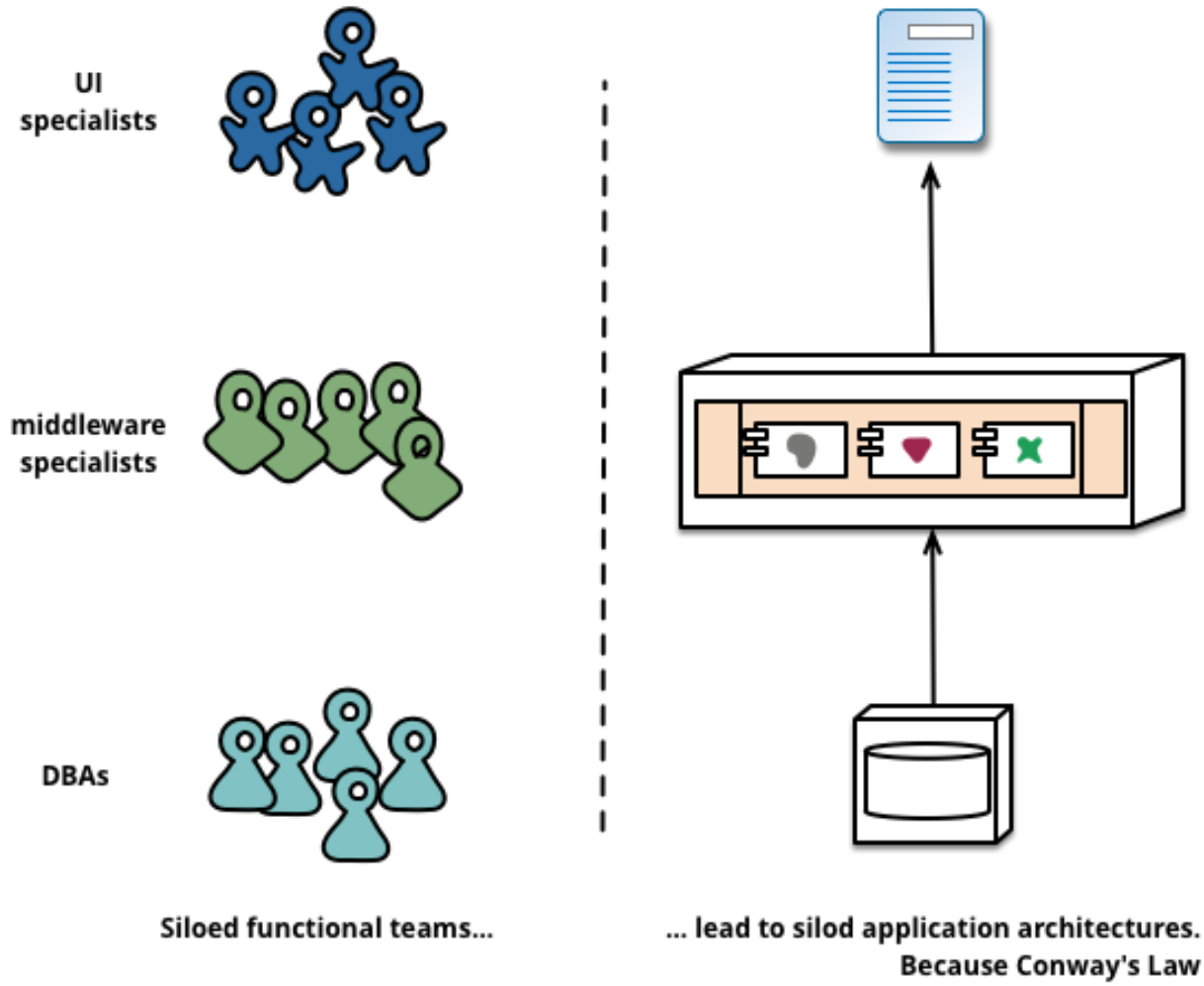

Abbildung 3: Conway's Law in action. (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014)

Bei der Entwicklung von Microservice Architekturen werden die Teams nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) an den Geschäftsbereichen ausgerichtet und sind cross-funktional aufgebaut. Cross-funktionale Teams enthalten Mitarbeiter die zusammen alle Fähigkeiten aufweisen, um ein Produkt zu entwickeln. Die Grenzen der Teams spiegeln sich nach (Conway, 1968) also auch wieder in den Grenzen der entwickelten Services wieder, wie in Abbildung 4 illustriert.

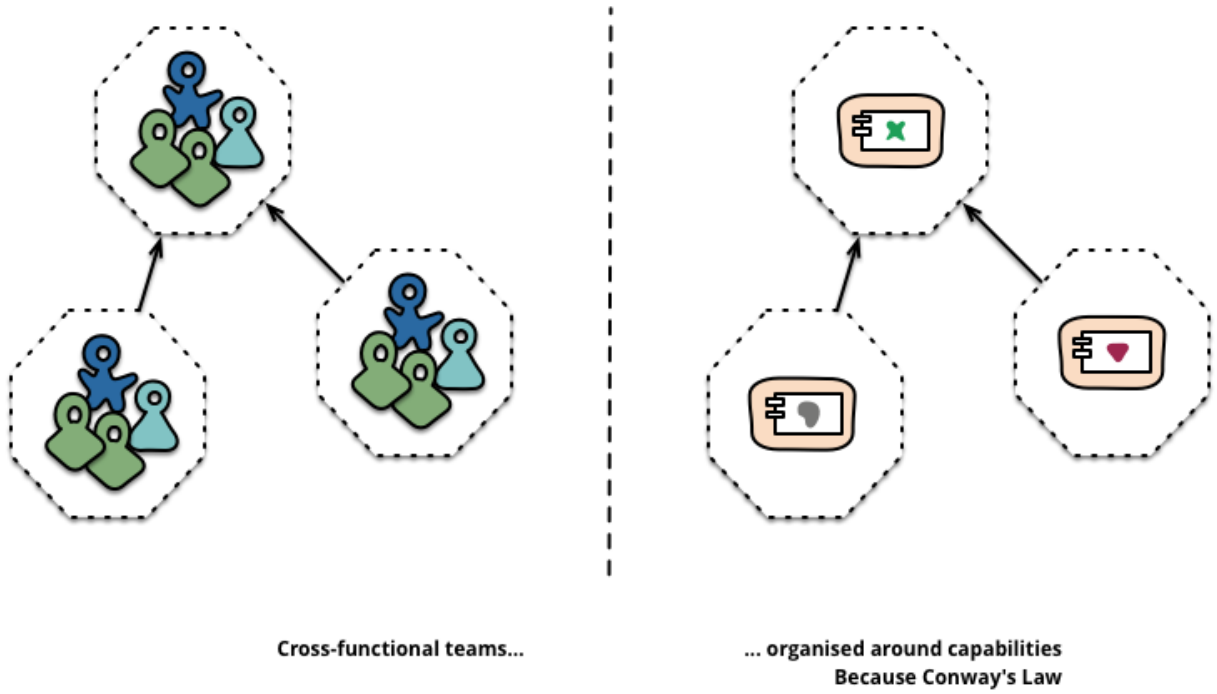

Abbildung 4: Service boundaries reinforced by team boundaries. (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014)

3.1.3 Kluge Endpunkte und dumme Pipes

Für die Kommunikation beschreibt (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) die allgemein vorherrschende Einstellung in der Microservice Community mit *„smart endpoints and dumb pipes“*.

Dieses Prinzip besagt, dass es sich bei den Kommunikationskanälen die durch Microservices verwendet werden um einfache Protokolle wie HTTP oder Messaging handelt. Dadurch, dass Microservices, wie in Kapitel 3.1.1 beschrieben, eigenständige und entkoppelte Komponenten mit klaren Schnittstellen sind muss im Kommunikationskanal keine Logik mehr ausgeführt werden, damit die Anfragen sowie Antworten verstanden werden können.

3.1.4 Dezentrale Governance

Anstatt Vorgaben bezüglich der zu verwendenden Programmbibliotheken oder Technologien zu machen bevorzugen nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) Teams bei der Umsetzung von Microservices es anderen Entwicklern bei sich erprobte, nützliche Werkzeuge zur Verfügung zu stellen. Die definierten Schnittstellen der Microservices dienen wie auch bei der SOA als Vertrag zwischen den Serviceanbietern und -konsumenten. Die Verantwortung der einzelnen Teams ihre Services zu verwalten und zu betreiben führt dazu, dass im Kontext von Microservice Architekturen dezentrale Governance zum Einsatz kommt.

3.1.5 Dezentrales Datenmanagement

Dezentrales Datenmanagement ist nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) in vielen Formen sichtbar. Als Beispiel wird angeführt, dass zwei unterschiedliche Systeme an eine Geschäftsentität unterschiedliche Anforderungen haben und somit unterschiedliche Sichten auf diese besitzen.

Microservices verwenden neben dem dezentralen Management der Datenmodelle zusätzlich auch möglicherweise noch eigene Datenbanktechnologien, die den speziellen Anforderungen an den Service entsprechen.

Bei einem dezentralen Datenmanagement gibt es besondere Anforderungen an die Bewahrung der Konsistenz der unterschiedlichen Datenquellen im Falle einer Datenaktualisierung. Als eine technische Möglichkeit die Konsistenz zu gewährleisten nennt (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) den Einsatz von Transaktionen. Verteilte Transaktionen über mehrere Datenquellen sorgen jedoch für eine zeitliche Kopplung aller involvierten Datenquellen und sind außerdem schwierig zu implementieren.

Als eine bessere Variante der Konsistenzhaltung im Kontext von Microservices bezeichnet (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) das Konzept der „eventuellen Konsistenz“.

3.1.6 Infrastrukturautomatisierung

Im Rahmen Infrastrukturautomatisierung nennt (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) zwei Arten der Automatisierung die nicht spezifisch für Microservice Architekturen sind aber bei der Implementierung dieser extensiv eingesetzt werden.

- Test- und Deploymentautomatisierung
- Infrastrukturautomatisierung in Produktion

3.1.7 Design for failure

In Bezugnahme auf die in Kapitel 3.1.1 beschriebene Verwendungen von Services als Komponenten ist es nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) notwendig, dass Anwendungen auf einen Ausfall einzelner Service Komponenten nicht mit einem Ausfall des Gesamtsystems reagieren.

Aus diesem Grund sieht (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) es als Notwendigkeit an schnell auf eventuelle Ausfälle zu reagieren und gegebenenfalls die Lauffähigkeit der ausgefallenen Services automatisiert wiederherstellen. Um dies zu gewährleisten ist ein fortgeschrittenes Echtzeit-Monitoring notwendig.

3.1.8 Evolutionary Design

Nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) haben Entwickler, die im Microservice Umfeld tätig sind, einen Hintergrund im sogenannten „evolutionary design“ und sehen in Microservices die Chance Änderungen an Software durchzuführen ohne die Entwicklungsgeschwindigkeit zu bremsen.

Die Haupteigenschaft die betrachtet werden muss, wenn die Änderbarkeit von Komponenten bewerten zu können ist nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) die Fähigkeit unabhängig von anderen Komponenten ersetzt oder erweitert zu werden.

Neben der Änderbarkeit spielt nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014) auch die Modularisierung eine große Rolle beim Design von Microservice Architekturen. Aus diesem Grund werden Funktionen die häufig zusammen geändert werden einem gemeinsamen Modul zugeordnet, was im Falle von Microservice Architekturen einem Microservice entspricht. Hierdurch wird nach (Fowler & Lewis, Microservices - a definition of this new architectural term, 2014), dass Änderungen möglichst geringe Auswirkungen haben und im Idealfall lediglich ein Microservice neu deployed werden muss.

3.2 Erwartete Vorteile

In diesem Kapitel werden die primären Vorteile die (Wolff, 2016) Microservice Architekturen zurechnet erläutert. Dies geschieht in drei Kategorien von Vorteilen:

- Technische Vorteile
- Organisatorische Vorteile
- Geschäftsvorteile

3.2.1 Technische Vorteile

3.2.1.1. Modularisierung

Wie bereits in Kapitel 3.1.1 geschieht die Kommunikation von Services in einer Microservice Architektur untereinander durch verteilte Kommunikation über das Netzwerk. Dies wird durch (Wolff, 2016) als ein Vorteil der Microservice Architektur gesehen, da hierdurch der Entwickler von Services explizit dazu gezwungen wird eine über das Netzwerk aufrufbare Schnittstelle bereitzustellen. Die Konsumenten der Services sind bei der Benutzung der Services ebenso gezwungen eine explizite Clientschnittstelle zu diesem Service bereitzustellen. Dies führt nach (Wolff, 2016) automatisch dazu, dass nur sehr schwer ungewollten Abhängigkeiten unter den Services hergestellt werden, was wiederrum zu einer starken Modularisierung der Gesamtanwendung führt.

3.2.1.2. Ersetzbarkeit

Das Ändern von vor allem alten, großen Software-Systemen stellt nach (Wolff, 2016) immer eine Herausforderung dar, da das System aufgrund seiner Größe keine einfache Übersicht über alle Funktionalitäten und die Auswirkung von Änderungen an einer Funktionalität auf das Gesamtsystem ermöglicht.

Im Gegensatz hierzu bietet die Microservice Architektur durch die geringe Größe und klare fachliche Abgrenzung der Services den Vorteil, dass diese änderbar und vor allem auch komplett ersetzbar sind. Als einen wichtigen Grund hierfür sieht (Wolff, 2016) die klare fachliche Abgrenzung die es den Entwicklern erlaubt die gesamte Fachlichkeit des Services zu kennen, was eine Grundvoraussetzung dafür ist diesen ersetzen zu können.

Dadurch, dass es sich bei den Services um eigenständig deploybare Programme handelt ist es nach (Wolff, 2016) ebenfalls kein Problem bei einer Ersetzung den kompletten Technologiestack der Anwendung auszutauschen.

3.2.1.3. Nachhaltige Entwicklung

Durch die in Kapitel 3.2.1.2 beschriebene einfache Ersetzbarkeit von Services in einer Microservice Architektur ist es nach (Wolff, 2016) möglich eine nachhaltige Entwicklung der Software zu betreiben.

Durch die zunehmende Erosion der Softwarearchitektur durch stetige Weiterentwicklung kann es nach (Wolff, 2016) dazu kommen, dass ein Service zu einer schlecht wartbaren Legacy Software wird. In diesem Fall führt die einfache Ersetzbarkeit der Services jedoch dazu, dass diese Legacy Software relativ problemlos ersetzt werden kann. In diesem Zuge merkt (Wolff, 2016) an, dass eine solche Ersetzung jedoch auch Entwicklungsaufwände nach sich zieht und so die Produktivität negativ beeinflusst.

3.2.1.4. Umgang mit Legacy Software

Als einen weiteren Vorteil von Microservice Architekturen sieht (Wolff, 2016) darin, dass diese einfach dafür verwendet werden kann Legacy Anwendungen entweder um neue Funktionalitäten zu ergänzen oder abzulösen.

Das Vorgehen zum Nutzen der Funktionalitäten der Legacy Anwendungen beschreibt (Wolff, 2016) analog zum dem Vorgehen bei Serviceorientierten Architekturen: es muss durch die Legacy Software eine Schnittstelle bereitgestellt werden über die andere Services die Funktionalitäten verwenden können. Bei diesem Vorgehen kann die Gesamtanwendung um neuen Funktionalitäten ersetzt werden ohne den Restriktionen der Legacy Anwendung zu unterliegen.

3.2.1.5. Continuous Delivery

Da es sich bei Services in einer Microservice Architektur um unabhängig voneinander deploybaren Anwendungen handelt eigenen sich diese nach (Wolff, 2016) besonders für den Einsatz in einem Continuous Delivery Umfeld. Der Umstand, dass es sich um unabhängig voneinander deploybaren Anwendungen handelt hat nach (Wolff, 2016) folgende Auswirkungen auf eine Continuous Delivery Pipeline:

- Kürzere Durchlaufzeiten der Pipeline aufgrund von kleiner Anwendungsgröße, was zu einem schnelleren Feedback über durchgeführte Änderungen führt.
- Risikoreduzierung eines Deployments durch geringere Auswirkungen von einzelnen Services auf die Gesamtanwendung.
- Geringere Anforderungen an die Testumgebung durch eine kleinere Anzahl von benötigten Drittsystemen durch jeden einzelnen Service.

Den Einsatz einer Continuous Delivery Pipeline bezeichnet (Wolff, 2016) als einen der wichtigsten Gründe warum Unternehmen auf Microservice Architekturen umsteigen. Umgekehrt sind Continuous Delivery Pipelines für den Einsatz einer Microservice Architektur eine Voraussetzung, da die Produktivsetzung von einer derart hohen Anzahl von Services wie sie in einer Microservice Architektur vorhanden sind nicht praktikabel wäre.

3.2.1.6. Skalierbarkeit

Die verteilte Kommunikation von Services in einer Microservice Architektur führt nach (Wolff, 2016) zu einer hohen Skalierbarkeit die durch folgende Mechanismen erreicht werden kann:

- Mehrere Instanzen eines Service auf unterschiedlichen Servern starten. In diesem Zuge nennt (Wolff, 2016) ebenfalls die Möglichkeit jeden Service auf einen unterschiedlich, auf die Bedürfnisse des Service, ausgelegten Server zu starten.
- Durch die häufige Verwendung von HTTP als Kommunikationsprotokoll lassen sich nach (Wolff, 2016) Caches aufgrund der guten Verfügbarkeit von HTTP-Caching Funktionalität implementieren.
- Die Instanzen von Services können weltweilt in verschiedene Netzwerkregionen deployed werden, was für zu einer geringeren Latenzzeit für international verteilte Clients führt.

3.2.1.7. Robustheit

Einen weiteren Vorteil durch den Einsatz von verteilter Kommunikation sieht (Wolff, 2016) darin, dass es dies ermöglicht ein sehr robustes und ausfallsicheres System zu entwerfen. Außerdem führen technische Probleme wie Memory Leaks durch die Ausführung in isolierten Prozessen nicht dazu, dass andere Services beeinträchtigt werden.

Um eine robuste Microservice Architektur aufzubauen müssen nach (Wolff, 2016) jedoch einige Voraussetzungen erfüllt sein:

- Ein Fehler innerhalb eines Service darf nicht zu Fehlern oder Ausfällen anderer Services führen. Die Clients müssen zum Beispiel durch Default-Verhalten oder anderweitige Kompensationen auch bei einem Ausfall bestimmter Services weiterhin eingeschränkt funktionsfähig sein.
- Es muss bewusst mit Ausfällen und Fehlern die der verteilten Kommunikation geschuldet sind gerechnet werden und entsprechendes Fehlerverhalten implementiert werden. Als Beispiel führt (Wolff, 2016) an, dass das Warten auf die Default TCP/IP-Timeouts dazu führen kann, dass das Gesamtsystem in einen Wartezustand gerät und komplett ausfällt.

3.2.1.8. Technologische Wahlfreiheit

Durch die Kommunikation über das Netzwerk in Microservice Architekturen kommt es nach (Wolff, 2016) zu einer Freiheit was die Wahl der eingesetzten Technologien für jeden Service angeht. Durch diese technologische Wahlfreiheit ergeben sich nach (Wolff, 2016) folgende Vorteile:

- Die Erprobung von neuen Technologien kann auf einen einzelnen Service begrenzt werden und somit auch das Risiko für den Einsatz deutlich reduziert werden.
- Für den jeweiligen Anwendungsfall der durch einen Service abgedeckt werden soll kann die dafür am besten geeignetste Technologie gewählt werden.

3.2.2 Organisatorische Vorteile

3.2.2.1. Selbstständigkeit von Teams

Nach dem im Kapitel 3.1.2 beschriebenen Gesetz von Conway führt die Unabhängigkeit der einzelnen Services in Microservice Architekturen ebenso dazu, dass die Teams voneinander unabhängig sind.

Dies fördert nach (Wolff, 2016) durch die Notwendigkeit die komplette Entwicklung, Wartung und Betrieb der Services dazu, dass die Selbstständigkeit der einzelnen Teams gefördert wird. Außerdem erhalten Teams die Verantwortung für die von ihnen entwickelten Services und entscheiden abgesehen von Ausnahmen wie sicherheitskritischen Entscheidungen über die Softwarearchitektur ihrer Lösungen.

3.2.2.2. Kleinere Projekte

Durch die Unabhängigkeit der Entwicklungsteams und der Services innerhalb einer Microservice Architektur können nach (Wolff, 2016) große Projekte leicht in kleiner Projekte aufgeteilt werden was folgende Vorteile mit sich bringt:

- Es entsteht weniger Kommunikations-Overhead und weniger Aufwand bei der Projektorganisation, da durch das selbstständige Arbeiten der Teams der Koordinierungsaufwand sinkt.
- Kleine Projekte erlauben es belastbarere Aufwandsschätzungen abzugeben und schlagen nach (Wolff, 2016) seltener fehl als große Projekte.

3.3 Herausforderungen

3.3.1 Technische Herausforderungen

3.3.1.1. Verteilte Systeme

Da es sich bei Microservice Architekturen um verteilte Systeme unterliegen diese ebenfalls den üblichen Problemen von über das Netzwerk kommunizierenden Anwendungen. Die Latenz ist nach (Wolff, 2016) der ist einer der zentralen Einflussfaktoren was die Performance in verteilten Systemen betrifft. Zusätzlich zu der Netzwerkstrecke kommt bei verteilten Systemen ebenfalls das Marshalling sowie Demarshalling der Serviceanfragen- und Antworten zu den die Antwortzeit beeinflussenden Faktoren.

Durch den Umstand, dass Services in einer Microservice Architektur nach (Wolff, 2016) üblicherweise alle Funktionalitäten von der UI bis zum Datenbankzugriff enthalten entfallen viele Latenzzeiten, die in üblichen Drei-Schicht Architekturen aufgetreten sind. Leidlich die Kommunikation von Services untereinander unterliegt hier den Herausforderungen von verteilten Systemen.

3.3.1.2. Code-Abhängigkeiten

Der in Kapitel 3.2.1.5 aufgeführte Vorteil des unabhängigen Deployments voneinander kann nach (Wolff, 2016) dadurch zunichte gemacht werden, wenn die Services gemeinsame Abhängigkeiten zu bestimmten Programmbibliotheken haben.

Änderungen an den Bibliotheken kann zur Folge haben, dass alle die Bibliothek verwendenden Services gleichzeitig oder zeitnah in einer bestimmten Reihenfolge produktiv gesetzt werden müssen.

Um dieser Herausforderungen zu begegnen verfolgen nach (Wolff, 2016) Microservice Architekturen das „Shared-Nothing“ Prinzip das besagt, dass Services keinerlei Codeabhängigkeiten teilen.

3.3.1.3. Unzuverlässige Kommunikation

Wie schon in Kapitel 3.3.1.1 beschrieben hat die Tatsache, dass Services über das Netzwerk kommunizieren negative Auswirkungen auf die Performance. Zusätzlich zu Performance Einschränkungen sieht (Wolff, 2016) auch noch die Gefahr des kompletten Ausfalls der Netzwerkstrecke oder den Services an sich, die potentiell die Funktionsfähigkeit der gesamten Anwendung beeinträchtigen können. Um einen Komplettausfall des Systems zu verhindern müssen alle Services Mechanismen implementieren, die es ermöglichen mit einer eingeschränkten Funktionalität weiterhin lauffähig zu sein und somit ein Teilausfall kompensiert werden kann.

3.3.1.4. Technologie-Pluralismus

In Kapitel 3.2.1.8 wurde die Möglichkeit der freien Wahl von Technologien als ein Vorteil von Microservice Architekturen beschrieben.

Nach (Wolff, 2016) ist bei der Wahl der Technologien jedoch auch zu beachten, dass eine zu vielfältige Technologielandschaft in einem Unternehmen dazu führen kann, dass es einem Team nicht mehr möglich ist alle Technologien zu überblicken. Für die Entwicklung und Applikationsbetrieb stellt dies zwar keine Herausforderung dar, da jedes Team nur die Verantwortung für die selbst entwickelten Services trägt.

Eine Vereinheitlichung in einem gewissen Rahmen macht nach (Wolff, 2016) jedoch vor allem aus der Sicht des Betriebs Sinn. Als mögliche sinnvolle Vorgaben werden zum Beispiel ein einheitliches Logging Framework oder die Verwendung von nur auf der JVM laufenden Programmiersprachen genannt.

3.3.2 Architekturelle Herausforderungen

Neben den technischen Herausforderungen sieht (Wolff, 2016) im Rahmen von Microservice Architekturen auch die Architektur selbst betreffende Herausforderungen:

- Eine automatisierte Architekturanalyse durch Codeanalysen lässt sich durch die voneinander unabhängigen Service Komponenten nur schwierig durchführen. Hierdurch wird nach (Wolff, 2016) ein Abgleich der Ist-Architektur mit der Soll-Architektur erschwert.
- Wie in Kapitel 3.1.2 beschrieben ähneln die Strukturen einer Softwarearchitektur denen der Organisation. Nach (Wolff, 2016) führt dies dazu, dass es für die Umsetzung einer Microservice Architektur notwendig ist ebenfalls Änderungen an der Organisationsstruktur vorzunehmen.

- Bei einer unsauberen Verortung der Microservices in Geschäftsbereiche kann es nach (Wolff, 2016) dazu kommen, dass neue Anforderungen gleichzeitig mehrere Teams betreffen und somit durch den notwendigen Abstimmungsaufwand einer der wesentlichen Vorteile der Microservice Architektur verloren geht.
- Das Refactoring ist nach (Wolff, 2016) sehr aufwendig, wenn Bestandteile eines Services in einen anderen Service verschoben werden müssen. Dies ist dem in Kapitel 3.3.1.2 beschriebenen „Shared-Nothing“ Prinzip geschuldet, der es notwendig machen kann, dass komplette Bibliotheksfunktionen bei einem Refactoring neu entwickelt werden müssen. Dies führt nach (Wolff, 2016) ebenfalls dazu, dass Änderungen an der Architektur des Gesamtsystems sehr schwierig werden, obwohl Änderungen an der Architektur einzelner Services sehr einfach bleiben.

3.3.3 Infrastruktur und Betrieb

Zusätzlich zu den bereits beschriebenen technischen und architekturellen Herausforderungen bei der Implementierung von Microservice Architekturen sieht (Wolff, 2016) auch noch Herausforderungen in den Bereichen der Infrastruktur und des Betriebs:

- Durch den Umstand, dass Services in Microservice Architekturen selbstständig deploybare Einheiten darstellen benötigen diese nach (Wolff, 2016) eine Vielzahl von Servern auf denen diese ausgeführt werden können. Dies erhöht die Notwendigkeit einer starken Automatisierung der Infrastruktur um einer derart hohen Zahl von virtuellen Maschinen zur Verfügung stellen zu können.
- Jede deploybare Einheit benötigt eine wie in Kapitel 3.2.1.5 beschriebene Continuous Delivery Pipeline. In diesem Umstand sieht (Wolff, 2016) eine große Herausforderung bei der Erstellung und Wartung einer derart hohen Anzahl an Pipelines.
- Das Monitoring stellt den Betrieb nach (Wolff, 2016) ebenfalls vor eine große Herausforderung, da eine sehr große Anzahl von Systemen zu überwachen ist und zusätzlich eine Überwachung von anwendungsspezifischen Parametern notwendig sein kann.
- Nach (Wolff, 2016) muss jede unabhängig deploybare Einheit ebenfalls unter eine eigenen Versionskontrolle gestellt werden. Dies führt bei der hohen Anzahl an Services in einer Microservice Architektur zu einem hohen Verwaltungsaufwand von Versionen.

4 ARCHITEKTURBEWERTUNG

Dieses Kapitel beschreibt verschiedene Möglichkeiten und Methoden Architekturbewertungen durchzuführen. Es wird auf die Architecture Tradeoff Analysis Method sowie die Attribute Based Architecture Styles eingegangen.

4.1 Architecture Tradeoff Analysis Method (ATAM)

Die Architecture Tradeoff Analysis Method (ATAM) nach (Kazman, et al., 1998) ist eine Methode zur Bewertung von Architekturdesigns die verschiedene Qualitätsattribute wie Änderbarkeit, Performance, Zuverlässigkeit und Sicherheit berücksichtigt, um die Eignung einer Softwarearchitektur für die gegebenen Anforderungen festzustellen.

Das Ziel der ATAM nach (Kazman, et al., 1998) ist es frühzeitig in einem Projekt die Auswirkungen verschiedener Anforderungen an die Qualitätsattribute einer Software zu erkennen. Aus diesem Grund beschäftigt sich die ATAM explizit mit den Abhängigkeiten zwischen verschiedenen Qualitätsattributen und deren Auswirkungen aufeinander. Diese Abhängigkeiten untereinander werden in der ATAM „tradeoff points“ genannt.

4.1.1 Die Ergebnisse

Als Ergebnis von einer nach ATAM durchgeführten Analyse kommen nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) folgende Ergebnisse zustande:

- **Eine knappe Präsentation der Architektur**: die Präsentation der Architektur im Rahmen der ATAM sollte nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) nicht länger als eine Stunde dauern und unterschiedet sich somit von der üblichen Architekturdokumentation, die Artefakte wie Objektmodelle enthält.
- **Beschreibung der Geschäftsziele**
- **Qualitätsanforderungen in der Form einer Szenariensammlung**
- **Verknüpfung von Qualitätsanforderungen mit den dazugehörigen Architekturentscheidungen**
- **Eine Sammlung von Sensitivitätspunkten und Tradeoffs**: hierbei handelt es sich nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) um Architekturentscheidungen die einen Einfluss auf die Qualitätsattribute haben. Beeinflusst eine Architekturentscheidung ein Qualitätsattribut positiv handelt es sich um

einen Sensitivitätspunkt. Führt diese Architekturentscheidung jedoch dazu, dass dafür ein anderes Qualitätsattribut negativ beeinflusst wird, handelt es sich um einen Tradeoff zwischen den beiden Qualitätsattributen.

- **Eine Sammlung von Risiken und Nicht-Risiken**: als Risiken werden nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) Architekturentscheidungen bezeichnet die unerwünschte Auswirkungen auf die Qualitätsanforderungen haben können. Als Nicht-Risiken werden Architekturentscheidungen bezeichnet die als „sicher“ eingestuft werden.
- **Eine Sammlung von Risikobereichen**: nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) werden alle identifizierten Risiken durch das Evaluierungsteam darauf untersucht, ob diese auf systematische Probleme in der Architektur hinweisen.

4.1.2 Die Phasen

Die ATAM ist nach (Kazman, et al., 1998) in vier Phasen aufgeteilt, die ähnlich dem Spiralmodell nach (Boehm, 1988) iterativ wiederholt werden, um das Architekturdesign in kritischen Bereichen genauer auszuarbeiten und dadurch Risiken zu minimieren.

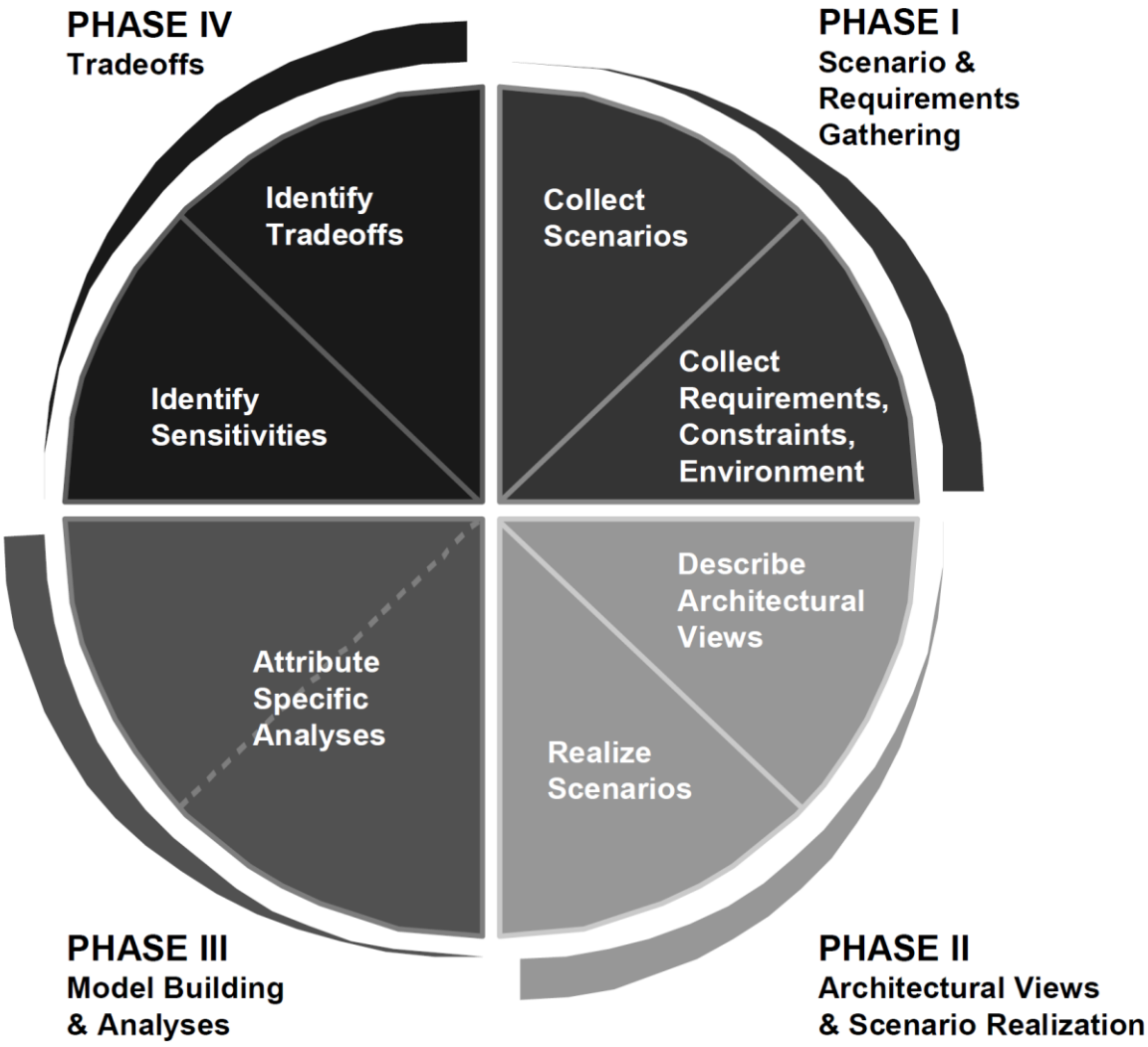

Abbildung 5: Die Phasen und dazugehörigen Schritte der ATAM nach (Kazman, et al., 1998)

In Abbildung 5 sind die vier Phasen der ATAM nach (Kazman, et al., 1998) mit den jeweils zu den Phasen gehörenden Durchführungsschritten abgebildet.

Die vier Phasen der ATAM lauten demnach wie folgt:

- **Phase 1**: Szenario- und Anforderungs-Sammlung
- **Phase 2**: Architektur-Sichten und Szenario-Umsetzung
- **Phase 3**: Modellbildung und Analyse
- **Phase 4**: Abhängigkeiten

In der späteren Literatur bleibt die ATAM weiterhin in Phasen aufgeteilt, deren Anzahl und Ausprägung sich jedoch von der ursprünglichen Definition nach (Kazman, et al., 1998) unterscheidet.

Nach (Gallagher, 2000) findet eine ATAM in zwei Phasen statt:

- **Phase 1**: es findet eine Zusammenkunft der Leiter des Evaluierungs-Teams mit den Architekten statt und es werden Vorbereitungen getroffen, um die Analyse durchzuführen.
- **Phase 2**: Durchführung der Evaluierung mit den nach (Gallagher, 2000) beschriebenen Schritten der Analyse, die in Kapitel 4.1.3 eine genauer beschrieben werden.

Eine andere Phaseneinteilung nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) sieht eine Einteilung in folgende vier Phasen vor:

- **Phase 0**: „Partnerschaft und Vorbereitung"; diese Phase entspricht der Phase 1 nach (Gallagher, 2000).
- **Phase 1**: Erste Durchführung der Evaluierung in der die Analyse mit dem Sammeln von Informationen der Entscheider im Projekt beginnt.
- **Phase 2**: Zweite Durchführung der Evaluierung in der die Analyse gemeinsam mit den Architektur-Stakeholdern weitergeführt wird.
- **Phase 3**: Nachbearbeitung der Analyse, in der das Evaluierungsteam einen schriftlichen Bericht bereitstellt und in der die Lessons-Learned der Analyse besprochen werden.

Für die weitere Beschreibung der ATAM wird die aktuellere Phaseneinteilung nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) verwendet.

4.1.3 Die Schritte

Die Analyse mit ATAM besteht nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) aus neun Durchführungsschritten. Die Schritte sind wie folgt auf die Phasen der ATAM aufgeteilt:

- **Phase 0**: keine Durchführungsschritte der Analyse.
- **Phase 1**:
  - Präsentation von ATAM
  - Präsentation der Geschäftsfaktoren
  - Präsentation der Architektur
  - Identifikation der architektonischen Ansätze
  - Erstellen des Quality Attribute Utility Tree
- **Phase 2**:
  - Brainstorming und Priorisierung der Szenarios

- Analyse der architektonischen Ansätze
- Präsentation der Ergebnisse

- **Phase 3**: keine Durchführungsschritte der Analyse

In den nachfolgenden Unterkapiteln werden zunächst die Beteiligten an der ATAM beschrieben, anschließend findet eine Beschreibung der einzelnen Durchführungsschritte der Analyse statt.

4.1.3.1. Die Beteiligten

Für die Durchführung einer ATAM Analyse sind nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) folgende Beteiligte erforderlich:

- Das Evaluierungsteam: das Evaluierungsteam ist eine unabhängige Einheit, die nicht bei der Bearbeitung des Projekts, dessen Architektur evaluiert wird, beteiligt ist. Das Team besteht nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) aus drei bis fünf Personen die folgende Rollen während der Analyse einnehmen:
  - Teamleiter: organisiert und koordiniert die Durchführung der Evaluierung beim Kunden.
  - Evaluierungsleiter: leitet und führt die Evaluierung beim Kunden durch.
  - Szenario-Schriftführer: nimmt erhobene Szenarios schriftlich auf einem Flipchart oder Whiteboard auf.
  - Vorgangs-Schriftführer: protokolliert die Durchführung der Evaluierung.
  - Zeitnehmer: achtet auf die für die Durchführung der einzelnen Schritte benötigte Zeit und hilft dem Evaluierungsleiter die vorgesehenen Zeiten einzuhalten.
  - Prozessbeobachter: beobachtet den Prozess der Evaluierung und notiert mögliche Verbesserungen und stellt diese in Phase 3 vor.
  - Prozessdurchsetzer: unterstützt den Evaluierungsleiter sich an die notwendigen Prozessschritte bei der Evaluierung zu halten.
  - Fragesteller: stellt auf die Architektur bezogene Fragen, die bei den Architekturstakeholdern noch nicht aufgetreten sind.
- Die Projektentscheider: die Projektentscheider sind Personen aus dem Projektteam die dazu befugt sind Änderungen an der Architektur zu genehmigen. Dies umfasst unter anderem folgende Personen:
  - Projektleiter
  - Budgetgenehmiger/verantwortlicher
  - Architekt

- Die Architekturstakeholder: die Stakeholder der Architektur umfassen alle Personen, deren Durchführung ihrer Arbeit durch die entstehende Architektur beeinflusst wird. Dies umfasst unter anderem Entwickler, Tester, Benutzer und auch Verantwortliche für Systeme die mit dem neuen System interagieren sollen.

4.1.3.2. Schritt 1: Präsentation von ATAM

In diesem Schritt wird die Architecture Tradeoff Analysis Method durch den Evaluierungsleiter dem gesamten Projektteam präsentiert, dies soll nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) Aspekte klären:

- Beschreibung des zu durchlaufenden Prozesses
- Offene Fragestellungen des Projektteams
- Erwartungshaltung für den Rest der Evaluierung

4.1.3.3. Schritt 2: Präsentation der Geschäftsfaktoren

Einer der Projektvertreter stellt nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) in diesem Schritt den Systemkontext und die primären Geschäftsfaktoren, die die Entwicklung des neuen Systems motiviert haben, vor. Nachfolgende Punkte sollten in der Präsentation erläutert werden:

- Die wichtigsten Funktionalitäten des Systems
- Relevante Randbedingungen (technisch, politisch oder auf Management-Ebene)
- Geschäftsziele des Projektes
- Die wichtigsten Stakeholder
- Die Anforderungen an die wichtigsten Qualitätsattribute

4.1.3.4. Schritt 3: Präsentation der Architektur

In diesem Schritt stellt der Architekt die vorgeschlagene Architektur in einer angemessenen Detailtiefe dar. Nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) ist eine angemessene Detailtiefe abhängig von dem Fortschrittsgrad des Architekturdesigns, der Dokumentation, der zur Verfügung stehenden Zeit und Anforderungen an Verhalten und Qualität.

Es sollte durch den Architekten primär auf technische Randbedingungen eingegangen werden sowie darauf mit welchen Lösungsansätzen die Anforderungen an die Qualitätsattribute erfüllt werden sollen.

4.1.3.5. Schritt 4: Identifikation der architektonischen Ansätze

Dieser Schritt stellt die nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) die logische Fortsetzung der Analyse dar, in dem die während der Präsentation der Architektur erkannten Lösungsansätze und verwendete Muster explizit identifiziert und dokumentiert werden.

4.1.3.6. Schritt 5: Erstellen des Quality Attribute Utility Tree

Die in Schritt 2 besprochenen groben Anforderungen an die Qualitätsattribute der Architektur geben nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) eine wichtige Richtung bei der Evaluierung einer Architektur vor, sind jedoch auf dem vorhandenen Detailgrad unzureichend, um bewerten zu können, ob eine Architektur die Anforderungen abdecken kann.

In diesem Schritt der Analyse werden die Qualitätsattribute mit Hilfe eines sogenannten Quality Attribute Utility Trees (QAUT) genauer aufgeschlüsselt und spezifiziert. Die Wurzel in einem QAUT stellt immer die „Brauchbarkeit“ (Utility) dar und wird weiter aufgeschlüsselt in die wichtigsten Qualitätsattribute wie Performance, Sicherheit, Änderbarkeit, Benutzbarkeit und Verfügbarkeit. Die Aufschlüsselung ist nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) nicht in festgelegte Attribute vorgegeben, sondern kann frei durch die Teilnehmer der Evaluierung definiert werden.

Die so identifizierten Qualitätsattribute werden im QAUT nun weiter aufgeschlüsselt in spezifische Attribute wie die Latenzzeit unter dem Qualitätsattribut Performance, welche wiederrum in konkrete Anforderungen aufgeschlüsselt wird wie zum Beispiel einer „Antwortzeit von unter 100 ms“.

Nachdem die Qualitätsattribute mit konkreten Anforderungen spezifiziert werden ist es nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) wichtig, dass diese durch die Projektentscheider auf Wichtigkeit für den Projekterfolg priorisiert werden, da aus Zeit- und Kostengründen nicht alle konkreten Anforderungen bei der Evaluierung berücksichtigt werden können. In einer zweiten Priorisierungsrunde werden die Qualitätsattribute durch den Architekten unter dem Gesichtspunkt der Umsetzbarkeit mit der vorgeschlagenen Architektur priorisiert.

Durch diese Bewertung kann nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) der Fokus der Evaluierung auf die Qualitätsattribute gelegt werden, die eine sehr hohe Wichtigkeit für den Projekterfolg und einen hohen Aufwand bei der Umsetzung mit der vorgeschlagenen Architektur haben.

4.1.3.7. Schritt 6: Analyse der architektonischen Ansätze

Bei der Analyse der architektonischen Ansätze stellt nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) der Architekt für die im vorherigen

Schritt als am wichtigsten priorisierten Qualitätsattribute die vorgesehenen Lösungsansätze der Architektur vor.

Für jeden besprochenen Architekturansatz werden die Risiken, Nicht-Risiken, Sensitivitätspunkte sowie Tradeoffs festgehalten. Das Ziel dieses Schrittes beschreibt (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) folgendermaßen:

> *„The goal is for the evaluation team to be convinced that the instantiation of the approach is appropriate for meeting the attribute-specific requirements for which it is intended.“*

Am Ende der Analyse sollte das Evaluierungsteam über die Verbindungen der Qualitätsattribute untereinander eine Übersicht erlangt haben sowie die wichtigsten Aspekte der Gesamtarchitektur in Bezug auf die Risiken, Nicht-Risiken, Sensitivitätspunkte sowie Tradeoffs identifiziert haben.

4.1.3.8. Schritt 7: Brainstorming und Priorisierung der Szenarios

Das Ziel des Brainstormings ist nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) die Einschätzung bezüglich der Qualitätsattribute mit denen aller Stakeholder übereinanderzulegen und ein gemeinsames Verständnis für die Architektur zu erhalten. Zusätzlich zu einem gemeinsamen Verständnis dient das Brainstorming zu identifizieren, ob während der Erstellung des Quality Attribute Utility Trees Szenarien nicht berücksichtigt wurden, die aber für die Stakeholder von Wichtigkeit sind.

In diesem Schritt können die Stakeholder außerdem die Aufmerksamkeit des Evaluierungsteams auf Szenarien lenken von denen Sie das Gefühl habe, dass diese bei der bisherigen Analyse nicht detailliert genug betrachtet wurden. Abschließend werden nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) die Stakeholder gebeten aus ihrer Sicht zusammengehörende Szenarien zu identifizieren, zusammenzuführen und abschließend zu priorisieren.

4.1.3.9. Schritt 8: Analyse der architektonischen Ansätze

Dieser Schritt ist eine erneute Iteration von Schritt 6 in dem nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) der Architekt erneut erläutert mit welchen architektonischen Ansätzen welche Anforderungen an die Qualitätsattribute abgedeckt werden. Das Hauptziel ist es die bereits in Schritt 6 analysierten Architekturansätze erneut zu präsentieren und für in Schritt 7 neu entdeckte Anforderungen an die Qualitätsattribute die Architekturansätze zu analysieren.

4.1.3.10. Schritt 9: Präsentation der Ergebnisse

In dem letzten Schritt der Analyse nach ATAM werden die ermittelten Ergebnisse präsentiert. Dies kann nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition,

2003) in mündlicher Form mit Unterstützung einer Folienpräsentation geschehen. Zusätzlich kann ein ausführlicherer schriftlicher Bericht erstellt werden.

Folgende Ergebnisse werden nach (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) im Rahmen der Präsentation gezeigt:

- Die architektonischen Ansätze
- Die ermittelten Szenarien inklusive ihrer Priorisierung aus dem Brainstorming mit den Stakeholdern
- Der Quality Attribute Utility Tree
- Die Risiken (zugeordnet zu den Geschäftsfaktoren aus Schritt 2)
- Die Nicht-Risiken
- Die Sensitivitätspunkte und Tradeoffs

4.2 Attribute Based Architectural Styles (ABAS)

In diesem Kapitel werden die Attribute Based Architectural Styles (ABAS) nach (Klein M. H., et al., 1999) beschrieben. Zunächst erfolgt eine allgemeine Beschreibung was Architekturstile und ABAS sind und welchen Nutzen die Verwendung von ABAS bringen kann. Anschließend wird dargestellt wie die Bewertung von Architekturentscheidungen durch ABAS unterstützt wird. Abschließend wird mit den Quality Attribute Models Parameters beschrieben wie Anforderungen an Qualitätsattribute messbar dargestellt werden können.

4.2.1 Beschreibung

Ein Architekturstil wird nach (Klein M. H., et al., 1999) folgendermaßen beschrieben:

> *„An architectural style includes a description of component types and their topology, a description of the pattern of data and control interaction among the components and an informal description of the benefits and drawbacks of using that style.“*

Architekturstile liefern nach (Klein M. H., et al., 1999) wichtige Erfahrungswerte zu bestimmten Arten von Designs und welche Eigenschaften diese Designs besitzen. Architekten können aus diesem Grund bei der Konzeption bestimmte Architekturstile wählen die den Qualitätszielen des Systems entsprechen.

ABAS sind Architekturstile die anstelle von Erfahrungswerten auf eine Bewertung durch auf Qualitätsattribute zugeschnittene Modelle setzen.

Ein ABAS zeichnet nach (Kazman, et al., 1998) durch folgende drei Eigenschaften aus:

1. Analog zu der Definition eines Architekturstils nach (Klein M. H., et al., 1999): *„the topology of component types and a description of the pattern of data and control interaction among the components“.*

2. Ein Qualitätsattribut-spezifisches Modell zum Schlussfolgern auf das Verhalten der Komponententypen.
3. Die Schlussfolgerung der Anwendung des Modells auf die Komponententypen.

ABAS können nach (Klein M. H., et al., 1999) für folgenden Zwecke verwendet werden:

- Die Analyse von Architekturentscheidungen und deren Auswirkungen auf das Verhalten eines Systems.
- Die Möglichkeit der Vorhersage eines erwarteten Verhaltens durch das Anwenden von Qualitätsattribut-spezifischen Modellen auf Architekturentscheidungen.

4.2.2 Architekturentscheidungen basierend auf ABAS

Beim Treffen von Architekturentscheidungen basierend auf ABAS werden nach (Klein M. H., et al., 1999) durch das Verknüpfen von Architekturstilen mit Attributsmodell-Frameworks Vorhersagen über das zukünftige Verhalten einer Architektur basierend auf Architekturentscheidungen getroffen. Der Aufbau von Entscheidungsmodellierungen durch ABAS ist in Abbildung 6 dargestellt.

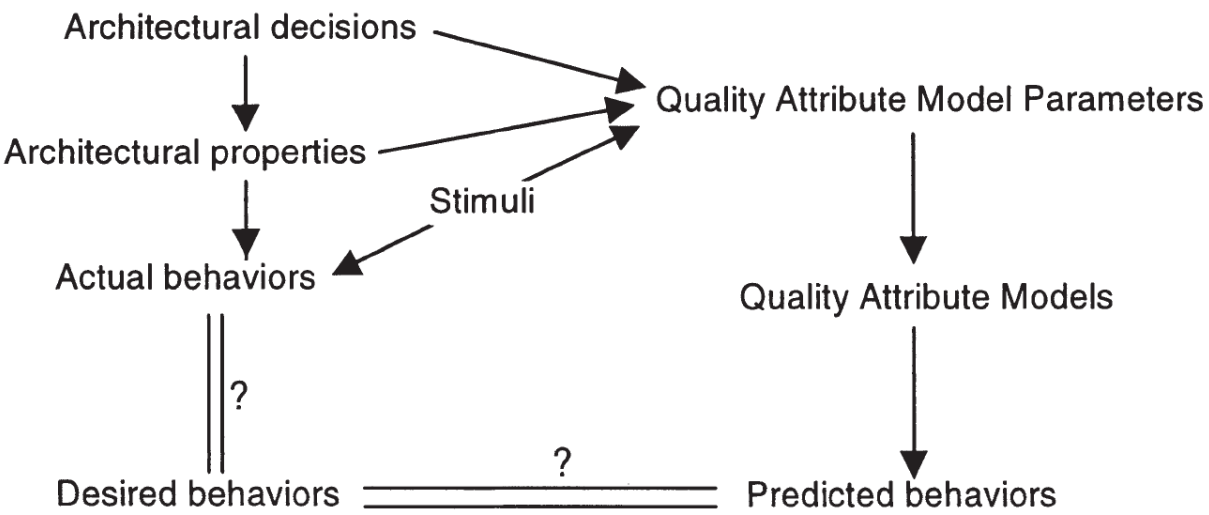

Abbildung 6: Architekturmodelle mit Attributsmodellen verknüpfen. (Klein M. H., et al., 1999)

Wie in Abbildung 6 zu sehen beeinflussen nach (Klein M. H., et al., 1999) Architekturentscheidungen die Eigenschaften einer Architektur und dadurch auch das Verhalten eines Systems.

Da über das tatsächliche Verhalten eines Systems ohne dieses zu entwickeln keine Aussagen getroffen werden können werden nach (Klein M. H., et al., 1999) Verhaltensmodelle verwendet, um das tatsächliche Verhalten des Systems zu beschreiben. Um ein Verhalten vorherzusagen müssen die Architekturmodelle mit Modellen verknüpft werden, die dazu entworfen sind über das Verhalten bezogen auf bestimmte Qualitätsattribute Vorhersagen zu treffen. Architekturentscheidungen nach (Klein M. H., et al., 1999) können demnach auf Basis eines

Vergleichs des gewünschten Verhaltens mit dem durch Modelle vorhergesagten Verhaltens getroffen werden.

Nach (Klein M. H., et al., 1999) besteht ein ABAS aus fünf Teilen:

1. **Problembeschreibung**: eine Beschreibung des Problems das durch den ABAS gelöst werden soll. Hierzu gehören nach (Klein M. H., et al., 1999) ebenfalls die relevanten Qualitätsattribute, der Kontext der Verwendung, Randbedingungen und Anforderungen an die Qualitätsattribute.
2. **Qualitätsattributs Messgrößen**: eine Beschreibung der im Rahmen der Qualitätsattribute zu messenden Ausprägungen des Attributsmodells sowie die Eingangsgrößen (*stimuli*) die Einfluss auf das Verhalten der Architektur haben.
3. **Architekturstil**: eine Beschreibung des Architekturstils wie in Kapitel 4.2.1 beschrieben.
4. **Qualitätsattributs Parameter**: eine Beschreibung der Eigenschaften des Architekturstils die einen einfach auf die Qualitätsparameter haben.
5. **Analyse**: eine Beschreibung wie die Attributsmodelle in Beziehung zu den Elementen des Architekturstils stehen. Außerdem sollte nach (Klein & Kazman, 1999) ebenfalls durch das Anwenden von Attributsmodellen gezogene Schlussfolgerungen mit in die Analyse aufgenommen werden.

4.2.3 Quality Attribute Models Parameters

Anforderungen an die Qualität einer Architektur müssen nach (Kazman, et al., 1998) durch Qualitätsattributs Messgrößen (*quality attribute measures*) beschrieben werden, um für eine Architektur bewerten zu können, ob sie diesen Anforderungen genügt.

Ob eine Architektur den Qualitätsanforderungen entspricht hängt nach (Klein M. H., et al., 1999) von den Qualitätsattributs Parametern (*quality attribute parameters*) sowie den Stimuli ab. Bei den Qualitätsattribut Parametern handelt es sich um anpassbare Aspekte der Architektur, bei den Stimuli um externe Ereignisse auf die die Architektur reagieren muss.

ABAS bilden nach (Klein M. H., et al., 1999) die Eigenschaften einer Architektur auf Qualitätsattributs Parameter ab. Diese werden dann mithilfe von Attributsmodellen auf die Stimuli abgebildet, um ein erwartetes Verhalten vorherzusagen.

5 QUALITÄTSKRITERIEN

Dieses Kapitel beschreibt die Auswirkungen von Anforderungen auf die Architektur und die daraus abgeleiteten Anforderungen an die Qualitätskriterien einer Architektur.

5.1 Auswirkungen von Anforderungen auf die Architektur

Nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) lassen sich alle Anforderungen an ein System einer von drei Kategorien zuordnen:

1. **Funktionale Anforderungen**: funktionale Anforderungen beschreiben die erforderlichen Fähigkeiten eines Systems und dessen Verhalten auf Stimuli von außen.
2. **Qualitätsanforderungen**: Qualitätsanforderungen geben vor welche qualitativen Eigenschaften die Umsetzung der funktionalen Anforderungen sowie das Gesamtsystem aufweisen müssen.
3. **Randbedingungen**: Randbedingungen sind festgelegte und nicht mehr veränderbare Architekturentscheidungen auf die der Architekt keinen Einfluss mehr nehmen kann.

5.1.1 Funktionale Anforderungen

Also Funktionalität beschreibt (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) die *„Fähigkeit des Systems die Aufgaben zu erledigen für die es vorgesehen war“*.

Die funktionalen Anforderungen bestimmen nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) nicht die Architektur, da die Funktionalität die ein System abbildet nicht abhängig von der eingesetzten Architektur ist.

Nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) führen funktionale Anforderungen dennoch durch das Zuweisen von bestimmten Funktionalitäten zu bestimmten Architekturbausteinen zu einer grundlegenden architektonischen Struktur.

5.1.2 Qualitätsanforderungen

Qualitätsanforderungen stehen nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) in enger Beziehung zu den funktionalen Anforderungen eines Systems, da sie beschreiben in welcher genauen Ausprägung und mit welchen Eigenschaften die Funktionalität umgesetzt werden soll.

Nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) gibt es drei primäre Problempunkte bei der Definition und Umsetzung von Qualitätsanforderungen:

1. Die Umsetzung der Anforderungen ist nicht testbar. Als Beispiel führt (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) das Attribut der Änderbarkeit an das je nach Art der Änderung erfüllt werden kann oder nicht.
2. Es ist nicht immer klar zuzuordnen zu welcher Qualitätsanforderungen bestimmte Aspekte eines Systems gehören, was zu Überlappungen führt und nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) das Treffen von Architekturentscheidungen erschwert.
3. Das Vokabular ist nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) übergreifend für mehrere Qualitätsattribute nicht einheitlich, sodass der gleiche Aspekt bei zwei Qualitätsattributen unterschiedlich bezeichnet wird.

Für das Lösen der Probleme der nicht-testbaren Qualitätsanforderungen sowie der Überlappungen zwischen mehrere Qualitätsanforderungen bietet (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) ein sogenanntes „Qualitätsattribut Szenario“ an, in welchem ein Qualitätsattribut spezifiziert wird. Eine genauere Beschreibung der Qualitätsattribut Szenarien ist in Kapitel 5.1.2.1 zu finden.

5.1.2.1. Qualitätsattribut Szenarien

Um Qualitätsattribute eindeutig und testbar zu definieren werden diese nach (Kazman, Clements, & Bass, Software Architecture in Practice, Third Edition, 2012) immer nach einem vorgegebenen Muster spezifiziert, das auch Qualitätsattribut Szenario genannt wird.

Folgende Angaben sind für ein Qualitätsattribut Szenario vorgegeben:

- **Stimulus**: ein Ereignis das durch einen beliebigen Auslöser an das System gegeben wird und auf das das System reagieren muss.
- **Stimulus Quelle**: der Auslöser eines Stimulus der eine Auswirkung darauf haben kann wie der Stimulus behandelt wird.
- **Umgebung**: die Umgebung beschreibt die Bedingungen unter denen das System auf einen Stimulus reagieren muss.
- **Artefakt**: der Teil des Systems (oder das Gesamtsystem) für das die Qualitätsanforderung definiert wurde.
- **Antwort**: die Antwort beschreibt das erwartete Verhalten des Systems bei der Verarbeitung des definierten Stimulus.
- **Antwort-Messgrößen**: die Messgrößen der Antwort werden dazu verwendet um zu bestimmen, ob die Antwort den Anforderungen entspricht.

Als Vorteil einer solchen Beschreibung sieht (Kazman, Bass, & Clements, Software Architecture in Practice, Second Edition, 2003) in der vereinheitlichten Beschreibung aller Qualitätsattribute

was wiederrum zu dem Nachteil führt, dass diese Beschreibung nicht für alle Qualitätsattribute optimal ist.

5.2 ISO/IEC 25010 Softwarequalitätsmodell

Das ISO/IEC 25010 Softwarequalitätsmodell nach (ISO/IEC JTC 1/SC 07 Software and systems engineering, 2005) stellt die Grundlage für die Evaluierung der Qualität von Produkten dar. Die Qualität einer Software nach (ISO/IEC JTC 1/SC 07 Software and systems engineering, 2005) wird folgendermaßen definiert:

> *„The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value. Those stakeholders' needs (functionality, performance, security, maintainability, etc.) are precisely what is represented in the quality model, which categorizes the product quality into characteristics and sub-characteristics.“*

Eine Übersicht der in dem Softwarequalitätsmodell definierten Qualitätskriterien ist in Abbildung 7 zu sehen.

Abbildung 7: ISO/IEC 25010 Softwarequalitätsmodell nach (ISO/IEC JTC 1/SC 07 Software and systems engineering, 2005).

In den nachfolgenden Kapiteln werden die einzelnen Qualitätskriterien genauer beschrieben.

5.2.1 Functional Suitability

Das Qualitätskriterium der *„functional suitablility“* beschreibt den Erfüllungsgrad der funktionalen Anforderungen durch ein System und wird durch folgende Unterkriterien genauer definiert:

- **Functional completeness**: sind alle geforderten Funktionen vorhanden?
- **Functional correctness**: liefern die implementierten Funktionen die korrekten Ergebnisse?
- **Functional appropriateness**: eignet sich das System für das Durchführen der spezifizierten Aufgaben?

5.2.2 Performance Efficiency

Das Qualitätskriterium der *„performance efficiency“* beschreibt die erreichte Performance in Relation zu den dafür benötigten Ressourcen und wird durch folgende Unterkriterien genauer definiert:

- **Time behaviour**: entsprechen die Antwortzeiten des Systems den Anforderungen?
- **Resource utilization**: entspricht der Ressourcenverbrauch des Systems den Anforderungen?
- **Capacity**: entspricht die Kapazität des Systems den Anforderungen?

5.2.3 Compatibility

Das Qualitätskriterium der *„compatibility“* beschreibt die Fähigkeiten eines Systems Informationen mit anderen Systemen auszutauschen sowie die Fähigkeit alle Funktionalitäten anzubieten, wenn die Laufzeitumgebung geteilt werden muss. Es wird durch folgende Unterkriterien genauer beschrieben:

- **Co-existence**: kann das System, wenn die Laufzeitumgebung mit anderen Systemen geteilt wird, uneingeschränkt alle Funktionalitäten anbieten ohne die anderen Systeme negativ zu beeinflussen?
- **Interoperability**: kann das System Informationen mit anderen Systemen austauschen und diese nutzen?

5.2.4 Usability

Das Qualitätskriterium *„usability“* beschreibt die Effektivität, Effizienz und Zufriedenstellung mit der ein Benutzer die definierten Aufgaben erledigten kann. Durch folgende Unterkriterien wird das Qualitätskriterium genauer beschrieben:

- **Appropriateness recognizability**: kann der Benutzer erkennen, dass das System für das Erledigen seiner Aufgaben angemessen ist?
- **Learnability**: kann der Benutzer das Arbeiten mit dem System effektiv, zufriedenstellen und ohne Risiken erlernen?
- **Operability**: ist das System einfach zu bedienen?
- **User error protection**: wird der Benutzer dafür geschützt Fehler zu machen?
- **User interface aesthetics**: ist die Benutzeroberfläche für den Benutzer ansprechend?
- **Accessibility**: ist das System für alle Benutzergruppen zugänglich?

5.2.5 Reliability

Das Qualitätskriterium der *„reliability“* beschreibt die Funktionsfähigkeit des Systems unter definierten Bedingungen über eine definierte Zeit und wird durch folgende Unterkriterien genauer beschrieben:

- **Maturity**: entspricht das System im normalen Betrieb den Anforderungen an die Zuverlässigkeit?
- **Availability**: steht das System zu den benötigten Zeiten zur Verfügung?
- **Fault tolerance**: arbeitet das System im Falle eines Fehlers oder Ausfalls wie vorgesehen?
- **Recoverability**: lässt sich das System nach einem Fehler oder Ausfall in dem gewünschten Zustand wiederherstellen?

5.2.6 Security

Das Qualitätskriterium der *„security“* beschreibt die Fähigkeit den Zugriff auf Informationen nur berechtigten Personen oder Systemen zugänglich zu machen und vor unberechtigtem Zugriff zu schützen. Durch folgende Unterkriterien wird das Qualitätskriterium genauer beschrieben:

- **Confidentiality**: haben nur Berechtigte Zugriff auf die Informationen des Systems?
- **Integrity**: verhindert das System den Zugriff auf oder das Verändern von Informationen und Programmen durch Unberechtigte?
- **Non-repudiation**: kann das System Aktionen im System nachweisen um spätere Verleugnungen zu verhindert?
- **Accountability**: können Entitäten eindeutig nachverfolgt werden?
- **Authenticity**: lassen sich die Identitäten nachweisen?

5.2.7 Maintainability

Das Qualitätskriterium der *„maintainability“* beschreibt die Fähigkeiten eines Systems korrigiert oder angepasst zu werden und wird durch folgende Unterkriterien genauer definiert:

- **Modularity**: ist das System so in Komponenten aufgeteilt, dass Änderungen an einer Komponente nur einen minimalen Einfluss auf anderen Komponenten haben?
- **Reusability**: können Teile des Systems innerhalb des Systems oder in anderen Systemen wiederverwendet werden?
- **Analysability**: können die Auswirkungen auf das System bei der Änderung von Systemkomponenten bestimmt werden?
- **Modifiability**: kann das System ohne Nebenwirkungen verändert werden?

- **Testability**: können Testkriterien zur Überprüfung des Systems bestimmt werden und regelmäßig durch Testen bestätigt werden?

5.2.8 Portability

Das Qualitätskriterium der *„portability“* beschreibt die Fähigkeit eines Systems auf andere Laufzeitumgebungen übertragen zu werden und wird durch folgende Unterkriterien genauer beschrieben:

- **Adaptability**: kann das System sich an sich ändernde Laufzeitumgebungen anpassen?
- **Installability**: kann das System in einer Umgebung installiert und deinstalliert werden?
- **Replaceability**: kann das System ein anderes System für denselben Zweck ersetzen?

6 QUALITÄTSKRITERIENBEWERTUNG VON SERVICEORIENTIERTEN ARCHITEKTUREN UND MICROSERVICE ARCHITEKTUREN

6.1 Vorgehen

Bei der Bewertung der Qualitätskriterien für die beiden in dieser Arbeit untersuchten Architekturen wird eine Mischung aus verschiedenen Bewertungsmethoden für die einzelnen Qualitätskriterien verwendet. Die Auswahl der Bewertungskriterien findet anhand der in Kapitel 5.2 beschriebenen Qualitätsmerkmale statt.

Zunächst werden die in den Kapiteln 2 und 3 beschriebenen Vor- und Nachteilen der Architekturen berücksichtigt. Zusätzlich werden die dort ermittelten Eigenschaften mit Rechercheergebnissen ergänzt.

Die so ermittelten Ergebnisse werden gemäß der in Kapitel 6.2 beschriebenen Bewertungsskala visualisiert.

6.2 Bewertungsmethode

Die Bewertung der verschiedenen Qualitätskriterien für die in dieser Arbeit untersuchten Architekturen findet anhand nachfolgend beschriebener Bewertungsmethode statt.

In Tabelle 1 sind die Bewertungsmetriken die in dieser Arbeit verwendet werden dargestellt:

Bewertungsmetrik	Punktewert
++	2
+	1
o	0
-	-1
--	-2

Tabelle 1: Bewertungsmetrik

Die Bewertung der in Kapitel 5.2 beschriebenen Qualitätskriterien für die beiden in dieser Arbeit untersuchten Architekturen kann anhand von den beiden folgenden Granularitäten stattfinden:

1. Bewertung der Subattribute wie zum Beispiel „Capacity“ unter dem übergeordneten Qualitätsattribut „Performance Efficiency“. Diese Subkriterien werden abschließend zu einer Bewertung für das übergeordnete Qualitätsattribut zusammengefasst. Wie diese Zusammenfassung stattfindet wird später in diesem Kapitel beschrieben.
2. Bewertung des übergeordneten Qualitätsattributs, falls eine Aufteilung der Bewertung in Subattribute für die Bewertung keinen Mehrwert bringt.

Das Ergebnis der Bewertungen wird mit der in Tabelle 1 in der Spalte „Bewertungsmetrik“ beschriebenen Bewertungszeichen visualisiert.

Die Bewertung des übergeordneten Attributes findet durch eine nicht gewichtete Berechnung des Durchschnittswerts der Punktewertung statt. Bei nicht geradzahligen Durchschnittswertungen findet entweder eine gemischte Bewertung durch die Bewertungszeichen statt oder eine Auf- bzw. Abrundung, falls der durchschnittliche Punktewert nicht genau zwischen zwei Bewertungen liegt.

Nachfolgend wird anhand von drei beispielhaften Bewertungen die Festlegung der Durchschnittswertung aufgezeigt.

- Tabelle 2 zeigt eine zusammengefasste Bewertung mit einem geradzahligen Durchschnitt.
- Tabelle 3 zeigt eine zusammengefasste Bewertung mit einem nicht geradzahligen Durchschnitt, der genau zwischen zwei Wertungen liegt und aus diesem Grund eine gemischte Wertung stattfindet.
- Tabelle 4 zeigt eine zusammengefasste Bewertung mit einem nicht geradzahligen Durchschnitt, der nicht genau zwischen zwei Wertungen liegt und aus diesem Grund abgerundet wird.

	Bewertung	Punkte
Subattribut 1	++	2
Subattribut 2	o	0
Übergeordnetes Qualitätsattribut	+	1

Tabelle 2: Beispielbewertung – geradzahliger Durchschnitt

	Bewertung	Punkte
Subattribut 1	++	2
Subattribut 2	+	1
Übergeordnetes Qualitätsattribut	+/++	1,5

Tabelle 3: Beispielbewertung – nicht geradzahliger Durchschnitt mit gemischter Bewertung

	Bewertung	Punkte
Subattribut 1	++	2
Subattribut 2	+	1
Subattribut 3	+	1
Übergeordnetes Qualitätsattribut	+	1,33

Tabelle 4: Beispielbewertung – nicht geradzahliger Durchschnitt mit Rundung

6.3 Auswahl der zu bewertenden Qualitätsattribute

In diesem Kapitel wird kurz beschrieben und begründet welche der in Kapitel 5.2 beschriebenen Qualitätsattribute nachfolgenden Bewertungen werden.

6.3.1 Functional Suitability

Da das Qualitätsattribut, wie in Kapitel 5.2.1 beschrieben, primär dafür dient eine bereits implementierte Software darauf zu überprüfen, ob alle geforderten Funktionen umgesetzt wurden wird dieses Qualitätsattribut nicht bewertet. Der Grund hierfür ist unter anderem, dass die Wahl der Architektur keinen grundlegenden Einfluss darauf hat, ob funktionale Anforderungen umgesetzt werden können. Diese Arbeit verfährt unter der Annahme, dass sie nur als Entscheidungshilfe für Projekte dienen kann, deren Anforderungen grundsätzlich mit den untersuchten Softwarearchitekturen umzusetzen sind.

Zusammenfassung: Es findet keine Bewertung statt.

6.3.2 Performance Efficiency

Die Performance einer Software wird grundlegend durch die verwendete Softwarearchitektur beeinflusst.

Zusammenfassung: Es findet eine Bewertung statt.

6.3.3 Compatibility

Die Fähigkeit Informationen mit anderen Systemen austauschen zu können ist in der heutigen Welt, in der die Vernetzung der Systeme untereinander immer weiter fortschreitet enorm wichtig.

Zusammenfassung: Es findet eine Bewertung statt.

6.3.4 Usability

Analog zu dem in Kapitel 6.3.1 beschriebenen Qualitätsattribut der Functional Suitability hat die Wahl der Softwarearchitektur keinen direkten Einfluss auf Möglichkeiten eine gute Usability umzusetzen.

Zusammenfassung: Es findet keine Bewertung statt.

6.3.5 Reliability

Die Wahl der Softwarearchitektur hat einen direkten und maßgeblichen Einfluss darauf mit welcher Zuverlässigkeit eine implementierte Software betrieben werden kann.

Zusammenfassung: Es findet eine Bewertung statt.

6.3.6 Security

Wie in den Kapiteln 2 und 3 beschrieben handelt es sich bei beiden in dieser Arbeit untersuchten Architekturen um verteilte Systeme, die über das Netzwerk kommunizieren. Nach (Dragoni, et al., 2017) müssen explizite Maßnahmen getroffen werden, um die Sicherheit in solchen Systemen zu gewährleisten. Diese Maßnahmen sind bei der Auswahl und dem Entwurf der Softwarearchitektur zu berücksichtigen.

Zusammenfassung: Es findet eine Bewertung statt.

6.3.7 Maintainability

Die Wartbarkeit einer Software kann nach (Eisenbarth & Koschke, 2012) einen entscheidenden Wettbewerbsvorteil darstellen, da diese eine schnelle Anpassung der Software an die Marktbedürfnisse erlaubt und eine Sicherstellung der Qualität der Software ermöglicht.

Zusammenfassung: Es findet eine Bewertung statt.

6.3.8 Portability

Die Portabilität einer Software hängt primär von der Auswahl bestimmter technischer Komponenten oder deren Hersteller ab. Da sich diese Arbeit primär mit der Bewertung von

Softwarearchitekturen und nicht deren detaillierte technische Umsetzung beschäftigt findet eine Bewertung der Portabilität nicht statt.

Zusammenfassung: Es findet keine Bewertung statt.

6.4 Performance Efficiency

In diesem Kapitel werden die Eigenschaften von softwareorientierten Architekturen und Microservice Architekturen bewertet, die einen Einfluss auf die Performance einer Software haben können.

6.4.1 Serviceorientierte Architekturen

6.4.1.1. Time behaviour

Die verteilte Natur von serviceorientierten Architekturen hat nach (O'Brien, Bass, & Merson, 2005) negativen Einfluss auf die Antwortzeiten des Systems, da die Services auf verschiedenen Maschinen betrieben werden und über das Netzwerk kommunizieren müssen.

Zusätzlich zu den Latenzzeiten bei der Netzwerkkommunikation bringt (O'Brien, Bass, & Merson, 2005) des Weiteren an, dass bei der Verwendung von in Kapitel 2.1.4 beschriebenen Servicerepositories zusätzlich zu den eigentlichen Serviceanfragen auch noch Anfragen zur Lokalisierung des Services durchgeführt werden müssen.

Durch die Verwendung der in Kapitel 2.2 beschriebenen standardisierten Technologien sowie einheitlichen Nachrichtenformaten beeinflusst nach (O'Brien, Bass, & Merson, 2005) das Marshalling und Demarshalling der Nachrichten die Performance negativ.

6.4.1.2. Resource utilization

In Kapitel 2.3.2 wird beschrieben wie die standardisierten Technologien einer serviceorientierten Architektur primär XML als Datenrepräsentation verwenden. Nach (O'Brien, Bass, & Merson, 2005) beeinflusst XML als Datenformat die Performance in serviceorientierten Architekturen negativ primär aus zwei Gründen:

1. XML Nachrichten besitzen einen großen Overhead, der dazu führt das die über das Netzwerk transportierten Nachrichten vergleichsweise groß sind. In einem Vergleich zu binären Formaten können nach (O'Brien, Bass, & Merson, 2005) XML Nachrichten zwischen 10 und 20-mal so groß sein. In einem Vergleich von JSON und XML Nachrichten beschreibt (Code Project, 2013), dass die beispielhaft verwendete XML Nachricht 84% größer ist als die äquivalente JSON Nachricht.
2. Die Verarbeitung von XML Nachrichten ist nach (O'Brien, Bass, & Merson, 2005) zusätzlich sehr CPU intensiv, da bei der Verarbeitung der Nachrichten folgende Schritte durchlaufen werden müssen:

a. Parsen: Die Textnachricht muss durch das System gelesen und in das interne Verarbeitungsformat geparst werden.

b. Validieren: Bei der Verarbeitung von XML muss dessen Format validiert werden was laut (O'Brien, Bass, & Merson, 2005) noch weiter negativ beeinflusst werden kann, wenn Document Type Definitions (DTD) oder XML Schema aufgelöst werden müssen.

c. Transformieren: Nachdem eine XML Nachricht geparst wurde muss diese in ein für den nächsten Service verarbeitetes Format transformiert werden. Nach (O'Brien, Bass, & Merson, 2005) ist dieser Vorgang bis zu 10-mal langsamer als das Parsen derselben Nachricht.

Um diese negativen Performanceeigenschaften von XML in Teilen abzuschwächen nennt (O'Brien, Bass, & Merson, 2005) einige Lösungsansätze.

Um die Größe der Nachricht, die über das Netzwerk transportiert werden muss, zu reduzieren kann diese Nachricht vor dem Versand komprimiert werden und nach Empfang durch den empfangenden Service wieder dekomprimiert werden. Die Komprimierung ist jedoch ein Tradeoff zwischen der über das Netzwerk zu versendenden Datenmenge und der benötigten CPU Ressourcen und muss je nach Ressourcenengpass entschieden werden.

In dem von (Code Project, 2013) gezeigten Vergleich von XML zu JSON Nachrichten wurde festgestellt, dass nach der Komprimierung die XML Nachricht nur noch ca. 10% größer war als die äquivalente JSON Nachricht. Hinzu kommt jedoch, dass die Komprimierung der XML Nachricht mehr als 20% mehr CPU Ressourcen in Anspruch genommen hat als die Komprimierung der JSON Nachricht.

Der zuvor beschriebenen benötigten CPU Ressourcen bei der Verarbeitung von XML Nachrichten können nach (O'Brien, Bass, & Merson, 2005) durch folgende Vorgehensweisen gemindert werden:

- Je nach Anwendungsfall das richtige XML Parsing-Modell verwenden. Bei zufälligem Zugriff innerhalb der Nachricht sollte das Document Object Model (DOM) verwendet werden bei dem die gesamte Nachricht vor der weiteren Verarbeitung komplett geparst wird. Muss die Nachricht nur sequentiell oder teilweise verarbeitet werden sollte das Simple Application Programming Interface for XML (SAX) verwendet werden, um den Ressourcenverbrauch zu optimieren.
- Die Validierung der XML Nachrichten kann deaktiviert werden, wenn garantiert werden kann, dass die versendeten Nachrichten immer ein valides Format besitzen.

6.4.1.3. Capacity

Die Verwendung von in Kapitel 2.1.4 beschriebenen Servicerepositories erlaubt es nach (O'Brien, Bass, & Merson, 2005) Services mehrfach an verschiedenen Stellen im Netzwerk zu betreiben und somit den Durchsatz des Services und damit des gesamten Systems zu erhöhen.

Die angebotenen Services können nach (O'Brien, Bass, & Merson, 2005) auf zwei Arten skaliert werden, um die Kapazität des Systems zu erhöhen:

- Horizontale Skalierung: die Last wird auf mehrere Server im Netzwerk verteilt.
- Vertikale Skalierung: die bestehenden Server werden mit leistungsstärkerer Hardware ausgestattet.

6.4.2 Microservice Architekturen

6.4.2.1. Time behaviour

Das Zeitverhalten in Microservices Architekturen ist wie schon in Kapitel x für serviceorientierte Architekturen stark davon beeinflusst, dass die Inter-Service Kommunikation über das Netzwerk stattfinden muss. Wie in Kapitel 3.3.1.1 beschrieben wird bei dem Entwurf von Microservices darauf geachtet, dass alle Funktionalitäten, die der Service benötigt seine Funktionalität anzubieten innerhalb dieses Services liegen und somit keine Netzwerkkommunikation mit anderen Services notwendig ist.

Einen zusätzlichen Fokus auf die Wichtigkeit des Designs der Services in einer Microservice Architektur legt (Fowler S. J., 2016). Folgende Designentscheidungen, die negative Auswirkungen auf die Performance von Microservice Architekturen haben, genannt:

- Ein Service muss zum Anbieten seiner Funktionalität eine hohe Anzahl von anderen Services aufrufen was mit steigender Anzahl der notwendigen Serviceaufrufe die Antwortzeiten stark in Mitleidenschaft zieht.
- Ein Service der Nachrichten und Aufgaben synchron verarbeitet, die jedoch auch durch nebenläufige und asynchrone Verarbeitung behandelt werden können, führt dazu, dass diese andere Anfragen blockieren können und somit die Antwortzeiten verschlechtern.

6.4.2.2. Resource utilization

In Kapitel 3.2.1.6 wurde bereits beschrieben, dass ein Vorteil der Aufteilung einer Anwendung in viele Services in Microservice Architekturen es ermöglicht die Serverumgebung an den speziellen Anforderungen des betriebenen Services auszulegen. Nach (Fowler S. J., 2016) muss die Hardware für eine optimale Ressourcenausnutzung ausgewählt werden. Services mit einem niedrigen CPU Verbrauch, dafür aber einem hohen Bedarf an Arbeitsspeicher sollten auf dementsprechend ausgelegt Servern betrieben werden.

6.4.2.3. Capacity

Wie bereits in Kapitel 6.4.1.3 für serviceorientierte Architekturen erlaubt es die Verteilung der Services im Netzwerk auch für Microservice Architekturen, dass diese horizontal skaliert werden können.

Eine gute horizontale Skalierbarkeit zu ermöglichen ist nach (Fowler S. J., 2016) ein entscheidendes Kriterium das einen Service in einer Microservice Architektur produktionsreif macht. Folgende Eigenschaften des Services müssen nach (Fowler S. J., 2016) bekannt sein, um diesen gut skalierbar designen zu können:

- Qualitative Skalierung: welche Maßeinheit ist ausschlaggebend für eine notwendige Skalierung? Als Beispiele hierfür nennt (Fowler S. J., 2016) die Anzahl der Aufrufe einer Seite oder die Anzahl der im System vorhandenen Kundenaufträge.
- Quantitative Skalierung: wie viele Anfragen von Clients sind pro Sekunde zu erfüllen?

Sind Services gemäß den Anforderungen an die qualitative sowie quantitative Skalierung entworfen ermöglicht es die in Kapitel 3.1.6 beschriebene Automatisierung der Infrastruktur Services in einer Microservice Architektur schnell angepasst an die aktuellen Anforderungen zu skalieren.

Als Beleg für die sehr gute Skalierbarkeit dienen unter anderem die in (Toth, 2016) beschriebenen Eckdaten der von Netflix angebotenen Dienste:

- Mehr als 600 einzelne Services im Einsatz
- Anzahl der Requests pro Tag im Milliarden Bereich
- Mehr als 10.000 Serviceinstanzen aktiv

Nachteile dieser extrem guten horizontalen Skalierbarkeit beschreibt (Toth, 2016) unter anderem mit sehr hoher Betriebskomplexität, schlechter Testbarkeit und Problemen bei der Ermittlung des aktuellen Systemstatus.

6.4.3 Bewertung Performance Efficiency

Architektur	Qualitätsattribut	Bewertung
Serviceorientierte Architekturen	Time behaviour	-
	Resource utilization	-
	Capacity	+
	Performance Efficiency (zusammengefasst)	-/o
Microservice Architekturen	Time behaviour	o
	Resource utilization	+
	Capacity	++
	Performance Efficiency (zusammengefasst)	+/++

Tabelle 5: Bewertung Qualitätsattribut Performance Efficiency

6.5 Compatibility

Dieses Kapitel beschäftigt sich mit der Bewertung den Fähigkeiten der, in dieser Arbeit untersuchten, Softwarearchitekturen Informationen mit anderen Systemen auszutauschen.

6.5.1 Serviceorientierte Architekturen

Serviceorientierte Architekturen verwenden wie in Kapitel 2.3.3 beschrieben primär in der Industrie standardisierte Technologien, was dazu führt, dass eine hohe Interoperabilität erreicht werden kann mit Systemen, die ebenfalls diese Standards unterstützen. Durch die in Kapitel 2.3.5 beschriebenen Verwendung einer zentralisierten Kommunikationsinfrastruktur wird die Interoperabilität innerhalb des Unternehmens ebenfalls erhöht.

Die Realität bei der Verwendung von Standards weicht nach (O'Brien, Bass, & Merson, 2005) jedoch von der Idee von herstellerübergreifender Kompatibilität der Schnittstellen untereinander ab. Neben den standardisierten Basistechnologien wie der Web Service Definition Language (WSDL) und dem Simple Object Access Protocol (SOAP) führen immer neue Web Service Standards dazu, dass nicht alle Plattformen die gleichen Standards implementieren. Dies wurde nach (O'Brien, Bass, & Merson, 2005) versucht durch die Einführung von Profilen durch die Web Services-Interoperability Organization (WS-I) zu verhindern. Durch die Definition von bestimmten Profilen, die vorgeben welche Web Service Standards eine Plattform unterstützt, wurde eine Basis gegeben, dass die unterstützten Standards zumindest anhand des Profils erkennbar sind.

6.5.2 Microservice Architekturen

Bei der Implementierung von Microservice Architekturen werden wie in Kapitel 3.1 beschrieben alle Services mit expliziten Komponentenschnittstellen, die über einfache Protokolle wie HTTP oder Messaging kommunizieren, umgesetzt. Dies führt zu einer hohen Interoperabilität der Microservices untereinander und ermöglicht eine einfache Anbindung von Drittsystemen.

6.5.3 Bewertung Compatibility

Architektur	Qualitätsattribut	Bewertung
Serviceorientierte Architekturen	Compatibility	++
Microservice Architekturen		+

Tabelle 6: Bewertung Qualitätsattribut Compatibility

Bei beiden Architekturen ist eine hohe Compatibility durch Definition von Standards zur Kommunikation. Serviceorientierte Architekturen setzen neben dem einfachen HTTP-Standard jedoch zusätzlich auf industrieweite Standards wie WS-*, die es erlaubt eine größere Palette von Software anzubinden.

6.6 Reliability

In diesem Kapitel werden verschiedene Merkmale der, in dieser Arbeit untersuchten, Softwarearchitekturen bewertet, die einen Einfluss auf die Zuverlässigkeit von Software haben.

6.6.1 Serviceorientierte Architekturen

Durch den in Kapitel 2.3 beschriebenen hohen Grad an Verwendung von etablierten Standardtechnologien sowie ausgereiften zentralen Kommunikationsinfrastrukturen wie einem Enterprise Service Bus können serviceorientierte Architekturen einen hohen Grad an Maturity erreichen.

Um den Nachrichtenversand in einer serviceorientierten Architektur sicherzustellen wurden nach (O'Brien, Bass, & Merson, 2005) Standards entwickelt, die herstellerübergreifend die Zuverlässigkeit für Nachrichten sicherzustellen. Der Standard WS-ReliableMessaging definiert nach (O'Brien, Bass, & Merson, 2005) folgende Garantien für die Zustellung von Nachrichten:

- in-order delivery: Es wird sichergestellt, dass Nachrichten in genau der Reihenfolge zugestellt werden in der sie versandt wurden.
- at-least-once-delivery: Es wird sichergestellt, dass eine Nachricht mindestens einmal zugestellt wird. Mehrfache Zustellungen sind jedoch möglich.

- at-most-once-delivery: Es wird sichergestellt, dass eine Nachricht maximal einmal zugestellt wird. Es ist jedoch auch möglich, dass die Nachricht gar nicht zugestellt wird.
- exactly-once-delivery: Es wird sichergestellt, dass eine Nachricht genau einmal zugestellt wird.

Durch die in Kapitel 2.1.4 beschriebenen Mechanismen eines Servicerepositories für den Lookup der zu verwendenden Services lässt sich nach (Peng & Huang, 2014) eine gute Ausfallsicherheit erreichen. Die verteilte Natur der verwendeten Services kann das Verhalten eines gesamten, durch eine serviceorientierte Architektur realisierten, Systems nach (Peng & Huang, 2014) unvorhersehbar machen. Durch die in Kapitel 2.2 beschriebene lose Kopplung und Eigenständigkeit von Services beeinflussen Fehler einzelner Services jedoch nicht direkt die Funktionalität von anderen Services, dennoch kann die Stabilität des Gesamtsystems in Mitleidenschaft gezogen werden.

Die Fehlertoleranz in serviceorientierten Architekturen lässt sich dank der in Kapitel 2.3.5 beschriebenen zentralisierten Kommunikationsinfrastruktur erhöhen. Mögliche Mechanismen sind hierfür zum Beispiel nach (Dikmans & Alves, 2016) die Verwendung von persistentem Messaging was es erlaubt, dass auch bei Ausfall von Services bereits versandte Nachrichten nicht verloren gehen, sondern bis zur Wiederherstellung des ausgefallenen Services für eine spätere Verarbeitung vorgehalten werden.

6.6.2 Microservice Architekturen

Bei der Gestaltung von Microservices Architekturen wird wie in Kapitel 3.1.7 beschrieben bereits ein Augenmerk darauf gelegt, dass das Gesamtsystem darauf ausgelegt ist bei Ausfall von Services die Funktionsfähigkeit des Systems nicht zu gefährden.

In Microservice Architekturen wird um dies zu gewährleisten verstärkt das in (Fowler M. , CircuitBreaker, 2014) beschriebene Reliability Patterns des Circuit Breakers eingesetzt, dessen einzige Aufgabe es ist zu verhindern, dass Service- oder Netzwerkausfälle sich nicht auf andere Services ausbreiten. In Abbildung 8 ist die Funktionsweise des Patterns dargestellt.

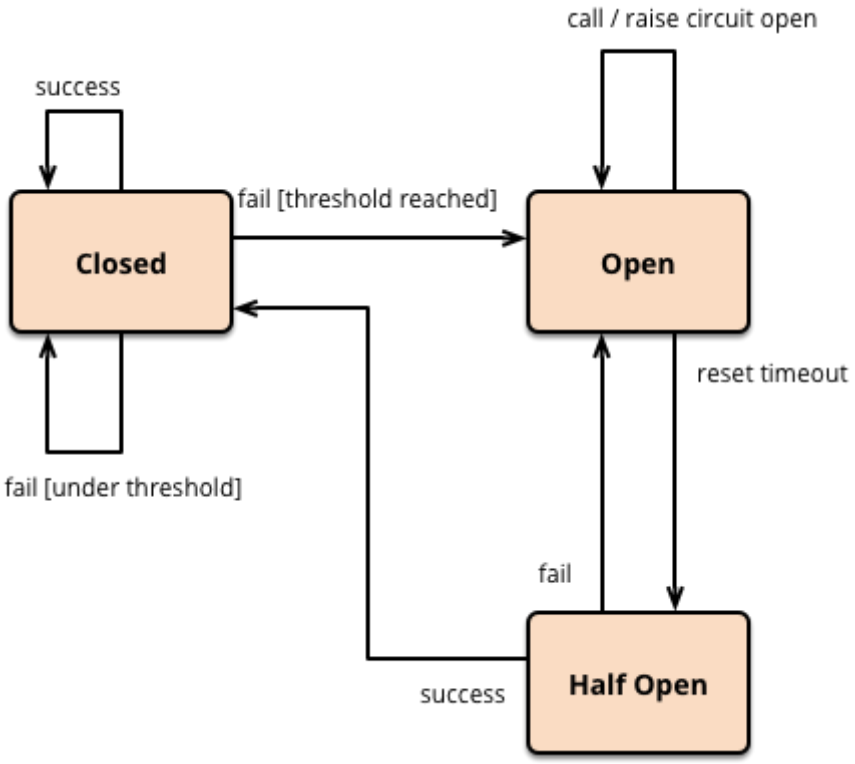

Abbildung 8: Circuit Breaker Pattern (Fowler M. , CircuitBreaker, 2014)

Grundlegend wird beim Circuit Breaker Pattern durch einen vorgeschalteten Proxy überwacht wie viele Fehler durch Verbindungsprobleme, Anwendungsfehler oder Verarbeitungszeiten auftreten. Wird ein bestimmter Threshold an Fehlern erreicht blockt der Proxy weitere Serviceanfragen ab, um das Gesamtsystem durch kaskadierende Fehler nicht zu gefährden.

Wie in Kapitel 3.2.1.6 beschrieben werden neben dem Einsatz von Pattern zur Steigerung der Fehlertoleranz ebenfalls mehrere Instanzen eines Services auf unterschiedlichen Servern verteilt. Hierdurch wird neben einer Lastverteilung ebenfalls erreicht, dass der Ausfall einzelner Serviceinstanzen durch die noch funktionsfähigen Instanzen übernommen wird und somit keine Auswirkungen auf andere Services hat.

Durch das „Design for failure“ sowie die Eigenschaften von stark verteilten Systemen können Microservice Architekturen einen sehr hohen Grad an Zuverlässigkeit erreichen.

6.6.3 Bewertung Reliability

Architektur	Qualitätsattribut	Bewertung
Serviceorientierte Architekturen	Reliability	+
Microservice Architekturen	Reliability	++

Tabelle 7: Bewertung Qualitätsattribut Reliability

Bei beiden behandelten Architekturen handelt es sich um verteilte Systeme mit unabhängig voneinander lauffähigen Services. Um eine hohe Zuverlässigkeit muss bei serviceorientierten Architekturen neben den einzelnen Services ebenfalls die zentrale Kommunikationsinfrastruktur durch Mechanismen wie Nachrichtenpersistenz hochverfügbar ausgelegt werden. Microservice Architekturen hingegen sind von Grund auf darauf ausgelegt Fehlerzustände in Services abfangen zu können, ohne andere Services zu beeinflussen. Aus diesem Grund erhalten Microservice Architekturen für die Zuverlässigkeit eine leicht höhere Wertung als serviceorientierte Architekturen.

6.7 Security

Dieses Kapitel bewertet die Fähigkeiten der untersuchten Softwarearchitekturen die Anforderungen an die Sicherheit der Software umzusetzen. In diesem Kapitel werden vorrangig beispielhafte Mechanismen beschrieben, die es der jeweiligen Softwarearchitektur erlauben diese Anforderungen umzusetzen. Eine Bewertung der einzelnen Subattribute des Qualitätsattributs Security wird in der abschließenden Bewertung aufgeschlüsselt.

6.7.1 Serviceorientierte Architekturen

Die Sicherheit in verteilten Systemen ist nach (Dragoni, et al., 2017) durch die Kommunikation über das Netzwerk der einzelnen Komponenten untereinander immer negativ beeinträchtigt.

Um diese grundlegenden Sicherheitsprobleme zu adressieren ist von (Buecker, et al., 2007) das „IBM SOA Security Reference Model“ definiert worden, das durch den Einsatz von zusätzlichen Komponenten den Sicherheitsanforderungen in verteilten Anwendungen genügen soll.

Nach (Buecker, et al., 2007) kann durch den Einsatz eines Enterprise Services Buses als zentrale Kommunikationsinfrastruktur folgende sicherheitsrelevante Aspekte garantiert werden:

- Message level security
- Confidentiality & Integrity
- Identity and Authentication
- Authorization and Privacy
- Propagation of identities between external consumer and provider environments
- Management of trust relationship between external consumer and service provider

Die Security Architektur schlägt außerdem eine logische Deployment Architektur vor, die die unterschiedlichen Aspekte der Security Anforderungen durch unterschiedliche Komponenten abdeckt. Diese Architektur ist in Abbildung 9 abgebildet.

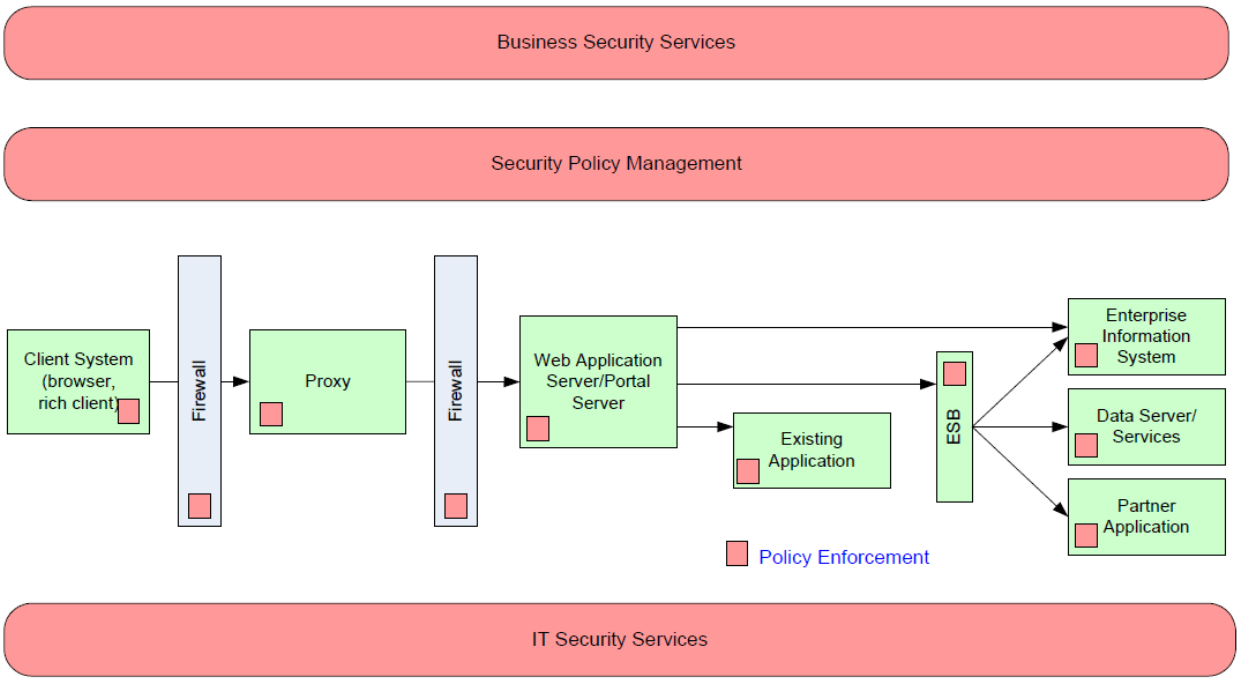

Abbildung 9: SOA Security logical architecture (Buecker, et al., 2007)

In den nachfolgenden Kapiteln werden folgende sicherheitsrelevante Komponenten aus der logischen Architektur und deren Fähigkeiten Sicherheitsanforderungen abzuducken beschrieben:

- Business Security Services
- IT Security Services
- Security Policy Management

6.7.1.1. Business Security Services

In Abbildung 10 ist eine Übersicht der in den Business Security Services enthaltenen Komponenten dargestellt.

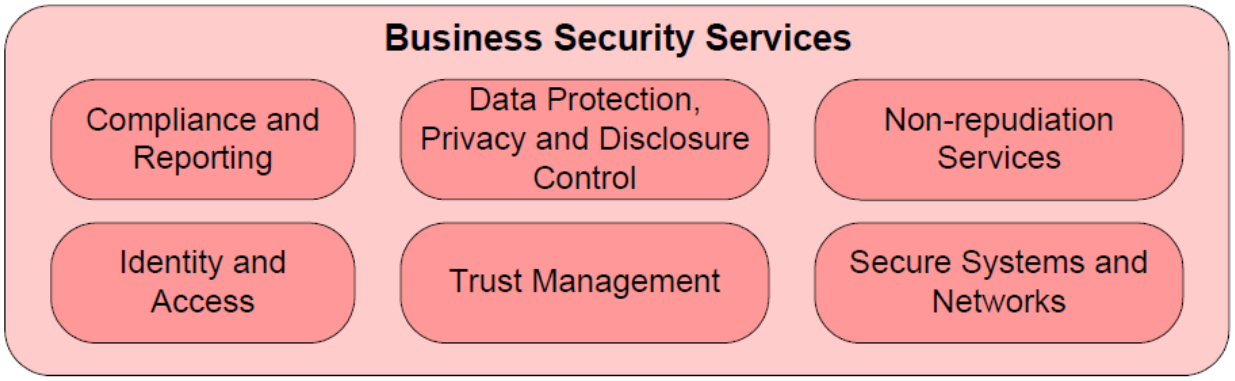

Abbildung 10: Business Security Services (Buecker, et al., 2007)

- Compliance and Reporting: das Systemverhalten wird durch das Führen eines Auditing Logs protokolliert und mit den hinterlegten Richtlinien verglichen, um Abweichungen festzustellen.
- Data Protection, Privacy and Disclosure Control: durch den Einsatz von Richtlinien für den Zugriff auf Geschäftsdaten sowie deren Verschlüsselung wird die Datenintegrität sichergestellt.
- Non-repudiation Services: durch die non-repudiation Dienste wird für die Service Clients sowie Provider sichergestellt, dass die Herkunft von Daten einwandfrei nachgewiesen werden kann.
- Identity and Access: für die Verwaltung von Systemzugriffen werden Dienste bereitgestellt, die eine Benutzerregistrierung sowie eine Verwaltung des Benutzers über den gesamten Lebenszyklus ermöglichen. Zu dem Lebenszyklus gehören nach (Buecker, et al., 2007) unter anderem die Bestätigung und die regelmäßige Validierung von Benutzerkonten.
- Trust Management: auf der technischen Ebene werden kryptografische Fähigkeiten angeboten, die notwendig sind, um eine trusted Beziehung zwischen Systemen herzustellen.
- Secure Systems and Networks: es werden eine Reihe von Technologien und Systemen eingesetzt, um das System abzusichern. Hierzu gehören nach (Buecker, et al., 2007) unter anderem der Einsatz von Firewalls, der Durchführung von sogenanntem Betriebssystem „hardening“ und Technologien für Intrusion sowie Malware Detection.

6.7.1.2. IT Security Services

In Abbildung 11 ist eine Übersicht der in den IT Security Services enthaltenen Komponenten dargestellt.

Abbildung 11: IT Security Services (Buecker, et al., 2007)

- Identity Services: die Identitäten aus verschiedensten Systemen, die Identitäten bereitstellen, werden verwaltet, geteilt und zusammengeführt.
- Authentication Services: es werden verschiedene Verfahren zur Verfügung gestellt mit denen sich am System angemeldet werden kann.
- Authorization Services: zur Sicherstellung, dass nur Benutzer in den richtigen Rollen Zugriff auf Services bekommen wird anhand des angemeldeten Benutzers und den für den Services hinterlegten Zugriffsrichtlinien entschieden, ob ein Benutzer Zugriff erlangt.
- Confidentiality Services: sicherheitsrelevante Informationen wie Verschlüsselungsschlüssen und Credentials werden durch die Confidentiality Services geschützt.
- Integrity Services: es wird geprüft, ob unerlaubte Änderungen an Daten aufgrund von Fehlern oder Sicherheitsangriffen durchgeführt wurden.
- Audit Services: es werden Audit Logs geführt, die unter anderem Logins, fehlgeschlagene Autorisierungen und Veränderungen an Sicherheitsrichtlinien nachvollziehbar protokollieren.

6.7.1.3. Security Policy Management

In Abbildung 12 ist eine Übersicht der im Security Policy Management enthaltenen Komponenten dargestellt.

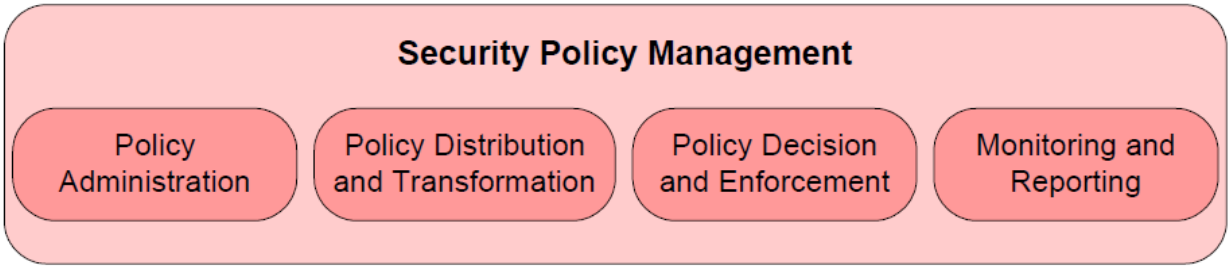

Abbildung 12: Security Policy Management (Buecker, et al., 2007)

- Policy Administration: der Lebenszyklus von Sicherheitsrichtlinien wird unterstützt.
- Policy Distribution and Transformation: die Verteilung von Sicherheitsrichtlinien an alle Clients sowie das Transformieren, in durch die Clients unterstützte Formate, werden unterstützt.
- Policy Decision and Enforcement: es werden sogenannte „policy enforcement points“ (PEP) verwendet, die dafür zuständig sind die definierten Richtlinien durchzusetzen.
- Monitoring and Reporting: durchgeführte Änderungen sowie eine Historie der Richtlinienversionen werden geführt, um eine Verantwortbarkeit für die Richtlinien herzustellen.

6.7.2 Microservice Architekturen

Analog zu den in Kapitel 6.7.1 beschriebenen Herausforderungen um den Sicherheitsanforderungen in verteilten Systemen zu genügen, müssen nach (Dragoni, et al., 2017) Maßnahmen getroffen werden, um eine Microservice Architektur abzusichern.

6.7.2.1. API Gateway

Als eine Möglichkeit dies sicherzustellen empfiehlt die Verwendung eines API Gateways, das die Services in einer Microservice Architektur von Zugriffen von Clients abkapselt und alle Anfragen entgegennimmt und an die entsprechenden Services weiterleitet.

Das API Gateway in einer Microservice Architektur nimmt nach (Moulliard, 2016), wie in Abbildung 13 zu sehen, die Requests von Clients entgegen und übernimmt die Verantwortung über die Sicherstellung der Sicherheitsanforderungen.

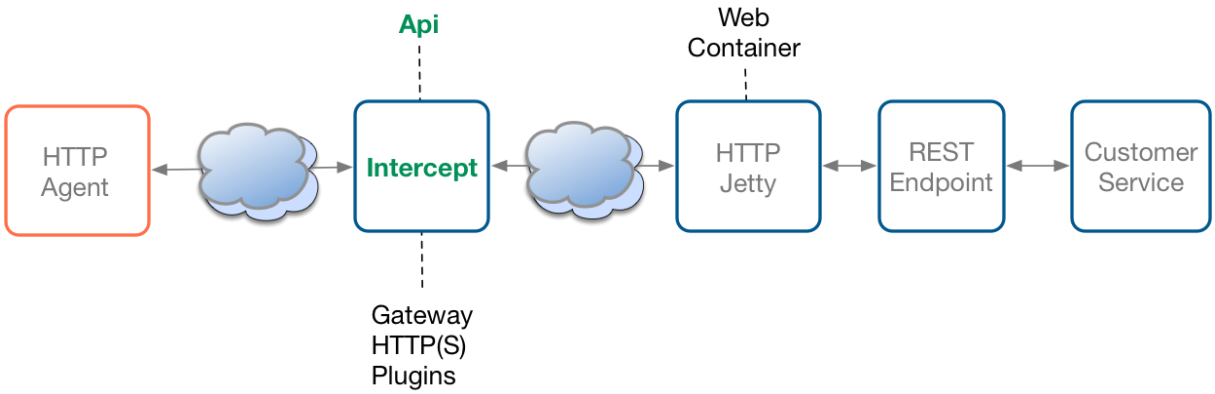

Abbildung 13: API Gateway (Moulliard, 2016)

Durch den Einsatz eines API Gateways werden nach (Apiman, 2017) unter anderem folgende Fähigkeiten zur Absicherung der Microservice Architektur unterstützt:

- Authentication: Clients müssen sich am API Gateway zum Beispiel über Credentials anmelden, um auf die Services zugreifen zu können.
- Authorization: Clients, die sich bereits am API Gateway angemeldet haben müssen zusätzlich in der richtigen Rolle sein, um auf bestimmte Services oder auch bestimmte Fähigkeiten eines Services zugreifen zu können.
- Audit Logging: Alle Metainformationen zu allen Requests können zur Sicherstellung der Nachvollziehbarkeit durch das API Gateway geloggt werden.

6.7.3 Bewertung Security

Architektur	Qualitätsattribut	Bewertung
Serviceorientierte Architekturen	Confidentiality	++
	Integrity	++
	Non-repudiation	++
	Accountability	++
	Authenticity	++
	Security (zusammengefasst)	++
Microservice Architekturen	Confidentiality	+
	Integrity	+
	Non-repudiation	+
	Accountability	+
	Authenticity	+
	Security (zusammengefasst)	+

Tabelle 8: Bewertung Qualitätsattribut Security

Sowohl serviceorientierte als auch Microservice Architekturen bieten Entwurfsmuster, Mechanismen und Produkte, um die in Kapitel 5.2.6 beschriebenen Sicherheitskriterien zu erfüllen. Das Qualitätsattribut wurde für serviceorientierte Architekturen besser bewertet, da es hierzu detaillierte Handlungsempfehlungen in Form einer Referenzarchitektur und vorhandenen Best Practices und Produkten in der Industrie gibt. Im Vergleich hierzu sind die Empfehlungen für Microservice Architekturen nur oberflächlich beschrieben.

6.8 Maintainability

Dieses Kapitel beschreibt und bewertet wie die Eigenschaften der beiden, in dieser Arbeit untersuchten, Softwarearchitekturen die Wartbarkeit einer Software beeinflussen.

6.8.1 Serviceorientierte Architekturen

6.8.1.1. Modularity

Wie in Kapitel 2.1 beschrieben ist ein grundlegendes Konzept von serviceorientierten Architekturen, dass Services eigenständige Logikbausteine sind, die unabhängig voneinander existieren und entwickelt werden können. Durch den starken Fokus auf Kapselung von zusammengehöriger Geschäftslogik und der Entkoppelung der Services über standardisierte

Servicebeschreibungen spricht für ein hohes Maß an Modularität in serviceorientierten Architekturen.

6.8.1.2. Reusability

Wie in Kapitel 2.2 beschrieben ist ein zentrales Prinzip beim Entwurf von Services in serviceorientierten Architekturen, dass Geschäftslogik so aufgeteilt wird, dass eine Wiederverwendung explizit gefördert wird.

6.8.1.3. Analysability & Modifiability

Durch die bereits bewertete Modularität von Services lassen sich die Auswirkungen von Änderungen eines Services in Regel auf den Service selbst beschränken und somit gut analysieren. Werden jedoch die in Kapitel 2.1.2 beschriebenen Servicebeschreibungen beziehungsweise Serviceverträge geändert kann es nach (O'Brien, Bass, & Merson, 2005) die Vorhersagbarkeit der Auswirkungen der Änderungen nach sich ziehen. Als Grund hierfür gibt (O'Brien, Bass, & Merson, 2005) damit an, dass nach bereits erfolgter Veröffentlichung der Servicebeschreibung nicht identifiziert werden kann, welche Clients diesen Service verwenden und somit über die geplante Änderung in Kenntnis gesetzt werden müssen.

Zusammengefasst sind Anpassungen innerhalb eines Services ohne Änderung der Servicebeschreibung in einer serviceorientierten Architektur zu einem hohen Maß möglich. Änderungen an Servicebeschreibungen sind jedoch nur sehr schlecht durchzuführen.

6.8.1.4. Testability

Durch die in den vorherigen Kapiteln bewertete gute Modularität lassen sich einzelne Services innerhalb einer serviceorientierten Architektur gut isoliert testen.

Die Testbarkeit des gesamten Systems kann nach (O'Brien, Bass, & Merson, 2005) aus verschiedenen Gründen schnell sehr komplex werden. Folgende Gründe gibt (O'Brien, Bass, & Merson, 2005) als problematisch für die Testbarkeit einer serviceorientierten Architektur an:

- Zum Testen kann die Interaktion zwischen verschiedenen, im Netzwerk verteilten Services notwendig sein.
- Werden Services verwendet, die durch externe Organisationen bereitgestellt werden können aufgrund des fehlenden Zugriffs auf den Quellcode nicht zuverlässig der gesamte notwendige Umfang an Testfällen identifiziert werden.
- Durch die Verwendung von in Kapitel 2.1.4 beschriebenen Servicerepositories kann es dazu kommen, dass erst zur Laufzeit der Anwendung bestimmt werden kann welche Services Verwendung finden. Dies stellt nach (O'Brien, Bass, & Merson, 2005) eine große Herausforderung bei der Durchführung von automatisierten und wiederholbaren Tests dar.

Neben den Problemen die beim Entwerfen und Durchführen von Tests bei serviceorientierten Architekturen vorliegen führt (O'Brien, Bass, & Merson, 2005) ebenfalls Herausforderungen bei der Fehleranalyse, von bei Tests aufgetretenen Fehlern, sowie der Reproduzierbarkeit der Fehler auf. Der Grund hierfür ist, dass ein Fehler nach (O'Brien, Bass, & Merson, 2005) unter anderem Fehlerquellen in folgenden Bestandteilen des Systems haben könnte:

- In der Anwendung selbst, die die Services verwendet.
- In einem der Services, der durch die Anwendung verwendet wird.
- In der Infrastruktur die entweder durch die Anwendung oder den verwendeten Services benutzt wird.
- Lastsituationen in den Systemen, die die Services bereitstellen.
- Bei der Komponente die die Services über das Servicerepository lokalisiert.

Zusammengefasst bedeutet dies, trotz der guten Testbarkeit einzelner Services, dass mit einer serviceorientierten Architektur umgesetzte Systeme nur eine schlechte Testbarkeit bieten.

6.8.2 Microservice Architekturen

6.8.2.1. Modularity

Die Modularisierung ist wie in Kapitel 3.2.1.1 einer der zentralen Vorteile bei der Verwendung von Microservice Architekturen. Auch die in Kapitel (Performance) beschriebene Notwendigkeit Services so zu modularisieren, dass möglichst wenig Kommunikation zwischen den Services stattfinden muss führt zu einer sehr hohen Modularität in Microservice Architekturen.

6.8.2.2. Reusability

Die in Kapitel 3.2.1.2 beschriebene Ersetzbarkeit, die ein zentrales Paradigma in Microservice Architekturen darstellt steht im Gegensatz zu der Anforderung an Wiederverwendbarkeit. Die hohe Modularität, die in Kapitel 3.1.1 beschriebene Komponentisierung von Services mit dem dazugehörigen unabhängigen Deployment stehen ebenfalls im direkten Gegensatz zu einer Anforderung an Wiederverwendbarkeit.

6.8.2.3. Analysability & Modifiability

Analog zu der Bewertung für serviceorientierte Architekturen in Kapitel 6.8.1.3 sorgt die Modularität der Services in einer Microservice Architektur dafür, dass sich die Auswirkungen von Änderungen an einzelnen Services sehr gut analysieren und bewerten lassen. Durch die im Kapitel 6.4.2 beschriebene Notwendigkeit die Kommunikation der Services untereinander zu minimieren haben sind Änderungen an fest definierten Schnittstellen immer noch sehr schlecht zu analysieren, die Auswirkungen sind jedoch deutlich geringer.

6.8.2.4. Testability

Die Testbarkeit von Microservice Architekturen ist analog zu den in Kapitel 6.8.1.4 beschriebenen Merkmalen der Testbarkeit von verteilten Systemen schwierig zu gewährleisten. Einzelne Services in einer Microservice Architektur lassen sich aufgrund ihrer Modularität sehr gut testen. Müssen Testfälle, die sich über mehrere Services innerhalb des Systems erstrecken, durchgeführt werden treten hier dieselben Problematiken wie bei serviceorientierten Architekturen auf.

6.8.3 Bewertung Maintainability

Architektur	Qualitätsattribut	Bewertung
Serviceorientierte Architekturen	Modularity	+
	Reusability	++
	Analysability & Modifiability	-
	Testability	--
	Maintainability (zusammengefasst)	o
Microservice Architekturen	Modularity	++
	Reusability	--
	Analysability & Modifiability	o
	Testability	-
	Maintainability (zusammengefasst)	-

Tabelle 9: Bewertung Qualitätsattribut Maintainability

6.9 Zusammenfassung

In diesem Kapitel werden die in Kapitel 5.2 beschriebenen Qualitätsattribute auf übergeordneter Ebene für beide, in dieser Arbeit untersuchten, Softwarearchitekturen gegenübergestellt.

Eine Übersicht der zuvor durchgeführten Bewertungen ist Tabelle 10 zu entnehmen.

Qualitätsattribut	Serviceorientierte Architekturen	Microservice Architekturen
Performance Efficiency	-/o	+/++
Compatibility	++	+
Reliability	+	++
Security	++	+
Maintainability	o	-

Tabelle 10: Zusammenfassung Qualitätsattributsbewertung

Bei der hier durchgeführten Bewertung wurden die Subattribute der Qualitätsattribute nicht gewichtet bewertet, sprich zu gleichen Teilen in der zusammengefassten Bewertung gerechnet.

Diese nicht gewichtete Art der Bewertung kann dazu führen, dass das Ergebnis der Qualitätsattributsanalyse für bestimmte Projekte ein falsches Ergebnis wiedergibt. Um dies zu kompensieren kann diese Bewertung für solche Projekte gewichtet stattfinden. Nachfolgend wird eine gewichtete Bewertung des Qualitätsattributs „Maintainability“ durchgeführt.

Die Gewichtungsfaktoren in dem beispielhaften Projekt sind wie in Tabelle 11 zu sehen definiert:

Qualitätsattribut	Gewichtungsfaktor
Modularity	3
Reusability	0
Analysability & Modifiability	2
Testability	2

Tabelle 11: Gewichtungsfaktoren für Qualitätsattribute Beispiel

In Tabelle 12 wird das Ergebnis dieser gewichteten Bewertung dargestellt:

Architektur	Qualitätsattribut	Punktewert * Gewichtungsfaktor	Bewertung
Serviceorientierte Architekturen	Modularity	1 * 3 = 3	+++
	Reusability	2 * 0 = 0	o
	Analysability & Modifiability	-1 * 2 = -2	--
	Testability	-2 * 2 = -4	----
	Maintainability (zusammengefasst)	-3	---
Microservice Architekturen	Modularity	2 * 3 = 6	++++++
	Reusability	-2 * 0 = 0	o
	Analysability & Modifiability	0 * 2 = 0	o
	Testability	-1 * 2 = -2	--
	Maintainability (zusammengefasst)	4	++++

Tabelle 12: Gewichtete Bewertung Beispiel

Die Bewertung aus Tabelle 12 zeigt, dass die projektbezogene Bewertung von einzelnen Qualitätsattributen sehr stark abhängig von den projektspezifischen Anforderungen ist.

Neben der Gewichtung von Anforderungen ist es des Weiteren wichtig zu berücksichtigen, dass Anforderungen an ein Qualitätsattribut Auswirkungen auf andere Qualitätsattribute haben können.

Ein Vorgehensmodell in dem berücksichtigt wird, dass Anforderungen gewichtet werden müssen und die Auswirkungen dieser Gewichtungen berücksichtigt werden ist das in Kapitel 4.1 beschriebene Architecture Tradeoff Analysis Method (ATAM).

In Schritt 7 der ATAM werden die Anforderungen an die Qualitätsattribute priorisiert. Im nachfolgenden Schritt findet eine Analyse statt welche Auswirkung die Wahl von Architekturansätzen zur Erfüllung bestimmter Qualitätsattribute auf die Erfüllung von anderen Qualitätsattributen hat. So können zum Beispiel sehr hohe Anforderungen an die Security einer Software negative Auswirkungen auf die Performanceeigenschaften einer Architektur haben.

Um diese Auswirkungen zu bewerten kann die in Kapitel 4.2 beschriebene Methode der Attribute Based Architecture Styles (ABAS) dienen. Dies geschieht anhand von bekannten Architekturstilen, die Erfahrungswerte zu den Eigenschaften und Auswirkungen bestimmter Architekturentscheidungen liefern. Anhand dieser Architekturstile kann vorhergesagt werden

welche Auswirkungen eine Architekturentscheidung auf das Verhalten des gesamten Systems und damit auf die Qualitätsattribute haben kann.

7 ANWENDUNGSTYPEN

In diesem Kapitel werden verschiedene Anwendungstypen beschrieben, für die die Verwendung von serviceorientierten und Microservice Architekturen in Frage kommen kann.

Hierbei wird sich auf Webanwendungen und beispielhaften Subtypen von Webanwendungen beschränkt.

Für diese Anwendungstypen werden die typischen Anforderungen an die in Kapitel 5.2 beschriebenen und in Kapitel 6.3 für diese Arbeit eingeschränkten Qualitätsanforderungen für diese Anwendungstypen zusammengetragen. Dies erfolgt anhand der Recherche von verfügbaren beispielhaften Anforderungen an spezifische Softwareprojekten und deren Anforderungen an die ausgewählten Qualitätsattribute.

Wurden in Kapitel 6.3 Kriterien von der Bewertung für die Softwarearchitekturen ausgeschlossen werden diese dennoch in diesem Kapitel bewertet, falls diese einen direkten Einfluss auf die Anforderungen an andere, für die Bewertung der Softwarearchitekturen ausgewählten, Qualitätsattribute haben.

7.1 Webanwendungen

Webanwendungen werden von (TechTerms, 2014) allgemein definiert als Software die, anstatt auf dem Betriebssystem eines Computers ausgeführt zu werden, auf einem Webserver gehostet wird und in einem Webbrowser geöffnet werden kann. Ein zentraler Vorteil von Webanwendungen ist laut (TechTerms, 2014) die Tatsache, dass diese Betriebssystemunabhängig sind und somit über verschiedene Plattformen eine gleichbleibende Benutzeroberfläche zur Verfügung stellen können. Zusätzlich werden alle getätigten Eingaben nicht auf dem lokalen Endgerät gespeichert, sondern auf dem zentralen Server, der die Webseite hostet.

Wie von (Rohr, 2015) beschrieben handelt es sich bei Webanwendungen nicht ausschließlich um Anwendungen, die aus dem öffentlichen Web verfügbar sind. Vielmehr beschreibt der Begriff Webanwendung eine Client-Server-Anwendung, die bei ihrer Implementierung und zur Laufzeit auf Webtechnologien wie HTTP, HTML etc. zurückgreift.

Abbildung 14 zeigt eine typische 3-stufige Architektur einer Webanwendung.

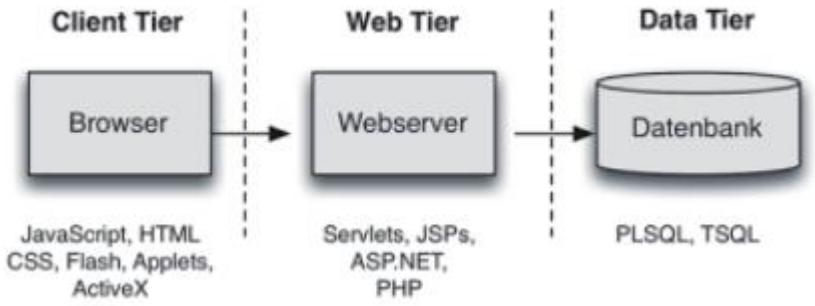

Abbildung 14: 3-Tier-Architektur einer Webanwendung (Rohr, 2015)

Nach (Rohr, 2015) handelt es sich hierbei um eine sehr einfache Anwendung, die durch eine 3-Schichten Architektur abgebildet werden kann. Bei modernen Enterprise-Webanwendungen kommen nach (Rohr, 2015) in der Regel Architekturen mit deutlich mehr Schichten zum Einsatz, da hier außerdem weitere Systeme wie ein ERP-System oder ein Enterprise Service Bus angebunden werden. Auf die speziellen Ausprägungen von Webanwendungen wird in den nachfolgenden Unterkapiteln eingegangen, da diese neben Auswirkungen auf die Architektur ebenfalls Auswirkungen auf die Anforderungen an Qualitätskriterien haben.

Es gibt eine ganze Reihe an speziellen Ausprägungen für Webanwendungen, die Einfluss auf ihrer Architektur und deren Anforderungen haben. (Business 2 Community, 2014) gibt einen guten Überblick über verschiedene Arten von Webanwendungen. Es werden unter anderem folgende, für die Bewertung von Architekturen relevante, Typen benannt:

- Content Management Systeme
- Customer-Relationship-Management Systeme
- Firmeninterne Webportale
- Kunden-Webportale
- E-Commerce Systeme / Online Shops

Eine genauere Definition und Beschreibung ausgewählter Typen von Webanwendung erfolgt in den nachfolgenden Unterkapiteln.

7.1.1 Web Content Management System

Ein Content Management System kann nach (Barker, 2016) beschrieben werden als ein „Softwarepaket, das ein gewisses Level an Automatisierungen bietet, die notwendig sind, um effektiv Inhalte verwalten zu können“.

Nach (Barker, 2016) gibt es verschiedene Ausprägungen von Content Management:

- Web Content Management: Verwaltung von Inhalten zur massenhaften Veröffentlichungen über eine Webseite

- Enterprise Content Management: Verwaltung von allgemeinen geschäftsrelevanten Inhalten, die in der Regel nicht für eine massenhafte Veröffentlichung vorgesehen sind.
- Digital Asset Management: Verwaltung von digitalen, medialen Inhalten wie Bildern, Soundfiles Videos.
- Records Management: Verwaltung von Informationen über Geschäftstransaktionen, die als Nebenprodukt des täglichen Geschäfts anfallen.

Im Rahmen dieser Arbeit und dieses Kapitels wird sich ausschließlich auf die Bewertung Web Content Management Systemen beschränkt.

Abbildung 15 zeigt eine typische Architektur für ein Web Content Management System:

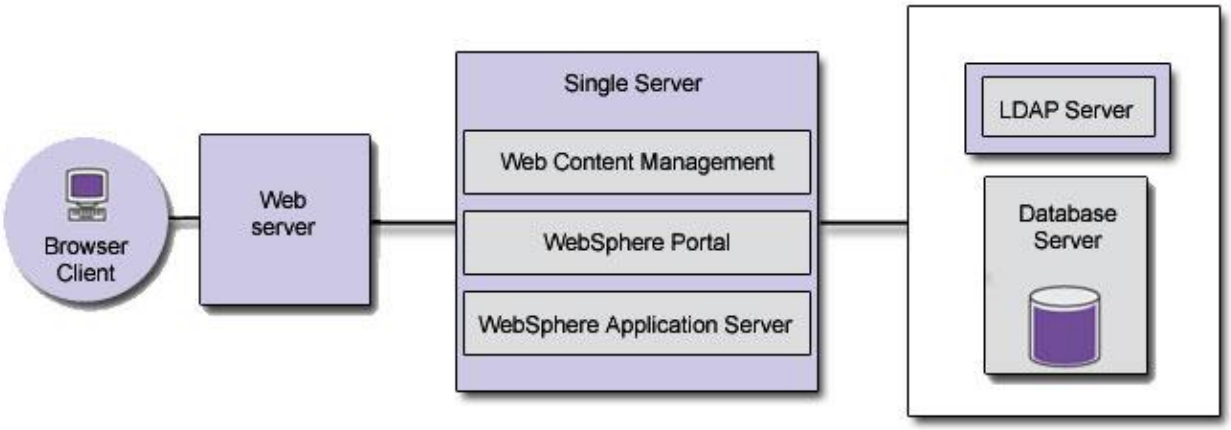

Abbildung 15: Web Content Management System Architecture (IBM, 2017)

Nachfolgend wird die Wichtigkeit der verschiedenen Qualitätskriterien bei Web Content Management Systemen bewertet.

7.1.1.1. Performance Efficiency

Performance beeinflusst nach (West, 2014) stark die Usability einer Webseite. Vielmehr beschreibt West die wahrgenommene Performance einer Webseite als das, was der Anwender denkt wie schnell eine Webseite ist. Spezifische Werte für Antwortzeiten, die diese wahrgenommene Performance messbar machen sollen hat (Nielsen, 1993) folgendermaßen definiert:

- 0.1 Sekunden: das Laden der Webseite fühlt sich unverzögert an und sollte das Ziel bei der Optimierung von Ladezeiten sein.
- 1 Sekunde: die Ladezeiten sind noch OK, sollte jedoch jedes Laden auf der Webseite eine Sekunde dauern fühlt sich die Anwendung träge an.
- 10 Sekunden: bei einer Ladezeit von mindestens 10 Sekunden wird es schwer das Interesse des Anwenders zu halten.

Geschuldet des Veröffentlichungszeitraums von Nielsen's Buch merkt (West, 2014) an, dass die obere Grenze des Verträglichen heute eher mit 5 Sekunden anstatt 10 Sekunden angegeben werden muss.

Ein anderer Ansatz die Wichtigkeit von Performance zu bewerten ist die Gewichtung von nichtfunktionalen Anforderungen im Vergleich zueinander. In einem CMS Review Report des australischen Departments of Finance and Deregulation (siehe (Australian Goverment - Department of Finance and Deregulation, 2012)) wurden die funktionalen sowie nichtfunktionalen Anforderungen gewichtet und mit einem Prozentsatz versehen, der diese Gewichtung wiederspiegelt. Aus den 30% Gewichtung die den nichtfunktionalen Anforderungen bemessen wurden, nehmen performancerelevante Anforderungen alleine 45% ein. Diese Performanceanforderungen unterteilt (Australian Goverment - Department of Finance and Deregulation, 2012) zu gleichen Teilen von 15% in allgemeine Performance, Verfügbarkeit und Skalierbarkeit. Zusammengefasst bedeutet dies, dass der Anteil der Performanceanforderungen an die gesamte Softwarelösung nach (Australian Goverment - Department of Finance and Deregulation, 2012) mit 13,5% gewichtet wird und somit einen zentralen Aspekt der Architekturarbeit definiert.

Einen anderen Ansatz hat das CULTURe Exchange Platform project bei der Auswahl eines Open Source Content Management Systems für grenzübergreifenden Austausch auf dieser Plattform gewählt. Nach (CULTURe EXchange Platform) soll durch Überwachung der Systemauslastung während Performance- und Lasttests garantiert werden, dass die wahrgenommene Performance nicht durch Bottlenecks in der Systemarchitektur leiden soll.

Zusammengefasst ist die Performance eine der wichtigsten nichtfunktionalen Anforderungen in Webanwendungen, da diese große Auswirkungen auf die, in Kapitel 7.1.1.3 beschriebenen, Usability hat.

7.1.1.2. Compatibility

Die Anforderungen von Web Content Management Systemen an die Fähigkeiten Informationen mit anderen Systemen austauschen zu können hängen stark von den funktionalen Anforderungen an das System ab. Bei den Recherchen für diese Arbeit konnten keine konkreten Anforderungen an Informationsaustausch mit Systemen dritter für Web Content Management Systeme identifiziert werden. Aus diesem Grund wird die Bewertung dieses Qualitätsattributs für diesen Anwendungstypen ausgelassen.

7.1.1.3. Usability

Die Anforderungen an die Usability in Web Content Management Systemen fokussiert sich nach (usability first) primär auf Performanceanforderungen und den Funktionsumfang der zur Verfügung gestellten Tools. Als Grund hierfür wird genannt, dass es bei den Anwendern primär um trainierte Mitarbeiter handelt, die Experten der Software sind. Diese Sicht beschränkt sich jedoch auf die Anwender, die Inhalte für das System erstellen.

Im Gegensatz zu dieser Ansicht beschreiben (Goheen & Martin, 2007) bei der Auswahl eines Web Content Management Systems für das U.S. Department of Health and Human Services die Usability als ein wichtiges Auswahlkriterium. Der Fokus liegt hier auf der einfachen Erlernbarkeit des Systems, da sonst die Gefahr zu hoch ist, dass das System durch die Benutzer nicht zu ihrem Vorteil benutzt wird.

Diese beiden Ansichten sind eine Indikation dafür, dass der Fokus der Usability im Redaktionsbereich stark davon abhängt in welcher Organisation diese Software eingesetzt werden soll. Handelt es sich bei den Redakteuren um ein kleines Team von Experten, die spezialisiert sind in der Bedienung von Web Content Management Systemen rückt die Usability in den Hintergrund und die Anforderungen an die Funktionalitäten überwiegen. Soll die Software jedoch in einer größeren Organisation ausgerollt werden in der die meisten Anwender noch nicht mit Web Content Management Systemen in Berührung bekommen sind steigen die Anforderungen an die Usability.

In der Gewichtung der nichtfunktionalen Anforderungen hat (Australian Goverment - Department of Finance and Deregulation, 2012) der Usability eine Gewichtung von 10% der nichtfunktionalen und somit 3% der gesamten Anforderungen bemessen.

7.1.1.4. Security

Bei der Definition der Anforderungen an die Security hat (CULTURe EXchange Platform) Kriterien definiert, die ein Web Content Management System erfüllen muss:

- Webserver müssen gegen Denial of Service und anderen Attacken abgesichert sein.
- Benutzeranmeldungen finden verschlüsselt über SSL per Username und Passwort statt.
- Die Software sollte Penetrationstests für gängige Sicherheitslücken unterzogen werden.
- Die Server sollten durch System Scanner Tools durchgängig auf Sicherheitslücken überprüft werden.
- Die Software soll gemäß des Open Web Application Security Projects (OWASP) auf alle bekannten Sicherheitsprobleme geprüft werden.

Diese Anforderungen sind ein Indikator dafür, dass die Security der Software bewusst und systematisch bewertet wird und somit für relativ hohe Anforderungen an die Security spricht.

Im Gegensatz hierzu gewichtet (Australian Goverment - Department of Finance and Deregulation, 2012) die Security mit 10% der nichtfunktionalen Anforderungen und somit 3% der gesamten Anforderungen verhältnismäßig niedrig.

Zusammengefasst sprechen diese Anforderungen dafür, dass Security in der Architekturauswahl für Web Content Management Systeme ernsthaft betrachtet werden müssen.

7.1.1.5. Maintainability

Die Erweiterbarkeit eines Web Content Management Systems ist nach (CULTURe EXchange Platform) ein entscheidender Faktor dafür, ob die Software ein erfolgreiches System sein kann. Der Fokus liegt hierbei auf die Erweiterbarkeit der Software durch Module und Plugins, die frei wählbar hinzugefügt werden können. Hieraus leitet sich ab, dass die Architektur der Software so gestaltet werden muss, dass in einem fest definierten Rahmen Erweiterungen entwickelt werden können und somit hohe Anforderungen an die Modularität und Reusability bestehen. Nicht hieraus abgeleitet werden kann eine erhöhte Anforderung an die Änderbarkeit des Systems, da für die Verwendung von zusätzlichen Plugins und Modulen keine Veränderungen am System selbst notwendig sind.

7.1.1.6. Zusammenfassung

In der nachfolgenden Matrix werden die zuvor bewerteten Qualitätskriterien zusammengeführt und dargestellt.

Qualitätskriterium	Bewertung
Performance efficiency	**+**
Compatibility	***/***
Usability	**+**
Security	**+**
Maintainability	**+**

Tabelle 13: Bewertung Qualitätskriteriengewichtung bei Web Content Management Systemen

7.1.2 E-Commerce System / Online Shop

Grundlegend wird E-Commerce nach (TechTarget, 2016) beschrieben als das „Kaufen und Verkaufen von Waren und Dienstleistungen, die Übertragung von finanziellen Mitteln oder Daten, über ein elektronisches Netzwerk“. Diese verschiedenen Arten von Transaktionen können zwischen unterschiedlichen Parteien stattfinden. (TechTarget, 2016) beschreibt folgende Transaktionswege:

- Business-to-Business (B2B)
- Business-to-Consumer (B2C)
- Consumer-to-Consumer (C2C)
- Consumer-to-Business (C2B)

E-Commerce kann laut (TechTarget, 2016) über verschiedenste Arten von Softwareanwendungen durchgeführt werden, wie zum Beispiel E-Mails, Online Shops, Eletronic Data Interchange (EDI) oder Webservices. Ja nach Transaktionsweg unterscheiden sich die

Qualitätsanforderungen an eine Architektur. Im Rahmen dieser Arbeit wird sich bei der Anforderungsbewertung auf den Business-to-Customer Weg in Form von Webshops sowie dem Business-to-Business Weg beschränkt.

7.1.2.1. Performance efficiency

Neben den bereits in Kapitel 7.1.1.1 beschriebenen Anforderungen an die Performance in Web Content Management Systemen ist die Performance in E-Commerce Systemen deutlich wichtiger. Nach (Aberdeen Group, 2008) sinkt die sogenannte Conversion Rate um 7% für jede Sekunde die, die Webseite länger braucht zu laden. Dies bedeutet, dass für jede Sekunde um die die Ladezeit einer Seite verlängert 7% der Kunden einem Online Shop verloren gehen, was zu einem direkten Umsatzverlust führt.

Zusammengefasst sind die Anforderungen an die Performance in E-Commerce Systemen sehr hoch.

7.1.2.2. Compatibility

Die Notwendigkeit in E-Commerce Systemen mit anderen Systemen Informationen auszutauschen ist hoch. Die Vorteile zum Beispiel von einem direkten Informationsaustausch mit einem Enterprise Resource Planning (ERP) System durchzuführen sind nach (Carpenter, 2015):

- Optimierung der Geschäftsprozesse durch direkte Synchronisierung von Bestellungen, Beständen etc.
- Einsparung von manuellem Arbeitsaufwand bei der Synchronisierung von Systemen.
- Sicherstellung, dass das E-Commerce System sowie das ERP System anhand von gleichen Informationen Entscheidungen treffen.

7.1.2.3. Usability

Die Usability in E-Commerce Anwendungen hat die gleichen Anforderungen an die Performance des Systems wie in Kapitel 7.1.1.3 beschrieben.

7.1.2.4. Security

In einer Umfrage mit 44 Teilnehmern hat (Rahman, et al., 2015) ermittelt, dass 88% der Teilnehmer Security für relevant halten, wenn sie online einkaufen oder Onlinebanking betreiben. Von diesen 88% halten 43% der Teilnehmer Security für sehr wichtig.

Nach (Judge, 2013) sind e-Commerce Systeme ein primäres Ziel für Angriffe durch dritte, da in diesen persönliche Informationen sowie Zahlungsinformationen wie Kreditkartendaten verwaltet werden. Judge beschreibt neben dem finanziellen Verlust durch einen erfolgreichen Angriff den

Schaden auf das Image des Online Shops als viel schwerwiegender. Es werden von (Judge, 2013) unter anderem die folgenden Anforderungen an die Security beschrieben:

- Extended Validation SSL verwenden: neben der einfachen Verifizierung, dass es sich bei dem Empfänger des Zertifikats um den Inhaber der Domäne handelt, werden weiterführende Prüfungen bezüglich der Identität und Echtheit des Beantragenden des Zertifikats durchgeführt.
- Security Vulnerability Scanners: es sollten Dienste in Anspruch genommen werden, die die Webseite regelmäßig auf Compliance mit dem Payment Card Industry Data Security Standard sowie auf korrupte, Malware ausliefernde, Seite prüfen.
- Penetration Testing: Sicherheitslücken in der Software sowie der gesamten Infrastruktur sollten durch externe Penetration Tests identifiziert und anschließend geschlossen werden.
- Multi-Faktor Authentifizierung: neben der Anmeldung über Username und Passwort sollte ebenfalls ein anderes Medium für die Sicherstellung der Identität des Benutzers verwendet werden. Dies kann nach (Judge, 2013) zum Beispiel ein Authentifizierungscode per E-Mail oder SMS sein, falls der Benutzer sich versucht mit einem anderen Computer anzumelden als zuvor.

Aus den Daten der Umfrage, dem Umstand welche Daten durch ein E-Commerce System verwaltet werden und den zuvor aufgelisteten Anforderungen an Sicherheitsmechanismen kann zusammenfassend gesagt werden, dass die Anforderungen an die Security in E-Commerce Systemen sehr hoch sind.

7.1.2.5. Maintainability

Die Wartbarkeit von E-Commerce Systemen ist nach (SEOMining, 2017) ein wichtiger, in der Architektur zu berücksichtigender, Faktor.

Als Gründe werden unter anderem angegeben, dass sich neben den häufigen Änderungen an dem Aussehen von E-Commerce Systemen ebenfalls die Inhalte ständigen Änderungen unterliegen.

7.1.2.6. Zusammenfassung

In Tabelle 14 werden die zuvor bewerteten Qualitätskriterien zusammengeführt und dargestellt.

Qualitätskriterium	Bewertung
Performance Efficiency	++
Compatibility	++
Security	++
Maintainability	+

Tabelle 14: Bewertung Qualitätskriteriengewichtung bei E-Commerce Systemen

7.2 Zusammenfassung

Die an beispielhaften Anforderungen in Kapitel 7.1 durchgeführte Bewertung der Anforderungen an bestimmte Arten von Webanwendungen ist zusammengefasst in Tabelle 15 zu sehen.

Qualitätsattribut	Web Content Management Systeme	E-Commerce Systeme
Performance Efficiency	+	++
Compatibility	/	++
Usability	+	+
Security	+	++
Maintainability	+	+

Tabelle 15: Zusammenfassung Bewertung Qualitätskriteriengewichtung

Durch die geringe Menge an öffentlich verfügbaren Anforderungen an diese Anwendungstypen ist die in Tabelle 15 zu sehende Bewertung nicht allgemeingültig.

Auch, wenn eine Allgemeingültigkeit dieser Bewertung nicht gegeben ist lassen sich für bestimmte Qualitätsattribute für bestimmte Anwendungstypen eindeutige Trends erkennen. So kann angenommen werden, dass das Qualitätsattribut der Security in E-Commerce Systemen immer eine sehr hohe Priorität sein sollte, um sensible Kundendaten zu schützen.

8 MAPPING VON ARCHITEKTUREN AUF ANWENDUNGSTYPEN

In diesem Kapitel werden die in Kapitel 6 ermittelten Eigenschaften der, in dieser Arbeit untersuchten, Architekturen mit den in Kapitel 7 ermittelten Anforderungen von bestimmten Anwendungstypen gegenübergestellt. Diese Gegenüberstellung wird anhand der nicht gewichteten Bewertung durchgeführt. Wie in Kapitel 6.9 beschrieben müssen für spezifische Projekte diese Bewertungen gewichtet durchgeführt werden.

8.1 Abdeckungsberechnung

Um feststellen zu können zu welchem Grad die, in dieser Arbeit untersuchten, Architekturen sich mit den Anforderungen der Anwendungstypen decken muss die Abdeckung der Anforderungen beziehungsweise die Abweichung der Architektureigenschaften von den Anforderungen bestimmt werden.

In diesem Kapitel wird die Abweichung der Architektureigenschaften von den Anforderungen der Anwendungstypen anhand der in Kapitel 6.2 beschriebenen Punktewerte der Bewertungsskala beschrieben.

Nachfolgend wird anhand eines Qualitätsattributs beispielhaft die Abweichung berechnet.

Qualitätsattribut A für Architektur A: + (1 Punkt)

Qualitätsattribut A für Anwendungstyp B: ++ (2 Punkte)

Abweichung: 1 Punkt

Es wird ausschließlich eine Abweichung berechnet, falls die bewertete Architektur die Anforderungen des Anwendungstyps nicht erfüllen kann. Eine Übererfüllung einer Anforderung führt zu keiner Abweichung.

Die Abweichungen aller Qualitätsattribute werden abschließend addiert und ergeben den gesamten Abweichungsgrad der Architektur von den Anforderungen. Je geringer dieser Abweichungswert desto besser eignet sich die Architektur für diesen Anwendungstypen.

8.2 Serviceorientierte Architekturen

In Tabelle 16 sind die Anforderungen der Anwendungstypen mit den Eigenschaften von serviceorientierten Architekturen gegenübergestellt.

Qualitätsattribut	Serviceorientierte Architekturen	Web Content Management Systeme	E-Commerce Systeme
Performance Efficiency	-/o	+	++
Compatibility	++	/	++
Security	++	+	++
Maintainability	o	+	+

Tabelle 16: Gegenüberstellung serviceorientierte Architekturen mit Anwendungstypenanforderungen

In Tabelle 17 sind die Abweichungen von serviceorientierten Architekturen zu den Anforderungen der Anwendungstypen dargestellt.

Qualitätsattribut	Abweichung zu Web Content Management Systemen	Abweichung zu E-Commerce Systemen
Performance Efficiency	1,5	2,5
Compatibility	0	0
Security	0	0
Maintainability	1	1
Gesamtabweichung	2,5	3,5

Tabelle 17: Abweichung Qualitätsattribute von serviceorientierten Architekturen

8.3 Microservice Architekturen

In Tabelle 18 sind die Anforderungen der Anwendungstypen mit den Eigenschaften von Microservice Architekturen gegenübergestellt.

Qualitätsattribut	Microservice Architekturen	Web Content Management Systeme	E-Commerce Systeme
Performance Efficiency			
Compatibility			
Security			
Maintainability			

Tabelle 18: Gegenüberstellung Microservice Architekturen mit Anwendungstypenanforderungen

In Tabelle 19 sind die Abweichungen von serviceorientierten Architekturen zu den Anforderungen der Anwendungstypen dargestellt.

Qualitätsattribut	Abweichung zu Web Content Management Systemen	Abweichung zu E-Commerce Systemen
Performance Efficiency	1,5	2,5
Compatibility	0	0
Security	0	0
Maintainability	1	1
Gesamtabweichung	2,5	3,5

Tabelle 19: Abweichung Qualitätsattribute von Microservice Architekturen

8.4 Zusammenfassung

Die in den Kapitel 8.2 und 8.3 separat bewertete Abweichung der, in dieser Arbeit untersuchten, Architekturen von den Anforderungen der Anwendungstypen ist in Tabelle 20 zusammengefasst.

Anwendungstyp	Abweichung serviceorientierte Architekturen	Abweichung Microservice Architekturen
Web Content Management Systeme	2,5	2
E-Commerce Systeme	3,5	4

Tabelle 20: Gegenüberstellung von Abweichungen Qualitätsattribute

Dieses Ergebnis zeigt, dass sich beide Architekturen für bestimmte Anwendungstypen leicht besser eignen aber es nur sehr geringe Unterschiede bei der Abweichung gibt.

Bei diesem Ergebnis ist wie bereits in Kapitel 6.9 zu beachten, dass es sich hierbei um nicht gewichtete Bewertungen handelt in denen jedes Qualitätsattribut die gleiche Priorität erhalten hat. In spezifischen Projekten müssen vor einer Auswahl der Architektur die Anforderungen an Qualitätskriterien gewichtet werden. Zusätzlich zu der Gewichtung muss ebenfalls der in Kapitel 6.9 beschriebene Einfluss von Anforderungen an bestimmte Qualitätsattribute auf die Erfüllung anderer Qualitätsattribute berücksichtigt werden.

9 FAZIT

Die in Kapitel 1 beschriebene Forschungsfrage „Ist neu immer besser?“ und die weitere Ausführung dieser Frage, ob Microservice Architekturen die besseren serviceorientierten Architekturen sind kann grundsätzlich mit einem „Nein“ beantwortet werden, da serviceorientierte Architekturen Stärken in Bereichen besitzt, die Microservice Architekturen nicht für alle Anforderungen erfüllen kann.

Wie die Ausarbeitungen in Kapitel 2 und 3 zeigen sind die grundlegenden Ideen der beiden Architekturen, dass diese basierend auf Services umgesetzt werden und es sich um verteilte Systeme handelt sehr ähnlich.

Die Ausgestaltungen dieser Architekturen unterscheiden sich aber in einigen zentralen Punkten stark. So wird bei klassischen serviceorientierten Architekturen stark auf das Prinzip der Reusability und der Kommunikation der Services untereinander über eine zentrale, intelligente Kommunikationsinfrastruktur gesetzt. Microservices hingegen setzen stark auf, wie in Kapitel 3.1 beschrieben, „kluge Endpunkte und dumme Pipes“ und dezentraler Governance und Datenmanagement. Außerdem wird sich explizit auf die Ersetzbarkeit von Services fokussiert im Gegensatz zu der großen Wichtigkeit von Reusability bei serviceorientierten Architekturen.

Die Bewertung der beiden Architekturen anhand der in Kapitel 5 definierten Qualitätskriterien hat in Kapitel 6 gezeigt, dass diese Unterschiede dazu führen, dass die Stärken der Architekturen bei verschiedenen Qualitätsattributen liegen.

Während serviceorientierte Architekturen beispielsweise nur eine beschränkte Skalierbarkeit und durch den hohen Aufwand bei der Servicekommunikation Einschränkungen bei der Performance aufweisen zeichnen sich serviceorientierte Architekturen durch ihre extreme Skalierbarkeit und der geringen Kommunikation der Services untereinander durch sehr gutes Performanceverhalten aus. Serviceorientierte Architekturen weisen wiederrum Stärken bei der Compatibility und Security durch die Unterstützung von Industriestandards und reifen Konzepten und Produkten auf während Microservice Architekturen in diesen Bereichen im Vergleich schwächer aufgestellt sind. Die Schlussfolgerung hiervon ist, dass je nach den Anforderungen der Software entweder klassische serviceorientierte Architekturen oder Microservice Architekturen geeignet sind und durch die Verwendung einer Architekturbewertungsmethode wie die Architecture Tradeoff Analysis Method die passende Architektur gewählt werden kann.

Anhand von beispielhaften Typen von Webanwendungen wurde in Kapitel 7 gezeigt, dass es Trends in den Anforderungen an bestimmte Qualitätsattribute für bestimmte Anwendungstypen gibt, die kompletten Anforderungen aber sehr stark von der spezifischen Software abhängig sind. Bei dem Vergleich der in Kapitel 7 festgestellten Anforderungen der Anwendungstypen wurde in Kapitel 8 gezeigt, dass serviceorientierte Architekturen und Microservice Architekturen

keine großen Abweichungen bei der Erfüllung der Anforderungen haben, solange die nicht gewichtet sind. Sobald spezifische Gewichtungen und Anforderungen definiert sind lässt sich die Entscheidung für eine Architektur wie zuvor beschrieben treffen.

ANHANG A - 1. Anhang

ANHANG B - 2. Anhang

ABKÜRZUNGSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

LITERATURVERZEICHNIS

Aberdeen Group. (2008). *The Performance of Web Applications.*

Apiman. (2017). *Apiman Policies*. Retrieved from https://apiman.gitbooks.io/apiman-user-guide/user-guide/gateway/policies.html

Australian Goverment - Department of Finance and Deregulation. (2012, Juni). *CMS Review Report.* Retrieved from http://www.finance.gov.au/sites/default/files/DoFD%20CMS%20Review%20Report%20-%20V2%202%20FINAL.pdf

Barker, D. (2016). *Web Content Management.* O'Reilly Media, Inc.

Boehm, B. W. (1988). Spiral Model of Software Development and Enhancement. *Computer Volume 21 Issue 5*, 61-72.

Buecker, A., Ashley, P., Borrett, M., Lu, M., Muppidi, S., & Readshaw, N. (2007). *Understanding SOA Security Design and Implementation.* IBM.Com/Redbooks.

Business 2 Community. (2014, März 5). *7 Types of Custom Web Applications to Help Simplify and Grow Your Business*. Retrieved from http://www.business2community.com/mobile-apps/7-types-custom-web-applications-help-simplify-grow-business-0800428#rCxMocRs1CHu1w1v.97

Carpenter, L. (2015, September 22). *eCommerce ERP Integration: Why Retailers Should Integrate and How to Do It*. Retrieved from https://www.nchannel.com/blog/ecommerce-erp-integration

Cearley, D. W., Fenn, J., & Plummer, D. C. (2005). *Gartner's Positions on the Five Hottest IT Topics and Trends in 2005.*

Code Project. (2013, Juni 10). *JSON vs. XML: Some hard numbers about verbosity*. Retrieved from https://www.codeproject.com/Articles/604720/JSON-vs-XML-Some-hard-numbers-about-verbosity

Conway, M. E. (1968). *How Do Committees Invent?* Retrieved from http://www.melconway.com/Home/Committees_Paper.html

CULTURe EXchange Platform. (n.d.). *CULTUR EXchange Platform - Non-Functional Requirements for Open-Source CMS Report.* Retrieved from http://project.cultur-exp.eu/admin/editor/uploads/files/CULTUR-EXP_Non-functional_Requirements.pdf

Dikmans, L., & Alves, B. N. (2016, Juni 21). *Oracle SOA Integration for Healthcare - 12C.* Retrieved from HL7 real time integration - A Case Study: https://www.slideshare.net/bmalves/ukoug-tech16-use-case-with-oracle-soa-integration-for-healthcare-12c

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017, April 20). *Microservices: yesterday, today, and tomorrow.* Retrieved from https://arxiv.org/abs/1606.04036v4

Eisenbarth, T., & Koschke, R. (2012, November 23). *Erosion von Software*. Retrieved from Software-Wartbarkeit – der unterschätzte Wettbewerbsvorteil: http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/testinstallation/articles/386818/

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR.

Fowler, M. (2014, März 6). *CircuitBreaker*. Retrieved from https://martinfowler.com/bliki/CircuitBreaker.html

Fowler, M., & Lewis, J. (2014, 03 25). *Microservices - a definition of this new architectural term*. Retrieved from http://martinfowler.com/articles/microservices.html

Fowler, S. J. (2016). *Production-Ready Microservices.* O'Reilly Media, Inc.

Gallagher, B. P. (2000). *Using the Architecture Tradeoff Analysis Method to Evaluate a Reference Architeture: A Case Study.*

Goheen, E., & Martin, A. B. (2007, Januar 1). *Usability Testing a Web Content Management System*. Retrieved from https://www.usability.gov/get-involved/blog/2007/01/hhs-usability-tests-wcms-options.html

Google. (2017, Juli). *Google Trends - Microservice*. Retrieved from https://trends.google.com/trends/explore?date=today%205-y&q=Microservice

IBM. (2017). *Single-server topology for Web Content Manager*. Retrieved from https://www.ibm.com/support/knowledgecenter/en/SSHRKX_8.0.0/wcm/wcm_topology_simple.html

ISO/IEC JTC 1/SC 07 Software and systems engineering. (2005). *ISO/IEC 25010*. Retrieved 09 03, 2016, from http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Josuttis, N. M. (2007). *SOA in Practice.* O'Reilly Media, Inc.

Judge, K. (2013, Juli 29). *6 Essential Requirements for a Secure e-Commerce Site*. Retrieved from http://it.toolbox.com/blogs/internet-security/6-essential-requirements-for-a-secure-ecommerce-site-56891

Kazman, R., Bass, L., & Clements, P. (2003). *Software Architecture in Practice, Second Edition.* Addison-Wesley Professional.

Kazman, R., Clements, P., & Bass, L. (2012). *Software Architecture in Practice, Third Edition.* Addison-Wesley Professional.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). *The Architecture Tradeoff Analysis Method.*

Klein, M. H., Kazman, R., Bass, L., Carriere, J., Barbacci, M., & Lipson, H. (1999). Attribute-Based Architectural Styles. *Proceedings of the First Working IFIP Conference on Software Architecture*, 225-243.

Klein, M., & Kazman, R. (1999). *Attribute-Based Architectural Styles.*

Krafzig, D., Banke, K., & Slama, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices.* Prentice Hall PTR.

Linthicum, D. S. (1999). *Enterprise Application Integration.* Addison-Wesley Professional.

Moulliard, C. (2016, Juni 22). *Security Enforcement of the Microservices*. Retrieved from https://cmoulliard.github.io/microservices-security

Nadhan, E. G. (2004, April). *Service-Oriented Architecture: Implementation Challenges*. Retrieved from https://msdn.microsoft.com/en-us/library/aa480029.aspx

Newman, S. (2015). *Building Microservices.* O'Reilly Media, Inc.

Nielsen, J. (1993). *Usability Engineering.* Morgan Kaufmann.

O'Brien, L., Bass, L., & Merson, P. (2005, September). *Software Engineering Institute.* Retrieved from Quality Attributes and Service-Oriented Architectures: https://www.sei.cmu.edu/reports/05tn014.pdf

Peng, K.-L., & Huang, C.-Y. (2014, Januar 9). Reliability Evaluation of Service-Oriented Architecture Systems Considering Fault-Tolerance Designs. *Journal of Applied Mathematics*.

Rahman, W. N., Kamal, A. B., Talha, H., Josiah, B., Adamu, L., Liming, W., & Rosli, N. S. (2015). Software Quality Assurance – E-commerce Customers Satisfaction in Requirements Engineering Process. *International Journal of Software Engineering and Its Applications Vol. 9, No. 3*, pp. 57-70.

Rohr, M. (2015). *Sicherheit von Webanwendungen in der Praxis: Wie sich Unternehmen schützen können – Hintergründe, Maßnahmen, Prüfverfahren und Prozesse.* Springer Vieweg.

SEOMining. (2017). *Maintainability: A consideration for each tool category*. Retrieved from http://www.seomining.com/e-business-technology/module2/maintainability-influences-ecommerce.php

Software Engineering Institute. (1998). *Architecture Tradeoff Analysis Method*. Retrieved 09 01, 2016, from https://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

TechTarget. (2014, November). *customer relationship management (CRM)*. Retrieved from http://searchcrm.techtarget.com/definition/CRM

TechTarget. (2016, Juni). *e-commerce (electronic commerce or EC)*. Retrieved from http://searchcio.techtarget.com/definition/e-commerce

TechTerms. (2014, Februar 17). *Web Application Definition*. Retrieved from https://techterms.com/definition/web_application

Toth, S. (2016, Mai). *An Inverse Evaluation of Netflix Architecture Using ATAM.* Retrieved from https://resources.sei.cmu.edu/asset_files/Presentation/2016_017_001_454646.pdf

usability first. (n.d.). *usability first*. Retrieved from Content Management Systems: http://www.usabilityfirst.com/about-usability/web-application-design/content-management-systems

webopedia. (2017). *enterprise application*. Retrieved from http://www.webopedia.com/TERM/E/enterprise_application.html

West, M. (2014, April 23). *An Introduction to Perceived Performance*. Retrieved from http://blog.teamtreehouse.com/perceived-performance

Wolff, E. (2016). *Microservices - Grundlagen flexibler Softwarearchitekturen.* dpunkt.verlag GmbH.