Institut für Softwaretechnologie
Empirical Software Engineering Group

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Migration monolithischer Anwendungen in Microservices-basierte Architekturen: Fallstudie einer Service/Sales-Applikation

Marvin Christian Knodel

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Wagner
Betreuer:	Jonas Fritzsch, M.Sc. Dr. Alexander Eisold (L-mobile)
Beginn am:	15. Februar 2023
Beendet am:	15. August 2023

Kurzfassung

Viele Altsysteme in der Industrie sind heutzutage in einer monolithischen Architektur implementiert. Manche Unternehmen setzen darauf ihre großen Applikationen in eine Microservices-Architektur zu migrieren, da sie sich hiervon viele Vorteile versprechen. So ist auch das Unternehmen L-mobile aus Sulzbach an der Murr dazu gewillt ihre Service/Sales-Applikation auf einen möglichen Microservices-Betrieb hin zu führen. Da es viele Ansätze gibt einen Monolithen in eine Microservices Applikation zu migrieren, hat die Abteilung Empirical Software Engineering des Institute of Software Engineering der Universität Stuttgart ein Framework für die Microservices Migration entwickelt, welches insbesondere Ansätze aus dem wissenschaftlichen Umfeld beinhaltet. Mithilfe dieses Frameworks wird in dieser Arbeit eine Teilmigration der Service/Sales-Applikation von L-mobile im Rahmen eines Proof of Concept durchgeführt. Dafür wurde zuerst eine Literaturrecherche durchgeführt um die Grundlagen von Monolithen, Microservices und dahingehende Migrationen im Allgemeinen zu erörtern. Anschließend wurde das Framework für Microservices Migration, für eine Teilmigration der Service/Sales-Applikation, durchgeführt. In dieser Durchführung wurde ein Service-Identifikationsansatz und eine Migrationsstrategie für die Applikation von L-mobile durch das Framework empfohlen. Während der Migration sind auch Herausforderungen aufgetreten. Einige der aufgetretene Herausforderungen wie die Migration der Datenbank werden auch in der wissenschaftlichen Literatur genannt, andere Herausforderungen, wie mangelnde Erfahrung mit Architekturbewertungen und der Implementierung von Microservices sind L-mobile spezifische Herausforderungen. Durch das Erheben strukturierter Feldnotizen während der Anwendung des Frameworks und durch verschiedene Reviews nach der Migration wurde das Framework hinsichtlich seiner Eignung für die Migration der Service/Sales-Applikation geprüft. Diese Evaluation ergab, dass sich das Framework für die Migration im Rahmen des Proof of Concept geeignet hat, da es umfangreich durch die Migration führt, eine Architekturbewertung berücksichtigt, geeignete Methoden für die Service-Identifizierung und Migration vorschlägt und durch das Vorschlagen von Patterns und Best Practices bei der Erstellung der Architektur unterstützt. Das Framework eignet sich auch für die komplette Migration der Service/Sales-Applikation.

Abstract

Many legacy systems in industry today are implemented in a monolithic architecture. Some companies rely on migrating their large applications to a microservices architecture because they expect many advantages from this. The company L-mobile from Sulzbach an der Murr is also willing to lead its service/sales application to a possible microservices operation. Since there are many approaches to migrate a monolith into a microservices application, the Empirical Software Engineering department of the Institute of Software Engineering at the University of Stuttgart has developed a framework for microservices migration, which includes approaches from the scientific environment in particular. With the help of this framework, a partial migration of the service/sales application from L-mobile is carried out in this work as part of a

proof of concept. For this purpose, a literature search was first carried out in order to discuss the basics of monoliths, microservices and related migrations in general. The framework for microservices migration was then carried out for a partial migration of the service/sales application. In this implementation, a service identification approach and a migration strategy for the application of L-mobile was recommended through the framework. Challenges appeared during the migration. Some of these challenges, such as the migration of the database, are also mentioned in the scientific literature, other challenges, such as a lack of experience with architecture assessments and the implementation of microservices, are L-mobile-specific challenges. The suitability of the framework for the migration of the service/sales application was checked by collecting structured field notes while using the framework and by carrying out various reviews after the migration. This evaluation showed that the framework was suitable for the migration as part of the proof of concept, as it comprehensively guides through the migration, takes into account an architecture assessment, proposes suitable methods for service identification and migration, and by proposing patterns and best Practices assisted in creating the architecture. The framework is also suitable for the complete migration of the service/sales application.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

API Application Programming Interface. 26

ARH Architektur Refactoring Helper. 20

ATAM Architecture Tradeoff Analysis Method. 33

AWS Amazon Web Service. 35

CI Continuous Integration. 68

CRM Customer-Relationship-Management-System. 19

DB Datenbank. 24

DoS Denial of Service. 94

DTO Daten-Transfer-Obkjekt. 38

ERP Enterprise-Resource-Planning-System. 19

ESE Empirical Software Engineering. 20

FMM Framework für Microservices-Migrationen. 20

IIS Internet Information Service. 96

ISTE Institute of Software Engineering. 20

LoC Lines of Code. 62

MDA Meta-Daten getrieben. 57

MDD Model-Driven Development. 35

MS Microservice. 36

MSA Microservices-Architektur. 19

MSs Microservices. 19

PoC Proof of Concept. 31

QAs Qualitätsattribute. 34

REST Representational State Transfer. 26

RR Rapid Review. 39

SAAM Software Architecture Analysis Method. 33

SaaS Software-as-a-Service. 19

SCA Statische Code Analyse. 57

SLR Systematische Literatur Recherche. 39

SPs System-Eigenschaften. 57

SQAs Sub-Qualitätsattribute. 57

SRP Single Responsibility Principle. 26

URI Uniform Resource Identifier. 86

VPN Virtual Private Network. 41

1. Einleitung

Auf Basis der heutigen Praxis ist es üblich, bei der Entwicklung einer neuen Software mit einer monolithischen Architektur zu beginnen [FL14, KMM18]. Allerdings offenbart die monolithische Architektur, sobald die Anwendung stark expandiert, zunehmend Schwächen [FBZW19, KMM18, PMA19]. Diese Schwächen zeigen sich vor allem in der schweren Verwaltbarkeit, der Herausforderung der Wartung sowie der Unfähigkeit von Monolithen, Situationen zu bewältigen, in denen die Serverkapazität von der Anzahl der Nutzer überschritten wird [FBZW19, FM17, KMM18, PMA19].

Der Microservices-Architektur (MSA)-Stil wurde entwickelt, um diesen Problemen entgegenzuwirken [FL14]. Microservices (MSs) sind unabhängig voneinander skalierbar, lassen sich unabhängig voneinander installieren und sie sind einfacher zu warten [FL14, TS19].
Um den Vorteil dieser Microservices-basierten Architektur zu nutzen, setzen viele Unternehmen, wie z. B. Netflix, Amazon, Google, eBay, LinkedIn, SoundCloud, Spotify und Otto, heutzutage darauf, ihre bestehenden Monolithen in Microservices umzustrukturieren [CM22, FBH+22, FBWZ19, KMM18, LXR+22, MCL17]. Jedoch ist die Migration eines Monolithen in eine MSA eine herausfordernde Aufgabe [ASM+21, DLM18, FSC+21]. Vielen Faktoren, wie die Auswahl einer Migrationsstrategie und eines Service-Identifikationsansatz, spielen dabei eine Rolle [ASM+21]. Diesen Paradigmenwechsel hin zu Microservices nennen Hassan et al. [HAB17] *microservitization*. In der wissenschaftlichen Literatur existieren verschiedene Methoden, um Legacy-Applikationen, die als Monolith implementiert wurden, in Microservices umzustrukturieren [FBZW19].

Die Firma L-mobile[1] hat sich dazu entschieden, die etablierte, webbasierte Service/Sales-Applikation von einer monolithischen in eine Microservices-Architektur umzustrukturieren. Die Service/Sales-Applikation orientiert sich an einem Customer-Relationship-Management-System (CRM). Mit dieser Applikation können u. a. verschiedene Auftragsarten für Serviceobjekte organisiert und Serviceeinsätze geplant werden. Die Applikation wird als Software-as-a-Service (SaaS) angeboten, kann aber auch auf Kundenservern betrieben werden. Die Applikation kann als Standalone-Applikation eingesetzt, jedoch auch in Verbindung mit einem Enterprise-Resource-Planning-System (ERP), wie proAlpha, SAP, Sage, etc., betrieben werden. Durch Customizing können kundenspezifische Anforderungen als Erweiterungen der Applikation implementiert werden.

[1] https://l-mobile.com/

Ziel dieser Arbeit ist es, eine (Teil-) Umstrukturierung der monolithischen Service/Sales-Applikation von L-mobile in Microservices durchzuführen. Für diese (Teil-) Umstrukturierung soll das Framework für Microservices-Migrationen (FMM) [FBH+22, Ins23] des Institute of Software Engineering (ISTE), Abteilung Empirical Software Engineering (ESE), der Universität Stuttgart, und das dafür implementierte Architektur Refactoring Helper (ARH) Tool verwendet werden. Dies umfasst etablierte Methoden durch das FMM zu extrahieren, sowie eine umfassende Architektur und einen Prozess zu beschreiben, um den existierenden Monolithen in Microservices umzustrukturieren. Weiterhin soll mit dem FMM ein Prototyp dieser Microservices-Applikation implementiert werden. automatisierte Tests gegen den Monolithen und Code-Reviews werden durchgeführt, um das Resultat der Umstrukturierung zu bewerten. Vor der Implementierung erfolgt eine Analyse, aus der resultiert, welcher Teil der Applikation sich am besten für eine prototypische Umstrukturierung von einem Monolithen in Microservices eignet. Diese Analyse wird auf der Basis der vom FMM vorgeschlagenen Refactoring-Verfahren durchgeführt.
Während der Umstrukturierung der Service/Sales-Applikation werden folgende Forschungsfragen verfolgt:

- **FF1**: Welche Frameworks werden in der wissenschaftlichen Literatur für die Umstrukturierung eines Monolithen in Microservices vorgeschlagen und welche Herangehensweisen bzw. Prozesse verfolgen diese?
- **FF2**: Inwieweit eignet sich das FMM des ISTE ESE, um den L-mobile Monolithen in Microservices umzustrukturieren?
- **FF3**: Welche Refactoring-Verfahren und Migrationsverfahren eignen sich am besten für die Dekomposition des L-mobile Monolithen in eine Microservices-Architektur?
- **FF4**: Welche allgemeine und L-mobile Monolith-spezifische Herausforderungen treten bei der Umstrukturierung auf?

Die Methoden die für die Beantwortung dieser Forschungsfragen eingesetzt werden, sind ausführlich in Kapitel 3 beschrieben. Um Forschungsfrage 1 zu beantworten, wird eine Literaturrecherche durchgeführt. Für die Beantwortung, welche allgemeinen Probleme bei einer Migration auftreten können, Teil der Forschungsfrage 4, wird ebenfalls eine Literaturrecherche durchgeführt. Forschungsfrage 3 wird durch die Anwendung des FMM auf die Service/Sales-Applikation von L-mobile beantwortet. Während der Anwendung des FMM werden strukturierte Feldnotizen erstellt und im Anschluss ausgewertet. Mit dieser Auswertung der Feldnotizen wird Forschungsfrage 2 und der zweite Teil der Forschungsfrage 4, welche L-mobile spezifischen Probleme auftreten, beantwortet.

In dieser Arbeit wird das FMM angewendet, um eine Planung und Architektur für die Migration der Service/Sales-Applikation von L-mobile zu erstellen. Außerdem wird ein Proof-of-Concept für die Migration erstellt, in dem ein Microservice der Applikation extrahiert und getestet wird. IT-Infrastrukturthemen, wie z. B. die Verwendung von Messaging und Containern, wird in dieser Arbeit nicht betrachtet.

Diese Arbeit ist wie folgt strukturiert: In Kapitel 2 werden die Grundlagen für eine Migration von Monolithen in eine Microservices-basierte Architektur erläutert. Dafür wird beschrieben,

was Monolithen und Microservices sind, was eine Migration von monolithischen Applikationen in eine Microservices-basierte Architektur ist und welche Gründe für, oder gegen eine solche Migration sprechen. In diesem Kapitel wird auch das FMM des ISTE ESE der Universität Stuttgart vorgestellt. Anschließend wird eine Literaturrecherche zu einer Framework basierten Migration durchgeführt. Kapitel 3 beschreibt die Methodik der Arbeit. Es werden wichtige Informationen zur Literaturrecherche, zur Migrationsstrategie und zu strukturierten Feldnotizen angegeben. Außerdem befindet sich hier die Abgrenzung der Arbeit. Die Durchführung der Phasen des FMM am L-mobile Monolithen wird in Kapitel 4 dargestellt. Dieses Kapitel ist in die drei verschiedenen Phasen des FMM unterteilt. Um sicher zu stellen, dass die Codequalität und die Datenverarbeitung nach der Migration, im Vergleich zum Monolithen, erhalten wurde, werden die Code-Reviews und die Durchführung der automatisierte Tests in Kapitel 5 aufgeführt. Die Erfassung und Auswertung der strukturierten Feldnotizen, zur Evaluierung des Migrationsprozesses, wird in Kapitel 6 veranschaulicht. In Kapitel 7 werden die Ergebnisse im Bezug auf die Forschungsfragen diskutiert. Die Arbeit wird mit Kapitel 8 durch ein Fazit und einen Ausblick abgeschlossen.

1.1. Aufgaben

In dieser Arbeit werden folgende Aufgaben verfolgt.

- Mit den drei Phasen des FMM eine (Teil-) Umstrukturierung der L-mobile Applikation durchführen.
- Ausarbeiten, welcher Teil der Anwendung sich für eine prototypische Umstrukturierung in Microservices eignet.
- Code-Reviews und automatisierte Tests durchführen.
- Strukturierte Feldnotizen erstellen und auswerten.

1.2. Abgrenzung

Im Rahmen dieser Arbeit soll die Planung und (Teil-)Durchführung einer Migration auf Basis dieses FMM auf ihre praktische Eignung und Anwendbarkeit hin untersucht werden. Dabei liegt der Fokus auf der Umstrukturierung aus Architektursicht und nicht auf IT-Infrastruktur-Themen. Weiterhin werden keine Performanzanalysen durchgeführt. Es wird mit dem FMM eine Auswahl an Methoden zur Umstrukturierung von Monolithen in Microservices recherchiert. Eine Gegenüberstellung der Methoden wird nur im Rahmen der Service/Sales-Applikation von L-mobile durchgeführt. Weitere Frameworks/Tools, die bei der Umstrukturierung unterstützen, werden nur vorgestellt, es wird keine Gegenüberstellung dieser Frameworks/Tools durchgeführt.

2. Grundlagen und verwandte Arbeiten

Um die Grundlagen für diese Arbeit zu erörtern, ist dieses Kapitel wie folgt strukturiert. In Kapitel 2.1 wird die monolithische Architektur von Software-Applikationen charakterisiert. Anschließend wird in Kapitel 2.2 die Microservices-Architektur charakterisiert, bevor Kapitel 2.3 die Grundlagen einer Migration von Monolithen in Microservices beschreibt und Gründe für und gegen eine Migration wiedergibt. In diesem Kapitel werden ebenfalls die Herausforderungen die eine Migration mit sich bringen kann, beleuchtet. Kapitel 2.3.1 beantwortet damit den ersten Teil der Forschungsfrage 4, Welche Herausforderungen treten bei der Umstrukturierung auf (allgemein und L-mobile Monolith-spezifisch)? In Kapitel 2.4 wird das FMM des ISTE ESE vorgestellt. In Kapitel 2.5 wird durch die Literaturrecherche zu Framework- und Tool-basierten Migrationen von Monolithen in Microservices die Forschungsfrage 1 beantwortet und dieses Kapitel damit abgeschlossen.

2.1. Charakterisierung von Monolithen

In der Domäne der Softwareentwicklung erhält eine Anwendung/Applikation, die aus unterschiedlichen Bestandteilen besteht, welche zu einer einzigen Einheit zusammengefügt werden, die Bezeichnung Monolith [FL14, GZ20, KMM18, PMA19]. Ponce et al. erwähnen, dass die einzelnen Komponenten eines Monolithen eine enge Verbindung zueinander aufweisen und dass sie nicht eigenständig ausgeführt werden können. Nach Ponce et al. liegt dies daran, dass zur Bearbeitung einer Anfrage lediglich ein einziger Prozess ausgeführt wird, der die komplette Logik abarbeitet [PMA19]. Das hat den Nachteil, dass ein Fehler in einer Komponente die gesamte Applikation zum Absturz bringen kann, dadurch wird die Applikation weniger zuverlässig und weniger verfügbar [GZ20, PMA19].
Bei Aktualisierungen muss der gesamte Monolith neu installiert werden, dadurch werden auch Komponenten, in denen keine Änderungen vorgenommen wurden während des Updates nicht verfügbar sein, dies ist ein weiterer Nachteil von Monolithen [FL14, GZ20, PMA19]. Außerdem lassen sich dadurch einzelne Komponenten, bei z. B. vielen gleichzeitigen Zugriffen, nicht individuell skalieren [FL14, PMA19]. Der gesamte Monolith wird in diesem Fall skaliert [FL14, GZ20, PMA19].
Monolithen können in Bezug auf Wartung und Entwicklung zunehmend komplex werden, was als Nachteil angesehen wird [GZ20, KMM18]. Dies liegt daran, dass es bei einer umfangreicheren Codebasis schwieriger wird den richtigen Ort für Änderungen an bestehenden Features zu identifizieren und neue Features zu Implementieren [KMM18, PMA19]. Dadurch

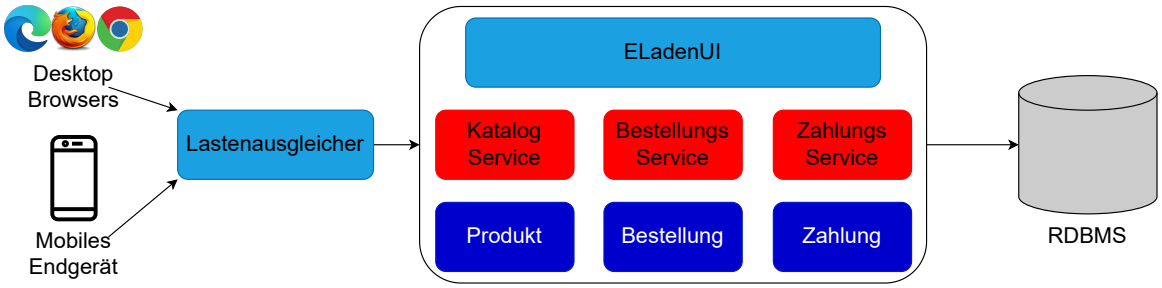


Abbildung 2.1.: Ein Beispiel für eine monolithische Applikation, übersetzt und adaptiert aus Gos und Zabierowski [GZ20]

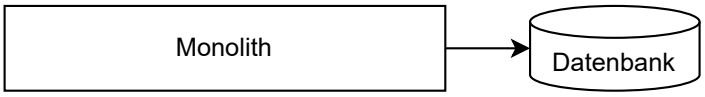
sind Monolithen schwer zu warten, es dauert lange um neue Features zu entwickeln und Entwickler haben eine niedrige Produktivität [PMA19].

Monolithen haben aber auch Vorteile. Monolithen sind anfangs, solange die Codebasis klein ist, einfach und schnell zu entwickeln, zu testen, zu debuggen, zu überwachen, zu verteilen und einfach zu betreiben [BOP22, GZ20, KMM18]. Weil mit Monolithen am Anfang schnellere Fortschritte bei der Entwicklung erzielt werden können als bei Microservices, ist es nach Kalske et al. heutzutage der Standard-Weg, mit einer monolithischen Architektur anzufangen, wenn eine neue Applikation implementiert werden soll [KMM18].
Auch Martin Fowler sieht es vor, erst wenn eine Applikation zu komplex wird, um sie noch als Monolith zu verwalten, Microservices in Erwägung zu ziehen [Fow15]. Wegen dieser Vorteile sollte die Mehrheit der Applikationen heutzutage noch als Monolithen implementiert sein [Fow15, KMM18].

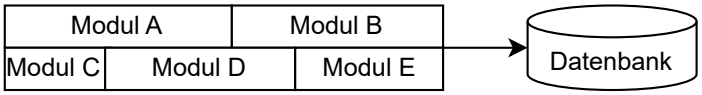
Alle Komponenten eines Monolithen interagieren mit einer Datenbank (DB), in der die Daten persistent ablegt werden können [FL14]. Um Zugriffskonflikte, durch gleichzeitige Lese- und Schreibzugriffe, auf die Datenbank zu vermeiden und die Konsistenz der Daten zu garantieren, sollten Transaktionen verwendet werden [FL14, Set81]. Dadurch werden die Komponenten auch zeitlich gekoppelt [FL14].

Ein Beispiel für einen Monolithen, welches von Gos und Zabierowski [GZ20] angegeben wurde, ist in Abbildung 2.1 dargestellt. In Abbildung 2.1 ist die typische Struktur einer Unternehmens-Applikation erkennbar [FL14]. Die vorhandenen Komponenten umfassen die Benutzerschnittstellen auf der Client-Seite, repräsentiert durch Internetbrowser und Smartphones. Des weiteren gibt es einen Lastenausgleicher, der die Skalierung des Monolithen überwacht. Eine serverseitige Applikation stellt den Monolithen dar, während eine Datenbank, dargestellt durch ein relationales Datenbankverwaltungssystem (RDBMS), zur Verfügung steht [FL14].

Nach Newman können Monolithen in mindestens drei Kategorien unterteilt werden, das Ein-Prozess-System, zu sehen in Abbildung 2.2a, das Black-Box-System und der verteilte Monolith [New20]. Black-Box-Systeme sind Systeme von Fremdherstellern. Newman beschreibt einen verteilten Monolithen als System, welches sich aus mehreren Servern zusammensetzt,



(a) Beispiel für einen Monolithen, aus Newman [New20].



(b) Beispiel für einen modularen Monolithen, aus Newman [New20].

Abbildung 2.2.: Beispiele für einen Monolith und einen modularen Monolithen, Beispiele aus Newman [New20].

die gemeinsam verteilt werden müssen. Nach Newman haben verteilte Monolithen die Nachteile von verteilten Systemen und von Ein-Prozess-Monolithen, ohne dabei die Vorteile beider Ansätze in Anspruch zu nehmen. Ein Ein-Prozess-Monolith, beschreibt der Autor, besteht aus einem System, welches als ein Prozess verteilt und ausgeführt wird. Um die Robustheit und Skalierbarkeit eines Ein-Prozess-Monolithen zu verbessern, besteht, nach Newman, die Möglichkeit mehrere Instanzen dieses Monolithen parallel zu betreiben.
Newman beschreibt eine Unterart des Ein-Prozess-Monolithen, den modularen Monolith, zu sehen in Abbildung 2.2b. Bei dieser Art, beschreibt Newman, besteht der Monolith (der eine Prozess) aus separaten Modulen. Er beschreibt, dass an diesen Modulen unabhängig voneinander gearbeitet werden kann, wenn das System jedoch verteilt und in Betrieb genommen werden soll, müssen alle Module wieder kombiniert werden, um als eine Einheit verteilt zu werden. Mit einem modularen Monolithen kann eine höhere Parallelität bei der Arbeit erzielt werden [New20].
Eine Möglichkeit den Monolithen in Module zu unterteilen, ist die Unterteilung in Front-End und Back-End Module [GFSP21]. Dabei enthalten nach Gonçalves et al. die Front-End Module die Schnittstellen zur Benutzerinteraktion und die Back-End Module sind gemäß des Domänen-Modell aufgeteilt. Gonçalves et al. beschreiben für die Kommunikation zwischen den Modulen Schnittstellen, welche zur Anforderung und Abonnierung dienen [GFSP21].
Faustino et al. schreiben, dass es wichtig bei der Implementierung eines modularen Monolithen ist, dass es zwischen den Modulen keine zirkuläre Abhängigkeit geben darf [FGPS22]. Außerdem sollte nach den Autoren die Kommunikation zwischen den Modulen gering gehalten werden, um Performance Probleme, welche durch zu häufige inter-Modul-Kommunikation entstehen können, zu vermeiden [FGPS22].

2.2. Charakterisierung von Microservices

Zitat: „Der Begriff "Microservices-Architektur" ist in den letzten Jahren entstanden, um eine bestimmte Art zu beschreiben, Software-Applikationen als Suiten von unabhängig einsetzbaren Diensten zu entwerfen.“ ~Martin Fowler [FL14]

Im Microservices-Architekturstil werden Services einer Applikation, welche jeweils in einem eigenen Prozess laufen, zu einem Verbund zusammengefasst [FL14, GZ20]. Nach Fowler und Lewis zielt dieser Architekturstil darauf ab, die Applikation so entkoppelt und so zusammenhängend wie möglich zu gestalten [FL14]. Lose Kopplung ist eine inhärente Eigenschaft von Microservices [KMM18]. Jeder Service besitzt nach Fowler und Lewis seine eigene Domänenlogik [FL14]. Sie beschreiben Microservices so, dass sie eher als Filter fungieren. Als Beispiel geben sie ein Service an, welcher eine Anfrage empfängt, seine Logik darauf anwendet und eine Antwort erzeugt [FL14]. Nach Blinowski et al. enthält jeder Service seine eigenen Eingabe-/Ausgabefunktionen, Geschäftslogik und Backend Funktionalitäten [BOP22, GZ20].
Kalske et al. beschreiben, dass die einzelnen Services in einer Microservices-Applikation dem Single Responsibility Principle (SRP) folgen, das bedeutet, dass sich jeder Service nur auf eine Funktionalität fokussiert [KMM18]. Sie erklären, dass es durch dieses Prinzip funktiosspeziefisch ersichtlich ist, wo der Code lokalisiert ist. Darüber hinaus ermöglicht dieses Prinzip eine klare Abgrenzung zwischen den einzelnen Services anhand von Funktionsdifferenzierung. Weiterhin können, nach Kalske et al., diese Microservices durch die Entkopplung individuell skaliert und verteilt werden.
In der Regel hat jeder Service, in einer Microservices-Architektur, seine eigene Datenbank [BOP22, FL14]. Nach Blinowski et al. ist es aber auch möglich, dass sich mehrere Services ein Backend teilen [BOP22]. Nach Fowler und Lewis sehen MSAs eine Transaktionslose-Koordination unter den Services vor. Sie erklären, dass eine Konsequenz daraus ist, dass die Daten in einer MSA nur das letztendliche Konsistenzlevel erreichen können [FL14].
Nach Fowler und Lewis wird die Kommunikation der Services untereinander, durch leichtgewichtige Mechanismen realisiert. Für diese Kommunikation können Representational State Transfer (REST) Application Programming Interface (API) und leichtgewichtiges Messaging verwendet werden [BOP22, FL14].
Nach Fowler und Lewis können in einer MSA mehrere Prozesse, die immer gemeinsam entwickelt sowie eingesetzt werden, zu einem Service vereinigt werden [FL14]. Die verschiedenen Services einer Microservices-Applikation können mit verschiedenen Technologien, in unterschiedlichen Programmiersprachen und verschiedenen Datenbanksystemen implementiert werden [FL14, Seb+17]. Das sind ein paar Gründe, weshalb verschiedene Services von verschiedenen, unabhängigen Teams entwickelt und installiert werden können [PMA19].
Da Microservices-Applikationen, im Gegensatz zu Monolithen, in verschiedenen, unabhängigen Prozessen betrieben werden und nicht aneinander gebunden sind, lassen sich Microservices unabhängig voneinander skalieren, verteilen und haben eine höhere Fehlertoleranz [FL14, KMM18, PMA19, Thö15].

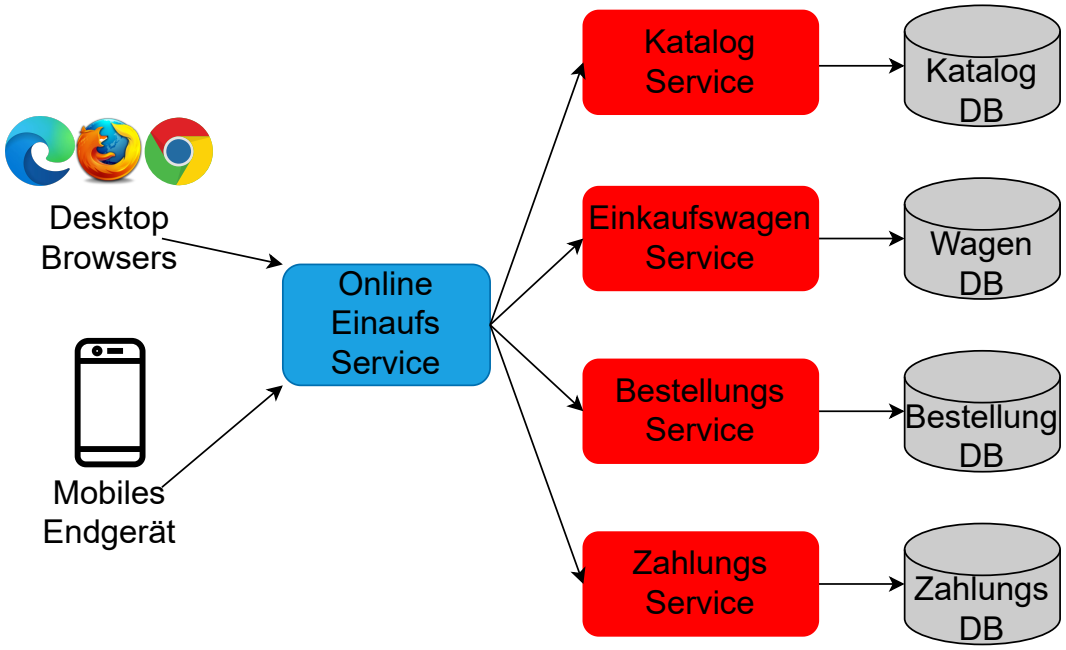


Abbildung 2.3.: Beispiel für eine Microservices-Applikation, übersetzt und adaptiert aus Gos und Zabierowski [GZ20]

Ein Beispiel für Microservices, von Gos und Zabierowski [GZ20], ist in Abbildung 2.3 dargestellt. Es existiert eine Benutzerschnittstelle in Form eines Online-Einkaufsservice, auf den die Benutzer über Internetbrowser oder ein mobiles Endgerät zugreifen können. Diese Schnittstelle greift auf unterschiedliche Services zu, wie den Katalog-Service, den Einkaufswagen-Service, den Bestell-Service und den Zahlungs-Service. Es ist erkennbar, dass jeder dieser Services unabhängig von den anderen Services arbeitet und über eine eigene Datenbank verfügt.

Die hier aufgelisteten Vorteile von Microservices wurden den Veröffentlichungen [FL14, GZ20, KMM18, PMA19] entnommen. Zu den Vorteilen von Microservices zählen u. a. ihre leichte Wartbarkeit und ihre verständliche Struktur für Softwareentwickler. Dies resultiert aus der Entwicklung jeder Kernfunktionalität in jeweils einem separaten Modul [GZ20]. Gegenüber Monolithen ist die Zuverlässigkeit des kompletten Systems höher, da durch die Unabhängigkeit der Services ein Fehler nur den fehlgeschlagenen Service betrifft. Die weiteren Services bleiben ansprechbar und einsatzfähig. Darüber hinaus wurde beschrieben, dass die Verfügbarkeit höher ist, da eine Aktualisierung nur den geänderte Service betrifft und die weiteren Services weiter arbeiten können. Die Skalierung wird als einfacher beschrieben, da jeder Service unabhängig skaliert werden kann [FL14, GZ20, KMM18, PMA19].

Aber auch Microservices haben Nachteile. Sie sind komplizierter zu betreiben und die Autonomie ist eine große Herausforderung [GZ20]. Fowler und Lewis führen außerdem auf, dass Aufrufe entfernter Prozesse aufwendiger in der Instantiierung sind als intra-Prozesskommunikation [FL14]. Sie merken weiterhin an, dass Services einer Microservices-

Applikation, ohne dass es einem Anwender auffällt, ausfallen können. Demnach ist es laut den Autoren erforderlich, dass die Betreiber von Microservices-Applikationen jederzeit dazu bereit sind, Ausfälle so schnell wie möglich zu detektieren und nach Möglichkeit automatisch den ausgefallenen Service wiederherzustellen. Das kann, nach Fowler und Lewis, durch Echtzeitüberwachung bewerkstelligt werden. Dieser zusätzliche Aufwand und Komplexität, um Microservices tolerant für den Ausfall eines, oder mehrerer, Services zu designen, wird von Fowler und Lewis als Nachteil gegenüber Monolithen beschrieben [FL14].

2.3. Migration von Monolithen in Microservices

In diesem Kapitel wird als erstes auf Software Migrationen im Allgemeinen eingegangen. Im zweiten Teil dieses Kapitels wird dargestellt, wie die Migration eines Monolithen in Microservices in der wissenschaftlichen Literatur beschrieben wird.

Im Allgemeinen wird Softwaremigration als eine Überführung eines Softwaresystems in eine neue technologische Umgebung, ohne dabei ihre Funktionalitäten zu verändern, beschrieben [GW05]. Gimnich und Winter beschreiben darüber hinaus, dass Migrationen durch Anforderungsdefinitionen beschrieben werden und dass es sich bei der Durchführung um eine reine technische Transformation handelt. Sie führen auf, dass die häufigsten Gründe für Migrationen üblicherweise geänderte Anforderungen an das bestehende System sind.
Dadurch können, laut Gimnich und Winter, Änderungen der Hardwareumgebung, der Laufzeitumgebung, der Softwarearchitektur und der Entwicklungsumgebung ausgelöst werden. Sie merken an, dass abhängig vom Migrationsziel Daten, Benutzerschnittstellen und Programme betroffen sind, welche durch Datenmigration, Benutzerschnittstellen-Migration und Programm-Migration in die neuen Strukturen transformiert werden.
Architektur-Migrationen, wie sie in dieser Arbeit durchgeführt wird, wird als eine grundlegende Änderung der System-Struktur bezeichnet [GW05].
Sie beschreiben, dass durch Wiederholung von Testfällen sichergestellt werden kann, ob nach erfolgreicher Migration die System-Funktionalitäten des Altsystems erhalten geblieben sind [GW05]. Nach Gimnich und Winter besteht eine Migration aus folgenden Schritten:

- Migrationsstrategie festlegen
- Zielumgebung definieren
- Unterschiede Analysieren
- Migrationsumfang festlegen
- Transformationen spezifizieren
- Transformation umsetzen
- Migriertes System übergeben
- Qualifizierung der Mitarbeiter
- Qualität sichern

Die Migrationsstrategie kann, nach Gimnich und Winter, entweder eine Neuentwicklung, eine Kapselung, in der Zugriffsschnittstellen im neuen System auf das Altsystem bereitgestellt werden, oder eine Konversion, also eine direkte Übertragung in die Zielumgebung sein.
Sie beschreiben, dass für die Zielumgebung die Hardwareumgebung, die Systemsoftware, die Entwicklungsumgebung, die Softwarearchitektur, die Daten, die Benutzerschnittstellen und die Programme noch vor der Durchführung der Migration, definiert werden müssen.
Nach Gimnich und Winter wird bei der Analyse der Unterschiede das Altsystem hinsichtlich der Migrationsfähigkeit bewertet und Unterschiede zum Zielsystem definiert.
Sie beschreiben, dass für den Migrationsumfang definiert werden sollte, wie umfangreich das Funktionsspektrum, die Qualität und die Wiederverwendbarkeit relevanter Komponenten des Altsystems gegenüber des Zielsystems ausfällt. Außerdem werden, laut den Autoren, qualitative und quantitative Analysen am Altsystem durchgeführt. Dabei sollte, nach Gimnich und Winter, eine hohe Wiederverwendung der Zielkomponenten angestrebt werden.
Die nächsten zwei Schritte von Gimnich und Winter beinhalten, dass Transformationen für die Migration erst spezifiziert und umgesetzt werden. Bei der Übergabe des Funktionsumfangs in das Zielsystem wird zwischen einer Integration auf einen Schlag (Big-Bang-Integration) und einer inkrementellen Integration in mehreren Schritten (sanften Integration) unterschieden [GW05, HRJ+04].
Außerdem beschreiben Gimnich und Winter, dass eine Migration noch zusätzlich fordert, dass Mitarbeiter, die an der Migration beteiligt sind, sowie Mitarbeiter und Kunden, welche mit dem migrierten System arbeiten, entsprechend geschult werden.
Schlussendlich folgt laut Gimnich und Winter die Qualitätssicherung, in der sichergestellt wird, dass zwischen Alt- und Zielsystem die funktionale Gleichwertigkeit gewährleistet ist.

Wie Ponce et al. beschreiben, ist die Durchführung einer Migration von Monolithen in Microservices, kein trivialer Prozess [PMA19]. Wie von Bajaj et al. [BBGG21] beschrieben, existieren in der wissenschaftlichen Literatur viele verschiedene technische Ansätze um einen Monolithen in Microservices zu migrieren. Weiterhin beschreiben sie, dass es keinen Ansatz gibt, der für alle Monolithen anwendbar ist [BBGG21]. Die Herausforderung liegt darin den richtigen Ansatz für die Migration in Microservices auszuwählen, da Monolith miteinander nicht vergleichbar sind [BBGG21, CM22]. Im Folgenden wird zuerst beschrieben, in wie vielen Schritten Migrationen von einem Monolithen in Microservices durchgeführt werden können. Anschließend werden die von Ponce et al. [PMA19] erarbeiteten Techniken kurz vorgestellt. Zum Schluss dieses Kapitels wird das *Stragler Fig* Pattern präsentiert.

Für die Migration eines Monolithen in Microservices existieren verschiedene Vorgehensweisen, die sich u. a. in der Anzahl der Schritten für die Migration unterscheiden. Sebastian und Sarita [Seb+17] migrierten einen Monolithen in Microservices in zwei Schritten. Im ersten Schritt separieren sie Backend und Frontend in verschiedene Schichten. In Schritt zwei sehen sie die Entkopplung der Module in Services vor. Dabei wird bei ihnen nicht jedes Modul als Service implementiert. Sie definieren zwei Faktoren, mit denen die Module in Kategorien unterteilt werden, welche dann als Microservices implementiert werden können: i) Als ersten Faktor sehen sie vor, dass das Modul eine wichtige Rolle in der Geschäftslogik haben muss sowie häufig geändert werden muss und ii) sehen sie die Entscheidungen basierend auf Ressourcen-

Anforderungen vor. Die Kommunikation zwischen Frontend und Backend funktioniert bei ihnen mit einer REST API [Seb+17].
Eine Migration, von einem Monolithen in Microservices, erfolgt bei Fritzsch et al. [FBH+22] in drei Schritten. Im ersten Schritt wird eine Entscheidung für oder gegen eine Migration getroffen. In Schritt zwei wird eine adäquate Migrationsstrategie definiert. Dabei wird ein Entwicklungsprozess auf der Basis verschiedener Typen der Software-Modernisierung definiert. Außerdem wird in diesem Schritt ein Service-Identifikationsansatz ausgewählt [FBH+22].
Häufig wird bei der Microservices-Migration zwischen Greenfield und Brownfield Entwicklung unterschieden [FBH+22, FBZW19, LXR+22]. Dabei steht Greenfield für eine Neuentwicklung und Brownfield für die Transformation eines Altsystems in eine MSA [BBGG21, LXR+22]. Die Brownfield Strategie kann dabei noch in Rebuild und Refactor Typen unterteilt werden [FBH+22]. Fritzsch et al. beschreibt als dritten Migrationsschritt die Identifizierung der Microservices, die Erstellung der Architektur und ihre Implementierung vor [FBH+22].
Viele Veröffentlichungen sehen die Migration eines Monolithen in Microservices in zwei Schritten vor. Im ersten Schritt werden die Microservices aus dem vorhandenen Monolithen identifiziert um eine MSA zu generieren [SSB+20, ZLD+20, ZSS+21]. Im zweiten Schritt werden Quellcode-Transformationen durchgeführt, um das vorhandene System in die vorher erarbeitete MSA zu überführen [SSB+20, ZLD+20, ZSS+21]. Selmadji et al. [SSB+18] haben vor der Service-Identifizierung noch den Schritt zum Aufbau des Systemverständnisses.
Die Methode von Fritzsch et al. [FBH+22] enthält die zwei Schritte welche in anderen Veröffentlichungen [SSB+20, ZLD+20, ZSS+21] für die Migration vorgeschlagen werden. Außerdem enthält sie ebenfalls den Schritt zum Aufbau des Systemverständnisses Selmadji et al. [SSB+18]. Deshalb, und weil in dieser Arbeit das FMM verwendet werden soll, eignet sich das Vorgehen von Fritzsch et al. [FBH+22] für diese Arbeit am besten.

Ponce et al. [PMA19] haben in ihrem Rapid Review folgende Techniken, für die Migration von Monolithen in Microservices, in der wissenschaftlichen Literatur gefunden:

- Modell getrieben
- Statische Analyse
- Dynamische Analyse

Sie beschreiben, dass diese Methoden einzeln oder miteinander verwendet werden können [PMA19]. Welche dieser Methoden, oder welche Mischung dieser Methoden, für die Migration der Service/Sales-Applikation von L-mobile zur Anwendung kommen, wird sich durch die Ausführung des FMM herausstellen, da wie Fritzsch et al. [FBH+22] beschreiben, das FMM mehrere Methoden für die Migration und Service-Identifikation vorschlägt.

Die Methoden des *Strangler* Patterns [YM20], auch *Strangler Fig* Applikation [Fow04] genannt, ist eine Art einen Monolithen in Microservices zu migrieren. Fowler beschreibt das *Strangler* Pattern so, dass dabei ein neues System über die Kanten des alten Systems entwickelt wird [Fow04]. Das bedeutet, dass neue Funktionen dem alten System als Microservices hinzugefügt, oder bestehende Komponenten aus dem System herausgezogen und als Microservices dem System zur Verfügung gestellt werden, bis das alte System irgendwann komplett ersetzt wurde [LML20]. Dabei bleibt das alte System im Einsatz und wird nach und nach durch

Microservices ersetzt, bis es schlussendlich komplett erdrosselt (strangled) wurde [Fow04, YM20]. Wie sich in dieser Arbeit herausgestellt hat, eignet sich das *Strangler* Pattern für die Migration der Service/Sales-Applikation von L-mobile, da für den Proof of Concept (PoC) nur ein Microservice extrahiert wurde und es das *Strangler* Pattern, nach Yoder et al. [YM20] beschreiben, erlaubt den Monolithen mit dem Microservice zu betreiben.

2.3.1. Herausforderungen einer Migration

In diesem Abschnitt werden allgemeine Herausforderungen einer Migration und im Speziellen einer Migration eines Monolithen in Microservices beleuchtet, die in der wissenschaftlichen Literatur beschrieben werden.

Nach Furda et al. [FFZ+17] sind drei Herausforderungen einer Migration von Monolithen in MSAs die Mehrmandantenfähigkeit, die Zustandsfestigkeit und die Datenkonsistenz. Da, nach Furda et al., Altsysteme oft nur für einen Mandanten designed wurden ist die Migration in mehrmandantenfähige Microservices oft schwierig. Sie beschreiben, dass Altsysteme oft Zustandsabhängig sind, idealerweise sind Microservices jedoch zustandslos. Microservices werden, nach Furda et al., oft dazu verwendet, die Skalierung und die Performance zu verbessern. Sie beschreiben, dass dafür oft mehrere Instanzen eines Microservice parallel nebeneinander betrieben werden. Nach den von Fowler und Lewis [FL14] postulierten Charakteristiken für Microservices hat jeder dieser Instanzen seinen eigenen Datenspeicher, was eine Konsistenz der Daten sehr schwierig macht [FFZ+17].

Velepucha und Flores [VF21] haben eine Literaturrecherche durchgeführt. Sie geben an, dass die Auswahl eines geeigneten Tools, Frameworks, Methode oder Modell für die Migration eine Herausforderung ist. Eine weitere Herausforderung die von ihnen angegeben werden sind die Änderung der hierarchischen Organisationsstruktur, so dass sie mit Microservices funktioniert. Die Autoren Velepucha und Flores stellen die Abhängigkeit von Frameworks als Herausforderung dar. Dabei beziehen sie sich u. a. auf die Bereitstellung der Konsistenz der Daten. Außerdem geben sie noch an, dass die Priorisierung der Funktionalitäten für die Migration, die Anpassung der Applikation um Logs zu generieren, eine adäquate Analyse des zu migrierenden Systems durchzuführen und gute Entscheidungen zu treffen weitere Herausforderungen der Migration eines Monolithen in Microservices sind [VF21].

Ponce et al. geben wieder, dass eine der Hauptherausforderungen bei einer Migration eines Monolithen in Microservices die Migration der Datenbank ist [PMA19]. Neben der Datenbankmigration haben die Autoren in ihrem Literaturreview noch die Zerlegung von Geschäftsfähigkeiten, den Bedarf an fähigen Entwicklern und das Resourcenmanagement als Herausforderungen identifiziert. Weiterhin geben die Autoren die Expertenbetrachtung der vorgeschlagenen Microservices bevor sie implementiert werden, komplexe Umgebungseinstellungen, die Schwierigkeit Transaktionen zu implementieren und die Adaption der neuen Art der Arbeit als Herausforderungen an [PMA19].

Die Service-Identifizierung, bzw. den richtigen Servicecut zu finden, ist nach Fritzsch et al. [FBWZ19] eine der technischen Hauptherausforderungen. Die Integration der Kommunikation mit externen Teilnehmern ist, laut Fritzsch et al., ebenfalls eine identifizierte Herausforderung. Das Erreichen einer gewissen Fehlertoleranz, die beiden Systeme (Alt- und Neusystem) während der Migration synchron zu halten, das Erreichen von Fehlertoleranz zwischen den Services und das Orchestrieren der Microservices sind weitere technische Herausforderungen die von Fritzsch et al. identifiziert wurden. Neben der reinen Implementierungsherausforderungen existieren auch Integrationsherausforderungen. Die von Fritzsch et al. identifizierten Integrationsherausforderungen sind, das Integrieren von externen Systemen, das Aufsetzen der neuen Umgebung (z. B. Kubernetes [1]), der Wechsel von einer zentralen Datenbank in eine streambasierte Architektur, das parallele Betreiben beider Systeme während der Migration und der Wechsel von einem zustandsfesten zu einem zustandslosen System. Neben den technischen Herausforderungen identifizierten sie auch Herausforderungen welche die menschliche Ressource betreffen. Dazu gehören die Verteilung der Aufgaben an die Entwickler (bei großen Firmen), der Wissenstransfer zwischen allen Teilnehmern und das Erwerben der nötigen Erfahrung (bei kleinen Teams) [FBWZ19].

Nach Fowler ist die größte Schwierigkeit bei der Migration von Monolithen in Microservices das Ändern der Kommunikationsmuster [FL14].

Zusammenfassend sind die wesentlichen, in der wissenschaftlichen Literatur genannten, Herausforderungen bei einer Migration in Microservices das Transformieren eines zustandsabhängigen in ein zustandsloses System [FBWZ19, FFZ+17], die Migration der Datenbank [FFZ+17, PMA19] und die Fähigkeit des Teams Microservices zu entwickeln [FBWZ19, FL14, PMA19].

2.3.2. Gründe für eine Migration

Monolithische Applikationen werden, nach Fritzsch et al., mit der Zeit zu komplex, so dass Entwickler und Architekten den vollständigen Überblick über die Details aller Komponenten des Monolithen, verlieren [FBZW19]. Daraus resultiert, dass monolithische Systeme schwer zu warten sind und es schwieriger wird neue Features und Technologien einzubringen [FBZW19, GZ20, KMM18, PMA19, Thö15]. Nach Sebastian und Sarita können Microservices-Architekturen dabei helfen die Zeit, um auf den Markt zu reagieren, zu verkürzen [Seb+17]. Sie beschreiben außerdem, dass es sich mit einer Microservices-Applikation erreichen lässt, ein qualitativ hochwertiges Produkt an den Kunden auszuliefern [Seb+17]. Ein weiterer Grund für eine Migration ist, dass die Skalierung eines Monolithen schwierig sein kann, wie Ponce et al. anmerken [PMA19]. Nach Fritzsch et al. [FBZW19] ist es für einen Monolithen meistens nicht möglich, auf dem Modul-Level zu skalieren. Sie merken an, dass daher der gesamte Monolith dupliziert werden muss, was meistens ineffizient ist [FBZW19]. Diese Art der Skalierung wird, u. a. von Ponce et al. Horizontale-Skalierung genannt [PMA19]. Nach Fowler sollte der

[1] https://kubernetes.io/

wichtigste Faktor aber sein, dass die Fähigkeiten der Entwicklerteams für eine Migration in Microservices sprechen [Fow15].

2.3.3. Gründe gegen eine Migration

Nach Fowler macht es der extra Aufwand, für wenig komplexe Applikationen, um Microservices zu verwalten, unnötig von einem Monolith auf Microservices umzusteigen, da in diesem Fall die Monolithen performanter sind [Fow15]. Ein Beispiel hierfür ist das Istio Projekt, beschrieben von Mendonça et al. [MBMR21]. In diesem Projekt kam es laut den Autoren nie zu den Benutzerfällen, in denen eine MSA von ihren Vorteilen profitiert hätte, und so blieben nur die Nachteile einer MSA übrig, weshalb die Entwickler des Istio Projekts ihre MSA in einen Monolith migriert haben [MBMR21]. Ein weiteres Projekt, in dem sich ein Monolith als geeigneter als Microservices herausgestellt hat, weshalb eine Migration von Microservices zurück in einen Monolithen durchgeführt wurde, ist der „Prime Video Audio/Video monitoring Service“ [Kol23]. Nach Fowler sollte der wichtigste Faktor aber sein, dass die Fähigkeiten der Entwicklerteams gegen eine Migration in Microservices und für einen Verbleib bei der monolithischen Architektur sprechen [Fow15].

2.4. Framework für Microservices-Migrationen

Das Framework für Microservices-Migrationen [FBH+22] und die Implementierung im Architektur Refactoring Helper [Hal22a, Hal22b, Ins23] wurde am Institute of Software Engineering in der Abteilung Empirical Software Engineering entwickelt und wird ständig weiterentwickelt [Koc22]. Fritzsch et al. [FBH+22] haben das FMM und das ARH-Tool entwickelt, um Softwarearchitekten bei den drei Schritten einer Migration von einem Monolithen in Microservices, zu unterstützen. Das FMM, so wie das ARH werden u. a. durch studentische Arbeiten, wie z. B. [Hal22a, Koc22] weiterentwickelt. Damit das FMM den Softwarearchitekten durch den Migrationsprozess führen kann, wurde es von Fritzsch et al. in drei Phasen aufgeteilt, zu sehen in Abbildung 2.4.

Fritzsch et al. sehen vor, dass in Phase 1 ein Verständnis für das existierende System aufgebaut wird, mit dem eine Entscheidung für oder gegen eine Migration getroffen wird. Mithilfe von Architektur-Review-Methoden, wie Architecture Tradeoff Analysis Method (ATAM) [KKC00], Software Architecture Analysis Method (SAAM) [KBAW94] oder der Methode von Auer et al. [ALFT21], etc., wird die Applikation erfasst und es werden Qualitätsaspekte des Systems erarbeitet. Mit diesen Qualitätsaspekten kann sich anschließend für oder gegen eine Migration in Microservices entschieden werden [FBH+22].
Das Ziel der zweiten Phase des FMM ist es, eine adäquate Migrationsstrategie zu definieren und wurde von Fritzsch et al. in zwei Aufgaben unterteilt. Die erste Aufgabe wurde von ihnen so gestaltet, dass ein Entwicklungsprozess definiert wird, der entweder eine Greenfield- oder Brownfield-Entwicklung beschreibt.

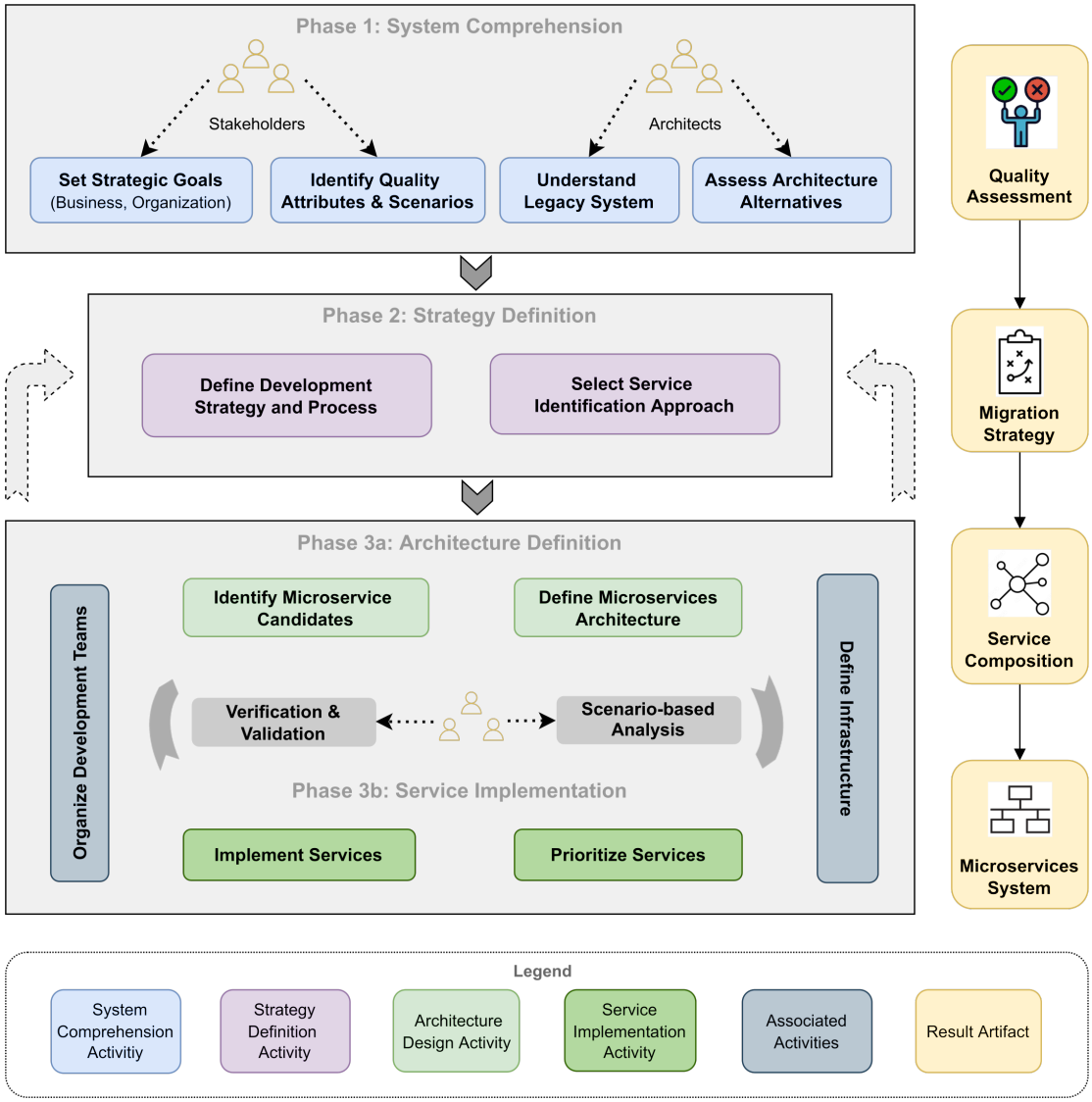


Abbildung 2.4.: FMM des ISTE ESE [FBH+22]

Sie unterteilen die Brownfield Prozesse noch in Refactor und Rebuild Typen. Sie beschreiben, dass abhängig von den Systemvoraussetzungen mehrere verschiedene Migrationsstrategien ausgewählt werden können. In der zweiten Aufgabe der Phase 2 sehen Fritzsch et al. die Auswahl eines Service-Identifikationsansatzes vor. Die resultierenden Services und die dadurch vorgegebene Granularität der Services werden durch den ausgewählten Service-Identifikationsansatz geliefert, beschreiben Fritzsch et al. Sie berücksichtigen dabei Faktoren wie Qualitätsattribute (QAs), verfügbare Eingabe Artefakte, das Potenzial der Automatisierung und die Verfügbarkeit von Tool-Support, welche die Auswahl eines geeigneten Service-Identifikationsansatzes und der Service-Identifikation beeinflussen können [FBH+22].
Die Identifikation der Services und eine vorläufige Definition der Systemarchitektur wurde von Fritzsch et al. als Bestandteil der Phase 3 des FMM gestaltet. Um zu entscheiden, in welcher

Reihenfolge die Services implementiert werden sollen, wird von ihnen eine Priorisierung vorgesehen. Sie beschreiben, dass die Services anschließend inkrementell nach ihrer Priorität implementiert werden sollen. Damit sichergestellt werden kann, dass die definierten Qualitätsziele erreicht werden, haben Fritzsch et al. im FMM einen Fokus auf Qualitätszusicherungsaspekte gelegt. Sie beschreiben, dass die Implementierung durch eine Szenario-basierte Analyse begleitet wird. Im Anschluss haben sie Verifikations- und Validations-Schritte eingefügt. Werden dabei Abweichungen von der Zieldefinition festgestellt, so sehen Fritzsch et al. eine Rückführung in Phase 2 vor. Sie beschreiben, dass dabei entweder die Zielarchitektur angepasst oder sogar ein alternativer Service-Identifikationsansatz ausgewählt werden soll [FBH+22].

2.5. Weitere Frameworks und Tools für die Migration

Es existieren neben dem FMM noch zahlreiche andere Frameworks und Tools, die bei der Migration eines Monolithen in MSs helfen sollen. Das FMM ist in seiner Herangehensweise jedoch einzigartig. In diesem Kapitel werden die Frameworks und Tools Log2MS, das Frameworks von Taibi und Systä, Arcan, MonoToMicro und Architect aus der wissenschaftlichen Literatur vorgestellt. Zum Abschluss werden die zwei Industrietools Mono2Micro von IBM und Amazon Web Service (AWS) Microservice Extractor für .Net von Amazon vorgestellt. Die vorgestellten Frameworks sollen aufzeigen, dass es viele verschiedene Vorgehen bei der Migration eines Monolithen in Microservices gibt. Dabei zeigen die zwei Frameworks auf, wie mit Log-Dateien, also dynamischen Daten, eine Aufteilung in Microservices erarbeitet werden kann. Die Tools zeigen verschiedene Methoden, welche statische Daten z. B. Klassen oder eine Architekturbeschreibung zur Identifizierung von Microservices nutzen.

2.5.1. Frameworks

Liu et al. [LXR+22] präsentieren das Log2MS Framework, welches dabei helfen soll, Legacy-Monolith-Architekturen automatisch in MSA zu transformieren. Sie haben Log2MS so gestaltet, dass es für diese Transformation nur Ausführungs-Logs benötigt. Das Log2MS Framework ist nach Liu et al. ein Model-Driven Development (MDD) Framework. Die Autoren haben Log2MS aus drei Komponenten gestaltet und sie haben das UML Standard Meta-Modell für die Arbeit des Log2MS um zwei Diagramme erweitert. Nach Liu et al. verfolgt Log2MS einen MDD-Brownfield-Design Prozess, welcher aus vier Schritten besteht,

1. Extrahieren der Funktionseinheiten.
2. Auswertung und Darstellung von Beziehungen.
3. Clustern zur Service-Identifikation.
4. Generieren von Entwurfsmodellen.

Sie geben an, dass das Extrahieren der Funktions-Einheiten die Ausführungs-Logs für das MDD-Brownfield-Design verwendet. Um die Service-Identifikation in Schritt drei zu bewerkstelligen, haben Liu et al. hierfür einen eigenen Clustering-Algorithmus implementiert, der auf dem Algorithmus Non-dominated Sorting Genetic Algorithm II [JLC+19] basiert [LXR+22]. Sie beschreiben, dass Log2MS im vierten Schritt mit den Ergebnissen aus den vorherigen Schritten die Microservices-Architektur-Modelle mit dem MSA-Generator generiert [LXR+22].

Taibi und Systä [TS19] stellen ein Framework, welches laut ihnen einen datengetriebenen Ansatz nutzt, für die Zerlegung eines Monolithen in Microservices vor. Dieses Framework soll nach Taibi und Systä Softwarearchitekten in 6 Schritten eine Menge an Zerlegungs-Optionen und eine Menge an Messungen bereitstellen, welche die Zerlegungs-Optionen evaluieren und ihre Qualitäten vergleicht. Sie haben das Framework so gestaltet, dass durch die Anwendung eines Prozess-Mining-Tools zur Laufzeit gesammelten Logs die Zerlegungs-Optionen identifiziert werden, beschreiben Taibi und Systä.
Die 6 Schritte des Frameworks von Taibi und Systä, in denen die Zerlegung stattfindet, sind:

1. Ausführungspfad Analyse.
2. Häufigkeitsanalyse der Ausführungspfade.
3. Entfernung zirkulärer Abhängigkeiten.
4. Identifizierung von Zerlegungs-Optionen.
5. Metriken basiertes Einstufen der Zerlegungs-Optionen.
6. Auswählen der Zerlegungslösung.

Die Metriken, welche sie in Schritt 5 verwenden, sind Kopplung, Anzahl an Klassen pro Microservice (MS) und Anzahl an Klassen, die dupliziert werden müssen. Mithilfe dieses Tools sollen den Autoren zu folge Softwarearchitekten dabei unterstützt werden, die richtige Zerlegungs-Option zu wählen [TS19].

2.5.2. Tools

Pigazzini et al. [PAM19] stellen ihr Tool Arcan, welches in monolithischen Java Projekten MS-Kandidaten identifiziert, vor. Ihr Ansatz besteht aus einer statischen Analyse der Systemarchitektur, Architekturmängel (Architektur-Smell) Detektion und Themen Detektion. Um Software-Domänen zu modellieren, verwendet Arcan laut Pigazzini et al., eine Text-Mining Methode. Arcan beruht auf Graph-Datenbank-Technologie und es hat seinen Ursprung, laut Pigazzini et al. so wie Fontana et al., als Tool zum Entdecken von Architekturmängeln [FPR+17, PAM19]. Pigazzini et al. beschreiben, dass in dem Graph Java Entitäten wie z. B. Klassen, Packages und Methoden die Knoten bilden und Kanten die Beziehungen unter den Entitäten repräsentieren. Sie beschreiben, dass ein weiterer Bestandteil des Graphen Super-Knoten sind, die für Architekturmängel stehen. Arcan nutzt laut den Autoren Techniken wie Graph-Algorithmen und Themen-Detektion, um Microservices-Kandidaten zu identifizieren.
Pigazzini et al. sehen einen Migrationsansatz in drei Schritten vor:

1. Architekturmängel Detektion.

2. Abhängigkeits-Graph Analyse.
3. Themen Detektion.

Schritt 1 gibt laut den Autoren durch Architekturmängel Hinweise darauf, wie das Projekt zerlegt werden kann.
Die Autoren beschreiben, dass in Schritt 2 der Fokus darauf gesetzt wird, strukturell unabhängige Blöcke aus dem Projekt zu finden, die in Microservices transformiert oder als Microservices wiederverwendet werden können. Schritt 2 wurde von Pigazzini et al. dafür in drei weitere Schritte unterteilt:

- Detektion verbundener Elemente des Abhängigkeits-Graphen.
- Generierung einer Vertikale-Funktionalitäten-Sicht.
- Generierung einer Logische-Schicht-Sicht.

Laut den Autoren wird im letzten Schritt, von Arcan, eine „semantische Karte“ des Projekts generiert. Damit die „semantische Karte“ generiert werden kann, müssen Teile des Projekts identifiziert werden, welche zur gleichen Domäne gehören, beschreiben die Autoren. Dafür haben Pigazzini et al. Schritt 3 in zwei weitere Schritte unterteilt, (1) Latente Dirichlet-Zuordnung und (2) Seeded Latent Dirichlet-Zuordnung. Dadurch können laut den Autoren die geeignetsten Zerlegungs-Lösungen für das Projekt gewählt werden [PAM19].

Um einen objektorientierten Monolithen in Microservices zu transformieren, schlagen Zaragoza et al. [ZSS+21] einen systematischen Ansatz vor. Sie beschreiben, dass dieser Ansatz auf Transformationsmustern basiert. Ihr Migrationsprozess besteht aus den zwei Schritten (1) MSA Gewinnung und (2) der Transformation. Um diesen Prozess für die Migration zu automatisieren, haben sie das Tool MonoToMicro implementiert.
Die Basis für Struktur- und Verhaltens-gültige Microservices wird bei Zaragoza et al. durch das Zerlegen von Klassen des Monolithen in Clustern erzielt. Sie beschreiben, dass Cluster aus Internen- und Rand-Klassen bestehen. Wenn eine Klasse keine Abhängigkeiten von Klassen eines anderen Clusters besitzt, so wird sie von Zaragoza et al. als interne-Klasse bezeichnet. Eine Klasse mit mindestens einer Abhängigkeit von einer Klasse aus einem anderen Cluster, wird von ihnen als Rand-Klasse bezeichnet.
Sie beschreiben, dass jedes Cluster in seinen eigenen Microservice transformiert wird. Zuerst müssen jedoch, so schreiben Zaragoza et al., MS-Verkapselungsverstöße, welche durch Rand-Klassen entstehen, beseitigt werden. Sie beschreiben, dass der Transformations-Prozess von MonoToMicro aus vier Schritten besteht [ZSS+21]:

1. Detektieren von Verkapselungsverstößen.
2. Heilen von Verkapselungsverstößen.
3. Zusammenpacken von Microservices.
4. Verteilen und Containerisieren von Microservices.

Volynsky et al. [VMK22] haben das Framework Architect entwickelt. Dabei benutzt Architect, wie die Autoren beschreiben, das *Saga* Pattern, um den Monolith in einem Greenfield Projekt, in Microservices zu zerlegen.

Sie beschreiben, dass Architect aus den Domänen Architekturmuster, Applikation, Entitäten Modell, Lösung, Verteilung und Konfiguration besteht, die bei der Beschreibung einer zuverlässigen Systemarchitektur helfen sollen. Um mit Architect zu arbeiten, muss nach Volynsky et al., zuerst eine Architekturbeschreibung erstellt und in die „Architect Engine“ importiert werden. Um die importierte Architekturbeschreibungssprache zu erkennen, benutzt der „Architect Compiler“ eine Grammatik und einen Parser. Außerdem wird von ihnen beschrieben, dass der „Architect Compiler“ die Meta-Informationen der Architekturbeschreibung in ein Daten-Transfer-Obkjekt (DTO) transformiert. Anschließend wird, nach Volynsky et al., der „Architect Generator“ benachrichtigt, damit er beginnen kann, die Projektstruktur zu generieren.
Es wird von ihnen beschrieben, dass der „Architect Generator“ im Anschluss den „Architect Supplier“ benachrichtigt. Die generierten Artefakte werden von dem „Architect Supplier“ in ein Repository oder einen lokalen Speicherort geladen, beschreiben Volynsky et al. Die Autoren beschreiben, dass der Nutzer benachrichtigt wird, sobald der Prozess abgeschlossen wurde [VMK22].

Durch die in diesem Kapitel vorgestellten Frameworks und Tools wurde Forschungsfrage 1 beantwortet.

2.5.3. Industrie Tools

Um neben den Frameworks für die Migration in Microservices aus der wissenschaftlichen Literatur auch Frameworks aus der Industrie zu beleuchten, werden im Anschluss noch Mono2Micro von IBM [IBM20, KXK+21] und der AWS Microservice Extractor für .Net [AWS23a] vorgestellt.

Nach Gulati et al. ist Mono2Micro ein Tool, welches den Refactoring-Prozess in MSs mit künstlicher Intelligenz unterstützt [IBM20]. Wie Kalia et al. und Gulati et al. [IBM20, KXK+21] beschreiben, analysiert Mono2Micro Laufzeit-Aufrufspuren, welche durch die Abarbeitung von spezifischen Geschäftsanwendungsfälle generiert werden. Sie beschreiben, dass durch diese Analyseklassen nach ihrer Geschäftslogik gruppiert werden. Dieser Ansatz wird von Kalia et al. [KXK+21] hierarchische räumlich-zeitliche Zerlegung genannt. Durch Mono2Micro versprechen sie sich die Identifizierung optimaler Microservices-Kandidaten [IBM20, KXK+21].

Nach der Beschreibung von Amazon Web Services, Inc. [AWS23a, AWS23b] visualisiert AWS Microservice Extractor für .Net die Komponenten des Quellcodes und ihre Kommunikation. Sie beschreiben, dass mithilfe dieser Visualisierung Softwarearchitekten die Klassen nach verschiedenen Bedingungen gruppieren können. Es wird von ihnen beschrieben, dass Funktionalitäten durch den Microservice Extractor in separate Codelösungen extrahiert werden, wenn die Codebasis bereit dafür ist [AWS23a, AWS23b].

Ein Framework wie das FMM, welches aus mehreren Service-Identifikationsansätze und Migrationsansätzen den am besten passenden auswählt und durch die Migration führt, wurde nicht gefunden.

3. Methodik

In diesem Kapitel wird die Methodik der Arbeit beschrieben. Die Methodik dieser Arbeit ist in drei Abschnitte unterteilt, die Literaturrecherche, die Migration der Service/Sales-Applikation und die Generierung und Auswertung von strukturierten Feldnotizen.
Kapitel 3.1 beschreibt die Literaturrecherche. Das Kapitel befasst sich mit dem Vorgehen der Sammlung und Auswertung von wissenschaftlicher Literatur, um Grundlagen für diese Arbeit zu schaffen und um Forschungsfrage 1 zu beantworten.
In Kapitel 3.2 wird das Vorgehen während der Migration des L-mobile Monolithen in Microservices beschrieben. In diesem Kapitel wird auf das FMM eingegangen und wie mit diesem Framework eine (Teil-) Migration der Service/Sales-Applikation angestrebt wird. Mit dieser Anwendung des FMM soll Forschungsfrage 3 beantwortet werden.
Kapitel 3.3 beschreibt wie bei der Generierung und Auswertung von Feldnotizen vorgegangen wird. Dabei wird definiert, was in der wissenschaftlichen Literatur unter strukturierten Feldnotizen verstanden wird und wo diese strukturierten Feldnotizen in der Wissenschaft zum Einsatz kommen. Dieses Kapitel enthält ebenfalls eine Definition der Struktur der in dieser Arbeit generierten Feldnotizen und es wird beschrieben, wie diese strukturierten Feldnotizen ausgewertet werden. Mithilfe dieser Feldnotizen werden die Forschungsfragen 2 und 4 beantwortet.

3.1. Literaturrecherche

Für die Grundlagen von Monolithen, Microservices und Migrationen wurde eine Ad-hoc-Literaturrecherche durchgeführt. Eine Systematische Literatur Recherche (SLR) [KBB+09, KDJ04] oder ein Rapid Review (RR) [CPF+19, CPS20] kam für diesen Schritt nicht in Frage, da eine SLR einen viel zu großen Aufwand für diese Arbeit hat, zum Teil mehrere Monate oder sogar Jahre wie Khangura et al. [KKC+12], Tricco et al. [TAZ+15] und Cartaxo et al. [CPS20] anmerken. Ein RR kam für diese Recherche auch nicht in Frage, obwohl sich diese Methode, nach Cartaxo et al., zügig und mit niedrigen Kosten durchführen lässt [CPS18]. Die Literaturrecherche steht in dieser Arbeit nicht im Fokus und es werden lediglich Grundlagen für diese Arbeit recherchiert. Dennoch wurden Teile einer SLR und eines RR in der Literaturrecherche angewendet. Eine Suchstrategie, wie von Cartaxo et al. [CPS20] für SLRs und RRs beschrieben, wurde modifiziert angewendet. Es wurde nur eine Suchmaschine, Google

Scholar[1], verwendet. Eine manuelle Suche wurde nicht angewendet aber Snowballing war Teil der Suchstrategie in dieser Arbeit. Eine Einschränkung der Veröffentlichungen durch ihr Datum, ihre geographische Lage oder ihre Forschungsmethoden, wie sie nach Cartaxo et al. als Teil der Suchstrategie eines RRs vorkommen, wurde nicht vorgenommen. Jedoch wurden die Veröffentlichungen durch ihre Sprache eingeschränkt, es wurden nur Veröffentlichungen berücksichtigt welche in Deutsch oder Englisch verfasst sind. Ein Selektionskriterium, welches nach Kitchenham [Kit04] zu einem SLR und nach Cartaxo et al. [CPS20] zu einem RR gehört, wurde eingesetzt. Eine Datenextraktion wurde so wie sie von Kitchenham [Kit04] beschrieben wurde nicht durchgeführt. Ein Review Protokoll, was nach Kitchenham ebenfalls zu einem SLR gehört, wurde nicht angefertigt. Die Planung, die Identifizierung der Forschung, eine Qualitätsprüfung, eine Datensynthese und das Reporten des SLR, welche nach Kitchenham zu einem SLR gehören wurden ebenfalls nicht durchgeführt. Wie Cartaxo et al. [CPS20] beschreiben, kann eine Qualitätsprüfung in einem RR u. U. ausgelassen werden. Eine Planung wie in einem RR wie von Cartaxo et al. beschrieben wurde ebenfalls nicht durchgeführt. Die Literaturrecherche wurde in sechs Schritten durchgeführt. In Schritt eins und zwei wurde Literatur für die Definition von Software Monolithen und Microservices recherchiert. Im dritten Schritt wurde Literatur zu Software-Migrationen und Migrationen von Monolithen in Microservices analysiert. Der vierte Schritt befasst sich mit wissenschaftlicher Literatur zu Herausforderungen von Migrationen. Eine Literaturrecherche zu strukturierten Feldnotizen und deren Generierung und Auswertung wurde im fünften Schritt durchgeführt. Im letzten Schritt wurde Literatur zu Frameworks und Tools, welche bei einer Migration von Monolithen in Microservices unterstützen, analysiert. Für die Literaturrecherche wurden die vom Betreuer Jonas Fritzsch vorgeschlagenen Veröffentlichungen berücksichtigt. Um die Grundlagen zu erläutern, wurden viele der Referenzen aus der Masterarbeit von Koch [Koc22] verwendet.
Als Einstiegspunkt für die Literaturrecherche zu verwandten Arbeiten wurde die Suchmaschine Google Scholar verwendet. Vorzugsweise werden Publikationen berücksichtigt, die zur Beantwortung der aufgestellten Forschungsfragen dienen oder Grundlagen für diese Arbeit erläutern. Es wurden nur Publikationen berücksichtigt, welche in deutsch oder in englisch verfasst sind. Mit den bereits zur Verfügung gestellten Publikationen wurde eine rückwärts bzw. vorwärts Schneeballsuche durchgeführt.
Als Einstiegspunkt für die Literaturrecherche von Grundlagen zu Monolithen, Microservices und Migrationen wurden die vorgeschlagene Literatur von dem Betreuer Jonas Fritzsch, so wie die Referenzen aus der Masterarbeit von Koch [Koc22] herangezogen und durch eine Suche nach Monolithen, Microservices und Softwaremigrationen auf Google Scholar ergänzt. Für die ergänzende Literatur wurde überprüft, ob sie neue Erkenntnisse im Bezug auf die Definition von Monolithen, Microservices und Softwaremigrationen haben und ob diese Arbeit die original Quelle der Erkenntnisse sind. Falls die Erkenntnisse aus einer referenzierten Arbeit stammen, so wurde die referenzierte Arbeit konsolidiert. Zum Schluss wurde eine rückwärts Schneeballsuche durchgeführt, vor allem bei der Masterarbeit von Koch [Koc22], welche u. a. in den Artikeln von Fowler und Lewis [FL14, Fow15] resultierte.

[1] https://scholar.google.de/

Für die Grundlagen des FMM wurde lediglich die zur Verfügung gestellte Arbeit von Fritzsch et al. [FBH+22] verwendet.
Für die Herausforderungen, welche bei einer Migration eines Monolithen in Microservices auftreten können, wurde eine Suche im Artikel mit dem Suchstring:
„microservices AND migration AND challenges“
durchgeführt. Außerdem wurden Veröffentlichungen, die bei der Erläuterung der Grundlagen einer Migration verwendet wurden, analysiert, ob diese auch Herausforderungen von Migrationen beschreiben.
Für die Grundlagen der Generierung und der Auswertung von strukturierten Feldnotizen wurde die von Jonas Fritzsch vorgeschlagene Literatur Seaman [Sea08] verwendet. Auf Google Scholar wurde lediglich weitere Anwendungsgebiete der strukturierten Feldnotizen gesucht. Dabei wurden die Veröffentlichungen [Don09, MB07, PL18] gefunden und verwendet. Für die Literaturrecherche zu Frameworks und Tools für die Migration von Monolithen in Microservices wurde ebenfalls eine Ad-hoc-Literaturrecherche durchgeführt. In der Suchstrategie wurde Google Scholar verwendet. Als Suchstring wurden die Strings
„framework AND microservices AND migration“und „tool AND microservices AND migration“
verwendet. Die Suche bezieht sich auf den gesamten Artikel. Der Selektionsprozess inkludierte alle Veröffentlichungen, welche ein Framework oder Tool für die Migration von Software Monolithen in Microservices vorstellen. Exkludiert wurden Veröffentlichungen, welche nicht mit einem Virtual Private Network (VPN) ins Universitätsnetzwerk aus verfügbar waren und gekauft werden mussten. Das Datum und der Publikationstyp der Veröffentlichung war für die Auswahl irrelevant. Die Sprache der Veröffentlichung musste Deutsch oder Englisch sein. Eine Qualitätsbewertung der Veröffentlichungen wurde nicht durchgeführt, da eine Qualitätsbewertung für die Sammlung von Frameworks und Tools für die Migration von Monolithen in Microservices als nicht nötig erachtet wurde. Für die beschriebenen Suchstrings gab es auf Google Scholar am 16ten Februar 2023 insgesamt 22.800 Ergebnisse (11.700 für „framework AND microservices AND migration“und 11.100 für „tool AND microservices AND migration“). Diese Ergebnisse wurden nach ihren Titeln gefiltert. Titel, die nur auf Migrationen oder Microservices verwiesen, wurden exkludiert und nur Titel, die auf ein Framework oder Tool hinwiesen, welche zur Migration in Microservices verwendet werden, wurden ausgewählt. Außerdem wurde darauf geachtet, dass die Publikationen kostenfrei, bzw. mit Open Access über das Universitätsnetzwerk, zur Verfügung standen. Diese Bedingungen wurden angewandt bis 25 Publikationen gesammelt waren.
Nach diesem Schritt wurde der Abstrakt, die Einführung und der Abschluss gelesen. Wenn kein neues Framework in dieser Veröffentlichung vorgestellt wurde, so wurde die Veröffentlichung zurückgewiesen. Nach diesem Schritt standen noch fünf Veröffentlichungen zur Verfügung.
Mit der Literaturrecherche zu Frameworks und Tools für die Migration von Monolithen in Microservices wurde Forschungsfrage 1, Welche Frameworks werden in der wissenschaftlichen Literatur für die Umstrukturierung eines Monolithen in Microservices vorgeschlagen und welche Herangehensweisen bzw. Prozesse verfolgen diese, beantwortet.

3.2. Migrationsstrategie

Die Umstrukturierung des Monolithen von L-mobile in Microservices wird als Fallstudie zur Unterstützung der Evaluation des durch das ISTE ESE vorgeschlagenen Frameworks [FBH+22, Ins23] durchgeführt. Der Prozess richtet sich dabei an den drei Phasen, siehe Abbildung 2.4, des Frameworks aus [FBH+22, Ins23]. Während der Anwendung des FMM stellt sich heraus, welche Migrationsstrategie oder welche Migrationsstrategien und welche Service-Identifikationsansätze für die zu migrierende Applikation verwendet werden sollen. Mit diesen vorgeschlagenen Strategien und Ansätzen wird dann die Migration der Service/Sales-Applikation von L-mobile durchgeführt.
Nach Fritzsch et al. werden mit der ersten Phase des FMM Qualitätsattribute und Metriken, welche für oder gegen die Umstrukturierung sprechen, gesammelt [FBH+22]. Dafür wird die Architekturevaluationsmethode von Svahnberg und Mårtensson [SM07] angewendet.
Die zweite Phase befasst sich mit der Erstellung einer Migrationsstrategie und besteht aus zwei Teilen, wie Fritzsch et al. beschreiben [FBH+22]. Im ersten Teil wird laut den Autoren entschieden, ob der Entwicklungsprozess in einem Rebuild oder Re-factor Prozess durchgeführt wird. Nach Fritzsch et al. identifiziert die zweite Aktivität der Phase 2 basierend auf den Ergebnissen der Phase 1, passende Verfahren zur Aufteilung des Monolithen in Microservices.
In der dritten Phase des FMM werden die Microservices identifiziert und die Architektur der resultierenden Microservices erstellt, beschreiben die Autoren. Damit kann dann z. B. durch eine Priorisierung der Microservices wie bei den Autoren beschrieben, entschieden werden, welcher Teil der Service/Sales-Applikation sich für einen Microservices Prototypen am besten eignet. Danach wird, wie bei Fritzsch et al. beschrieben, die Implementierung der Umstrukturierung durchgeführt [FBH+22]. Das Framework unterstützt hier durch Empfehlung von Best Practices und Patterns den Entwurf einer Microservices-Architektur [Ins23].
Da am Anfang keine Erfahrung mit der Implementierung von Microservices vorhanden war, wurde nach Informationen und Beispielen gesucht, welche bei der Implementierung von Microservices in .Net helfen konnten. Dabei wurde auf YouTube[2] das Video[3] von freeCodeCamp.org[4] gefunden. Dieses Video behandelt die Implementierung zweier Microservices in .Net, sowie die Implementierung ihrer Kommunikation. Im Anschluss war genug Wissen vorhanden, um mit der Implementierung des Microservice und seiner Kommunikation zum Monolithen zu starten. Für die Implementierung wurde die Entwicklungsumgebung Visual Studio [5] verwendet.
Mit dem Migrationskonzept und der (Teil-) Implementierung wird das FMM kritisch auf seine praktische Anwendbarkeit hin untersucht. Im Falle, dass das FMM in einem Punkt der Umstrukturierung an seine Grenzen stößt, wird eine Recherche zu weiteren Methoden oder Frameworks durchgeführt, damit der Informationsgrad in diesem möglichen spezifischen Fall erhöht werden kann. Das kann in unterschiedlichen Phasen des Frameworks durchgeführt werden und

[2] https://www.youtube.com
[3] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg
[4] https://www.freecodecamp.org/, https://www.youtube.com/@freecodecamp
[5] https://visualstudio.microsoft.com/de/

kann zu einer Erweiterung des Frameworks beitragen. Weiterhin werden Code-Reviews durch Experten durchgeführt, damit sichergestellt werden kann, dass der Codestandard erhalten bleibt. Um sicher zu stellen, dass die Verarbeitung der Daten nach der (Teil-) Umstrukturierung gegenüber dem Monolithen erhalten bleibt, werden automatisierte Tests für den prototypisch implementierten Teil ausgeführt. Um eine Bewertung über den Migrationsprozess durchzuführen, wird die qualitative Forschungsmethode, der strukturierte Feldnotizen verwendet [Sea08]. Während der Migration und Anwendung des FMM werden systematisch Erfahrungen und Herausforderungen dokumentiert und am Ende ausgewertet.

3.3. Strukturierte Feldnotizen

In verschiedenen Veröffentlichungen werden Feldnotizen als detaillierte Notizen, die sowohl im empirischen Software Engineering, als auch in der Pflege, Gesundheit, Psychologie und Medizin, z. B. für die begründete Theorie zum Einsatz kommen [Don09, MB07, PL18, Sea08]. Seaman [Sea08] beschreibt Feldnotizen als sehr detaillierte Notizen, welche bei qualitativen Datensammlungsmethoden, z. B. Teilnehmerbeobachtung und Interviews angewendet werden. In Feldnotizen können als Beispiel der Autorin Aktionen des Teilnehmers, wie z. B. Tastatur- und Maus-Interaktionen, Ziele und Motivation des Teilnehmers, die durch Aussagen des Teilnehmers begründet werden, notiert werden. Sie beschreibt aber auch, dass Dinge, die den Teilnehmer ablenken oder nervös machen, notiert werden können. Nach Seaman sollten Feldnotizen so zeitnah wie möglich nach ihrer Erstellung mit so vielen Details wie möglich, an die sich der Schreiber erinnert, ergänzt werden.
Nach Seaman sollten Feldnotizen auf jeden Fall den Ort, die Zeit, die Teilnehmer, die Diskussion, jedes Ereignis, welches während der Beobachtung/des Interviews stattgefunden hat, so wie den Tonfall und die Stimmung des Teilnehmers enthalten. Sollte der Beobachter das Bedürfnis haben, Eindrücke zu dokumentieren, welche durch etwas entstanden sind, das nicht direkt gesagt wurde oder aufgetreten ist, so wird von Seaman vorgeschlagen, extra markierte Kommentare in die Feldnotiz mit aufzunehmen.
Sie beschreibt, dass die Ziele des Forschers den Grad der Feinheit der Feldnotizen bestimmen. Sie gibt jedoch zu bedenken, dass es zeitaufwendiger wird, Feldnotizen zu erstellen, um so detaillierter diese Notizen gestaltet werden.
Die Autorin beschreibt, dass es wichtig ist, dass die Daten, die aufgenommen wurden, akkurat sind und in einer Form aufgenommen sind, welche nicht nur für den Beobachter verständlich ist, damit die Feldnotizen ausgewertet werden können. Außerdem sollte, nach Seaman, vor der Datensammlung die Datenanalyse geplant werden. Um Feldnotizen zu analysieren und auszuwerten, stellt Seaman die Konstante Vergleichsmethode und die Fall-übergreifende Analyse für die Generierung einer Theorie vor. Für die Bestätigung der Theorie beschreibt Seaman Methoden wie Negative Fallanalyse, wie beschrieben von Judd et al. [JSK91], Triangulation, wie beschrieben von Jick [Jic79] und Anomalien, wie die Ausreißeranalyse beschrieben von Miles und Huberman [MH94]. Wenn Anomalien durch keine anderen Variablen erklärt werden

können, können sie in der qualitativen Analyse der Erklärung, Gestaltung und sogar zur Unterstützung einer Aussage dienen, wie Seaman anhand der „Inspection Study“ erklärt [Sea08]. Weiterhin beschreibt Seaman die Replikation, wie beschrieben von Brooks et al. [BRW+08] und die Mitgliederprüfung, wie beschrieben von Lincoln und Guba [LG85] als Methoden für die Bestätigung der Theorie [Sea08]. Methoden der Bestätigung der Theorie werden auch auf die in dieser Arbeit erstellten strukturierten Feldnotizen angewendet.
Für die Feldnotizen die in dieser Arbeit zum Einsatz kommen wurde folgende Vorlage, welche sich an den Vorschlägen von Seaman [Sea08] orientiert, gewählt:

- Datum
- Uhrzeit
- Ort
- Phase des FMM
- Schritt
- Durchgeführte Aktivitäten
- Betroffener Teil der Software
- Was lief gut
- Was lief schlecht
- Empfindungen
- Kommentare
- Sonstiges

In Tabelle 3.1 werden die Bedeutungen der Einträge der Vorlage erläutert. Die Feldnotizen werden digital erfasst. Jede Feldnotiz wird in einer eigenen Textdatei gespeichert. Damit sichergestellt ist, dass jede Feldnotiz einen eindeutigen Namen bekommt, der schnell auf den Inhalt der Feldnotiz schließen lässt, wurde folgendes Namens-Schema ausgewählt: *FX_PY_Datum_Kürzel*. Dabei steht *X* für die Nummer der Feldnotiz, *Y* bezeichnet die Nummer der Phase, *Datum* ist das Datum der Feldnotiz in dem Format $JJMMTT$ und *Kürzel* ist eine Abkürzung für die Aktivitäten, die in der Feldnotiz beschrieben ist.
Da in dieser Arbeit mit den strukturierten Feldnotizen die Forschungsfrage 2: Inwieweit eignet sich das FMM des ISTE ESE, um den L-mobile Monolithen in Microservices umzustrukturieren, beantwortet und die Theorie, dass sich das FMM für diese Aufgabe eignet, unterstützt oder widerlegt werden soll, kommt für die Auswertung der strukturierten Feldnotizen in dieser Arbeit die Methoden der Bestätigung der Theorie zum Einsatz.
Mit den Feldnotizen, die während der Anwendung des FMM und ARH generiert und nach der Implementierung ausgewertet werden, werden die Forschungsfragen 2 und 4 beantwortet.

Eintrag	**Bedeutung**
Datum	Das Datum, an dem die Feldnotiz erfasst wurde, sollte nach Seaman enthalten sein.
Uhrzeit	Die Uhrzeit, zu der die Feldnotiz erfasst wurde, sollte nach Seaman enthalten sein.
Ort	Der Ort, an dem die Feldnotizen und die Analyse durchgeführt wurden, sollte nach Seaman enthalten sein.
Phase des FMM	In welcher Phase des FMMs befindet sich die Migration zu dem Zeitpunkt, in der die Feldnotiz Notiz erstellt wurde. Bei Seaman entspricht dies der Motivation und die verfolgten Ziele.
Schritt	Welcher Schritt der aktuellen Phase des FMMs wurde ausgeführt. Bei Seaman entspricht dies der Motivation und die verfolgten Ziele.
Durchgeführte Aktivitäten	Auf welche Handlung bezieht sich die Feldnotiz. Bei Seaman entspricht dies der Handlung des Subjekts.
Betroffener Teil der Software	Modul der Software und die Funktion an der in diesem Schritt gearbeitet wurde. Bei Seaman entspricht dies der Motivation und die verfolgten Ziele.
Was lief gut	Aktivitäten, die in diesem Schritt leicht umzusetzen waren. Bei Seaman entspricht dies der Diskussion.
Was lief schlecht	Aktivitäten, bei denen es in diesem Schritt Probleme gab. Bei Seaman entspricht dies der Diskussion.
Empfindungen	Wie waren die Empfindungen der Person welche die Umsetzung betrieben hat. Wahrzunehmen durch z. B. Stimmung und Tonlage.
Kommentare	Weitere Anmerkungen, die sich nicht unbedingt auf den Prozess der Umstrukturierung oder der Anwendung des FMM beziehen, aber die Ergebnisse des aktuellen Schritts beeinflussen können.
Sonstiges	Anmerkungen zum Umstrukturierungsprozess. Falls Methoden außerhalb des FMM angewendet wurden, so werden diese hier aufgeführt. Bei Seaman entspricht dies der Diskussion.

Tabelle 3.1.: Übersicht zu den verwendeten Einträgen der Feldnotizen nach Seaman [Sea08]

4. Durchführung einer Migration in Microservices unter Verwendung des FMM

In diesem Kapitel wird beschrieben, wie die Migration unter Verwendung des FMM und des ARH durchgeführt wurde. Dabei ist dieses Kapitel in Unterkapitel aufgeteilt, welche die Phasen des FMM repräsentieren. In Kapitel 4.1 wird das Projekt an sich, die strategischen Ziele, die Planung und Durchführung der Architekturbewertung und die daraus resultierenden Qualitätsattribute beschrieben.
Phase 2 des FMM wird in Kapitel 4.2 beschrieben. Dabei ist diese Kapitel in die zwei Aufgaben, welche in dieser Phase separat durchgeführt werden, unterteilt. Basierend auf den Ergebnissen der ersten Phase wird hier beschrieben, wie ein Service-Identifikationsansatz und eine Migrationsstrategie ausgewählt wurde.
Die Durchführung der Phase 3 wird in Kapitel 4.3 beschrieben. Hier befinden sich das konkrete Vorgehen während der Service-Identifizierung, die Definition der Zielarchitektur unter Berücksichtigung der Best Practices und Patterns und die Priorisierung der Services.

4.1. Phase 1

Im Zentrum der Phase 1 des FMM steht das Systemverständnis [FBH+22]. Um herauszufinden, ob eine Migration in Microservices oder in einen anderen Architekturstil infrage kommt, wird in dieser Phase eine Architekturbewertung durchgeführt [FBH+22]. Die Gliederung dieser Kapitel orientiert sich an dem Aufbau der Phase 1, wie sie im FMM und ARH sowie von Fritzsch et al. [FBH+22] beschrieben ist. Dieses Kapitel ist, um den Architekturbewertungsprozess übersichtlich und vollständig zu beschreiben, wie folgt gegliedert. In Kapitel 4.1.1 wird die Planung der Architekturbewertung beschrieben, dabei umfasst dieses Kapitel die Planung der Organisation sowie die Projektbeschreibung und die strategischen Ziele. Kapitel 4.1.2 geht auf die Ausführung der Architekturbewertung ein. Dafür werden die identifizierten Qualitätsattribute und Szenarien beschrieben sowie die Resultate der Architekturbewertung wiedergegeben.

4.1.1. Planung der Architekturbewertung

Um eine Architekturbewertung für die Service/Sales-Applikation bei L-mobile zielführend durchzuführen, wurde viel Zeit, insgesamt 21 Tage im dem Zeitraum vom 1.3.2023 bis zum 23.3.2023, in die Planung der Architekturbewertung investiert. Zuerst wurde eine Methode für die Bewertung, basierend auf den Faktoren Durchführungszeit, Anzahl der Stakeholder und das Einbinden von Qualitätsattributen und Szenarien ausgewählt.
Die Methoden ATAM [KKC00] und SAAM [KBAW94] sind bewährte Architekturbewertungsmethoden, die laut ihren Autoren Szenarien basierend auf Qualitätsattributen, verwenden. Jedoch ist ATAM sehr aufwendig und benötigt viel Zeit, laut den Autoren zum Teil mehrere Wochen [KKC00]. Da dieser Zeitaufwand während des kurzen Zeitraums dieser Arbeit nicht zur Verfügung steht und die Stakeholder ihre Aufgaben in der Firma erledigen müssen, wurde entschieden, ATAM nicht anzuwenden. Obwohl sich SAAM theoretisch an einem Tag durchführen lässt, wie Ionita et al. beschreiben [IHO02], wurde sich gegen diese Methode entschieden, da nach Dobrica und Niemela mit der Erhebung neuer Szenarien erst aufgehört wird, wenn neue Szenarien das Design der Softwarearchitektur nicht mehr stören [DN02] und diese Methode daher zu aufwendig wird.
Die Architekturbewertungsmethode von Auer et al. wurde, wie sie beschreiben, entwickelt, um bei der Entscheidung zu helfen, ob eine bestehende Software von einer monolithischen Architektur in eine Microservices-Architektur migriert werden soll [ALFT21]. Jedoch berücksichtigt dieses Framework nur Qualitätsattribute und keine Szenarien. Aus diesem Grund wurde sich gegen die Methode von Auer et al. [ALFT21] entschieden.
Svahnberg und Mårtensson [SM07] stellen eine Architekturbewertungsmethode vor, die, wie sie beschreiben, auf ATAM und SAAM basiert, innerhalb von 4 bis 5 Stunden durchgeführt wird und 3 bis 4 Stakeholder involviert. Aufgrund dieser Eigenschaften eignet sich diese Methode für die zur Verfügung stehenden Ressourcen am besten. Daher wurde sich dafür entschieden, diese Methode für die Architekturbewertung anzuwenden. Die Methode von Svahnberg und Mårtensson besteht laut den Autoren aus sechs Schritten:

1. Einführung
2. Identifizierung der Qualitätsattribute
3. Szenarien Erhebung
4. Präsentation der Architektur
5. Szenarien und Architektur Analyse
6. Ergebnis

Wobei nach Schritt 3 eine kurze Pause stattfindet [SM07].
Als nächstes mussten die Stakeholder identifiziert und die Vorgesetzten, in diesem Fall der Head of Development und der Product Manager, überzeugt werden, eine Architekturbewertung mit den identifizierten Stakeholdern zu genehmigen. Als Stakeholder wurden der Head of Development, der Product Manager, der Softwarearchitekt und ein leitender Entwickler identifiziert. Der Head of Development wurde als Stakeholder ausgewählt, da er die meiste Erfahrung in dem Produkt hat und die übergeordnete Strategie und Ziele am besten kennt.

Graaf et al. beschreiben, dass wichtige Personen der Organisation in einer Architekturbewertung teilnehmen sollen [GDD05], der Head of Development gehört zu dieser Gruppe. Der Product Manager wurde ausgewählt, da er die Qualitätsattribute und die Funktionen des Systems am besten kennt und gemeinsam mit dem Head of Development entscheidet, in welche Richtung das Produkt entwickelt wird. Svahnberg und Mårtensson verdeutlichen, dass der Product Manager, bzw. in ihrem Fall der Project Manager, ein wichtiger Stakeholder ist, da er den Einfluss der Qualitätsanforderungen verstehen muss [SM07]. Der Softwarearchitekt und der leitende Entwickler wurden ausgewählt, da sie sich mit der Architektur des Systems am besten auskennen und schon lange an dem Produkt entwickeln. Außerdem sollte nach Svahnberg und Mårtensson der Softwarearchitekt die Architektur des Systems vorstellen [SM07]. Graaf et al. geben an, dass auch Softwarearchitekten und Entwickler zu den typischen Stakeholdern einer Architekturbewertung gehören [GDD05]. Auch Kazman et al. beschreiben Softwarearchitekten, Softwareentwickler und Manager als typische Stakeholder einer Architekturbewertung [KKC00].
Um den Head of Development und den Product Manager zu überzeugen, wurde ein separates Meeting mit jedem gehalten, in denen die Bewertungsstrategie, die Ziele und auch der Nutzen abseits dieser Arbeit, einer Architekturbewertung erläutert wurden. Beide waren schnell überzeugt und gewillt, die benötigten Ressourcen freizugeben. Der Termin für die Architekturbewertung wurde schnell gefunden, und alle Stakeholder wurden eineinhalb Wochen vor dem Termin über die Architekturbewertung informiert und eingeladen. Der Softwarearchitekt wurde darum gebeten die Architektur des Systems in dem Meeting vorzustellen. Die Rolle des Moderators und des Protokollschreibers wurde vom Autor dieser Arbeit übernommen.

Projektbeschreibung

Das System, welches in dieser Arbeit von einer monolithischen Architektur in eine Microservices-Architektur (Teil-)migriert wird, ist eine Service/Sales-Web-Applikation. Das zu Grunde liegende Architekturmuster ist eine Client-Server-Architektur, zu sehen in Abbildung 4.1.
Auf der Client-Seite können dabei Browser oder Cordova Apps verwendet werden. Damit ein Service-Techniker im Einsatz auch ohne Internet arbeiten kann, verfügt die Client-Seite über einen Offline-Modus mit einer WebSQL-Datenbank, der sich auf Knopfdruck oder sobald er wieder mit dem Internet verbunden wird, mit dem Backend synchronisiert.
Die Kommunikation zwischen Frontend und Backend findet über HTTPS statt.
Das Backend, welches auf einem Windows Server verwaltet wird, ist mit der ASP.NET Technologie implementiert. Für die Datenspeicherung wird eine SQL-Datenbank verwendet.
Um eine hohe Verfügbarkeit zu garantieren, können mehrere Server im Web-Farm-Hosting betrieben werden. Die SQL-Datenbank wird in diesem Fall mit mehreren Instanzen in Failover-Clustern betrieben. Damit keine Caching-Probleme auftreten, welche auftreten würden, wenn jeder Server im Web-Farming seinen eigenen Cache hat, wird ein Redis Cluster eingesetzt, welches in diesem Fall den Cache für das System verwaltet. Damit wird verhindert, dass ein Server von ihm gecachte Daten ausgibt, welche von einem anderen Server bereits in der

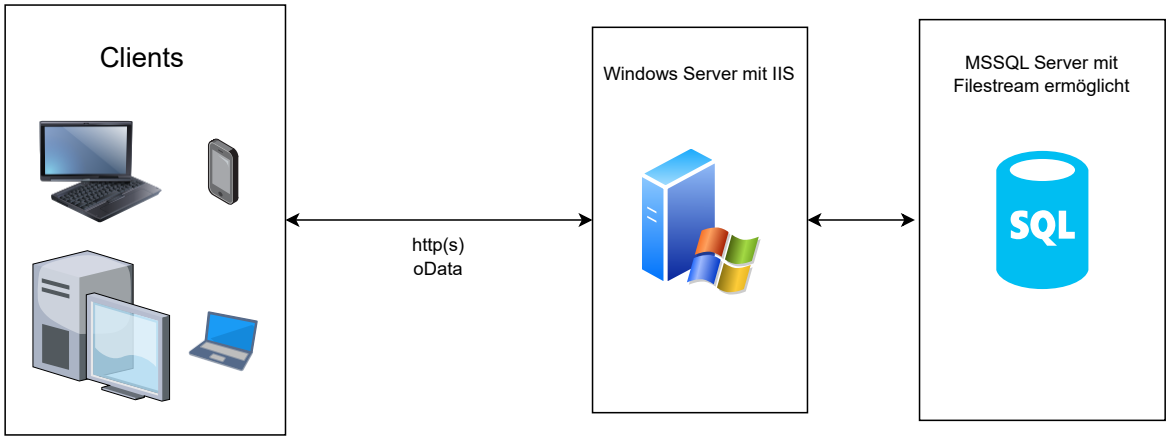


Abbildung 4.1.: L-mobile internes Diagramm der Client-Server-Architektur.

Datenbank geändert wurden. Dieses Setup ist in Abbildung 4.2 dargestellt.
Die Applikation ist als Monolith implementiert. Ihre Komponenten sind in verschiedene Module aufgeteilt. Die Applikation wird als eine Einheit installiert und verwaltet. Also ist die Service/Sales-Applikation von L-mobile ein modularer Monolith, wie in Kapitel 2.1 beschrieben.

Strategische Ziele

Die strategischen Ziele, welche L-mobile mit diesem Projekt der (Teil-) Migration in Microservices, verfolgt sind:

- Die Modularisierung der Software weiter vorantreiben.
- Kostenreduktion durch Verwendung günstiger Ressourcen.
- Die Codebasis auf einen möglichen Microservices-Betrieb hinführen.

Bei der Modularisierung verspricht sich L-mobile, dass die verschiedenen Teilbereiche des Codes einfacher und unabhängig voneinander bearbeitet werden können, wie von Parnas beschrieben [Par72]. Dies resultiert in einer besseren Wartbarkeit und Erweiterbarkeit des Codes und einzelne Module können isoliert voneinander entwickelt werden, wie Starke und Mazlami et al. beschreiben [MCL17, Sta15]. Außerdem lassen sich Module einfacher durch andere Module ersetzen, ohne dass weitere Module dahingehend angepasst werden müssen, beschreibt Starke [Sta15].
Je nach Kostenmodell der Cloud fallen weniger Kosten an, wenn z. B. nur ein Modul oder Microservice anstatt der ganzen Applikation skaliert werden kann, wie Villamizar et al. beschrieben haben [VGO+17]. Dadurch soll die Kostenreduktion erreicht werden.
Die Hinführung der Codebasis auf einen möglichen Microservices-Betrieb hat den Grund, dass für eine spätere vollständige Migration der Applikation in Microservices bereits eine Grundlage und Strategien existieren, auf denen aufgebaut werden kann. Außerdem existiert dann ein

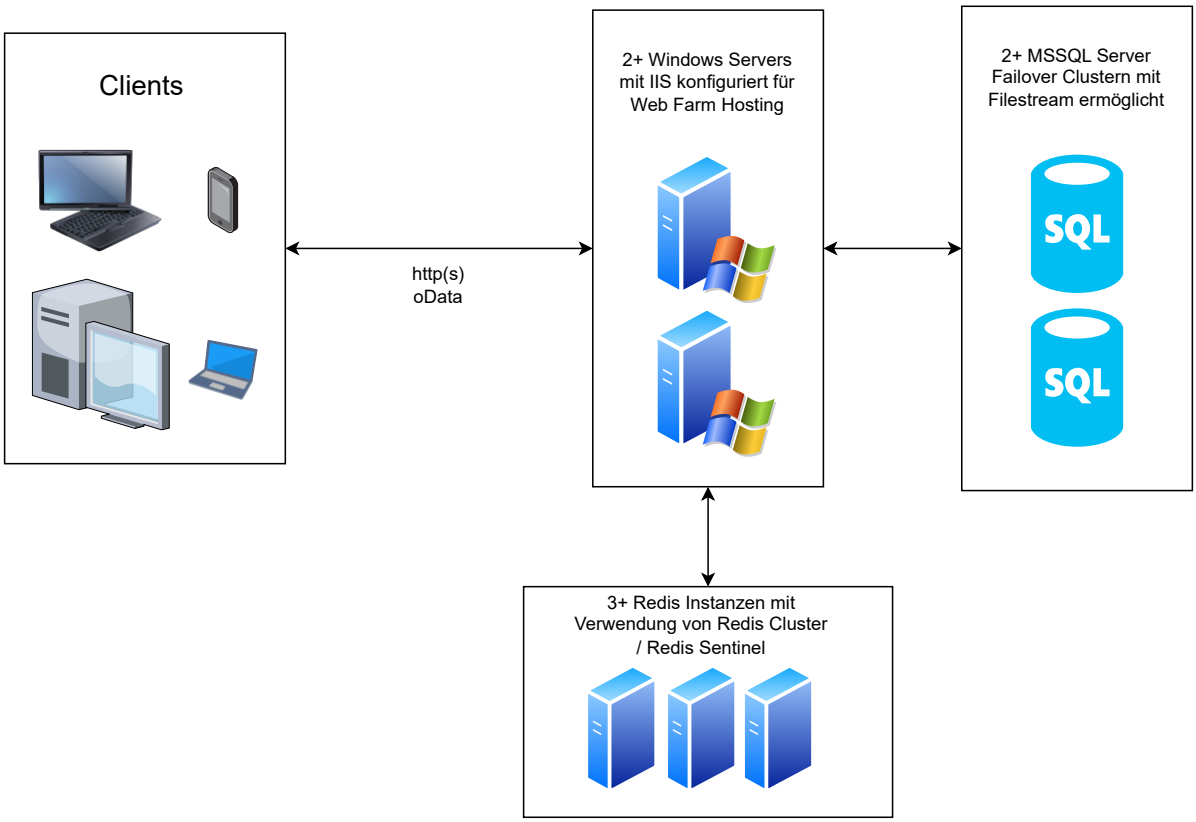


Abbildung 4.2.: L-mobile internes Diagramm der Client-Server-Architektur für hohe Verfügbarkeit.

Konzept, welches beweist, dass eine Migration der Service/Sales-Applikation in Microservices möglich ist. Damit lässt sich erreichen, dass die Migration ohne größere Schwierigkeiten ablaufen kann.

4.1.2. Bewertung

Der Workshop für die Architekturbewertung hatte eine Dauer von drei Stunden und 41 Minuten. Da alle Teilnehmer des Workshops noch keine Erfahrungen mit Architekturbewertungen hatten, gab es bei den Anfängen der Qualitätsattribut-Identifikation, Szenarien-Erhebung und Architekturanalyse Unklarheiten und Fragen darüber, was für eine Aufgabe sie in diesem Moment haben. Diese Unklarheiten und Fragen konnten jedoch von dem Moderator schnell beseitigt werden.
Die Identifizierung der Qualitätsattribute ging schnell vonstatten und während der Priorisierung gab es eine produktive Diskussion darüber, welche Qualitätsattribute wichtiger sind als andere. Da manche Teilnehmer der Ansicht sind, dass alle Qualitätsattribute gleich wichtig sind, musste vom Moderator klargestellt werden, dass diese Wertung nur der Architekturbewertung

dient und in der späteren Applikation alle Qualitätsattribute enthalten sein sollten.
Die Erhebung der Szenarien ging etwas zäh vonstatten, da viele Teilnehmer noch nie Szenarien erhoben haben und manchmal ein Messungskriterium vergessen wurde.
Die Diskussion darüber, ob die aktuelle Architektur die erhobenen Szenarien erfüllt, war anfangs etwas schwerfällig, da sich die Teilnehmer noch nicht vorstellen konnten, wie die Architektur mit diesen Szenarien bewertet werden kann. Jedoch kam auch in diesem Schritt nach ein paar Minuten eine produktive Diskussion zustande.
Während der Architekturbewertung wurden 17 Qualitätsattribute für neun Qualitäts-Charakteristiken identifiziert und priorisiert. Daraufhin wurden für die drei am höchsten priorisierten Attribute Szenarien erhoben.

Qualitätsattribute

Die Qualitätsattribute wurden auf Basis des ISO 25010[1] und auf Basis des Vorschlags von Auer et al. [ALFT21] identifiziert. Die identifizierten Qualitätsattribute sind in Abbildung 4.3 dargestellt.
Wie von Kazman et al. und Svahnberg und Mårtensson beschrieben, wurden eine Priorisierung der Qualitätsattribute durchgeführt [KKC00, SM07]. Die Priorität der Kategorien wurde durch eine Diskussion zwischen den Stakeholdern ermittelt. Dabei wurde angemerkt, dass eigentlich alle Attribute wichtig sind und implementiert werden sollten. Durch die Anmerkung des Moderators, dass die Priorisierung und die Einschränkung auf zwei Attribute pro Kategorie für die Architekturbewertung wichtig ist und später natürlich alle Attribute implementiert werden, konnten die Stakeholder von der Priorisierung der Attribute überzeugt werden.
Die am höchsten priorisierten Kategorien sind *Security*, *Reliability* und *Functional Suitability*. *Security* wurde am höchsten priorisiert mit den Attributen *Data Integrity* und *Authentication*. Da die Kunden von L-mobile andere Unternehmen sind und eine Hacker-Attacke oder Datenverlust bei gleichzeitiger Bearbeitung eines Datensatzes einen finanziellen Verlust für den Kunden bedeuten kann, ist das Qualitätsattribute *Data Integrity* sehr wichtig. Außerdem sollten manche Datensätze nur von Personen mit den korrekten Zugriffsrechten bearbeitet werden können, daher wurde das Qualitätsattribut *Authentication* identifiziert.
Für *Reliability*, der am zweit-höchsten priorisierten Kategorie, sind die Qualitätsattribute *Recoverability* und *Data Consistency* identifiziert worden. Da es wichtig ist, im Fehlerfall die Daten korrekt wiederherzustellen. Hierfür sollten die Daten konsistent sein.
Die am dritt-höchsten priorisierte Kategorie ist die *Functional Suitability*. Kunden erwarten bei einem Update der Applikation, dass alle Funktionen aus der alten Version verfügbar sind. Außerdem sollten bei gleichbleibenden Eingaben immer dieselben Ergebnisse erzielt werden.
Weitere Attribute sind Version *Consistency* und *Easiness* für *Maintainability*, *Hardware Requirement should not increase* und *No Data Loss* für *Compatibility*, *Resource utilization* und *Latency*

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

für *Performance*, *Users* und *Performance* für *Scalability*, *Compatible with CI* für *Testability* und *Cost for Scalability* und *Development Costs* für *Cost*.

Szenarien

Für die drei am höchsten priorisierten Qualitätsattribut-Kategorien wurden, wie in Svahnberg und Mårtensson [SM07] beschrieben, für jede Anforderung jeweils zwei Szenarien erhoben, was in insgesamt zwölf Szenarien resultierte. Die erhobenen Szenarien können in Tabelle 4.1 nachgelesen werden.
Die Szenarien wurden auch wie in ATAM von Kazman et al. [KKC00] bewertet. Die Bewertung der Szenarien orientierte sich an dem vom FMM und von Koch vorgeschlagenen Bewertungsschema [Ins23, Koc22]. Für die Wichtigkeit wurden die Buchstaben A, B und C vergeben. Dabei stehen die Buchstaben

A für sehr wichtig,
B für Medium wichtig,
C für weniger wichtig.

Die technische Schwierigkeit wurde mit den Zahlen 1, 2 und 3 bewertet. Dabei steht

1 für einfach/kein Risiko,
2 für normal,
3 für sehr schwierig/risikoreich.

Die erhobenen Szenarien wurden in einer Diskussion mit den vorgestellten Attributen bewertet.

Resultat

Die Teilnehmer der Architekturbewertung kamen zu dem Resultat, dass es zum aktuellen Zeitpunkt noch nicht nötig ist, die Applikation in Microservices zu migrieren, da die aktuelle Architektur alle Anforderungen an die Qualitätsattribute erfüllt. Auch mit allen identifizierten Problemen kommt die aktuelle Architektur zurecht. Es wurden zwei Szenarien erhoben, welche noch nicht von der bestehenden Architektur berücksichtigt werden, aber diese sind leicht in der bestehenden Architektur zu implementieren.
Trotzdem sind sich die Teilnehmer einig, dass es auf jeden Fall sinnvoll ist, sich bereits mit Microservices und Migrationsstrategien in eine MSA zu befassen, da es in der Zukunft wahrscheinlich wird, in diese Richtung zu gehen. Außerdem wird es wichtig sein zu wissen, ob eine Microservices-Applikation mit der Service/Sales-Applikation überhaupt realisiert werden kann. Es ist ebenso wichtig Konzepte, für die Migration und Infrastruktur-Themen für Microservices bereits erarbeitet zu haben, um eine zukünftige Migration erfolgreich und schnell durchzuführen. Ein erster Entwurf einer Microservices-Architektur für die Service/Sales-Applikation kann also sehr hilfreich für L-mobile werden.

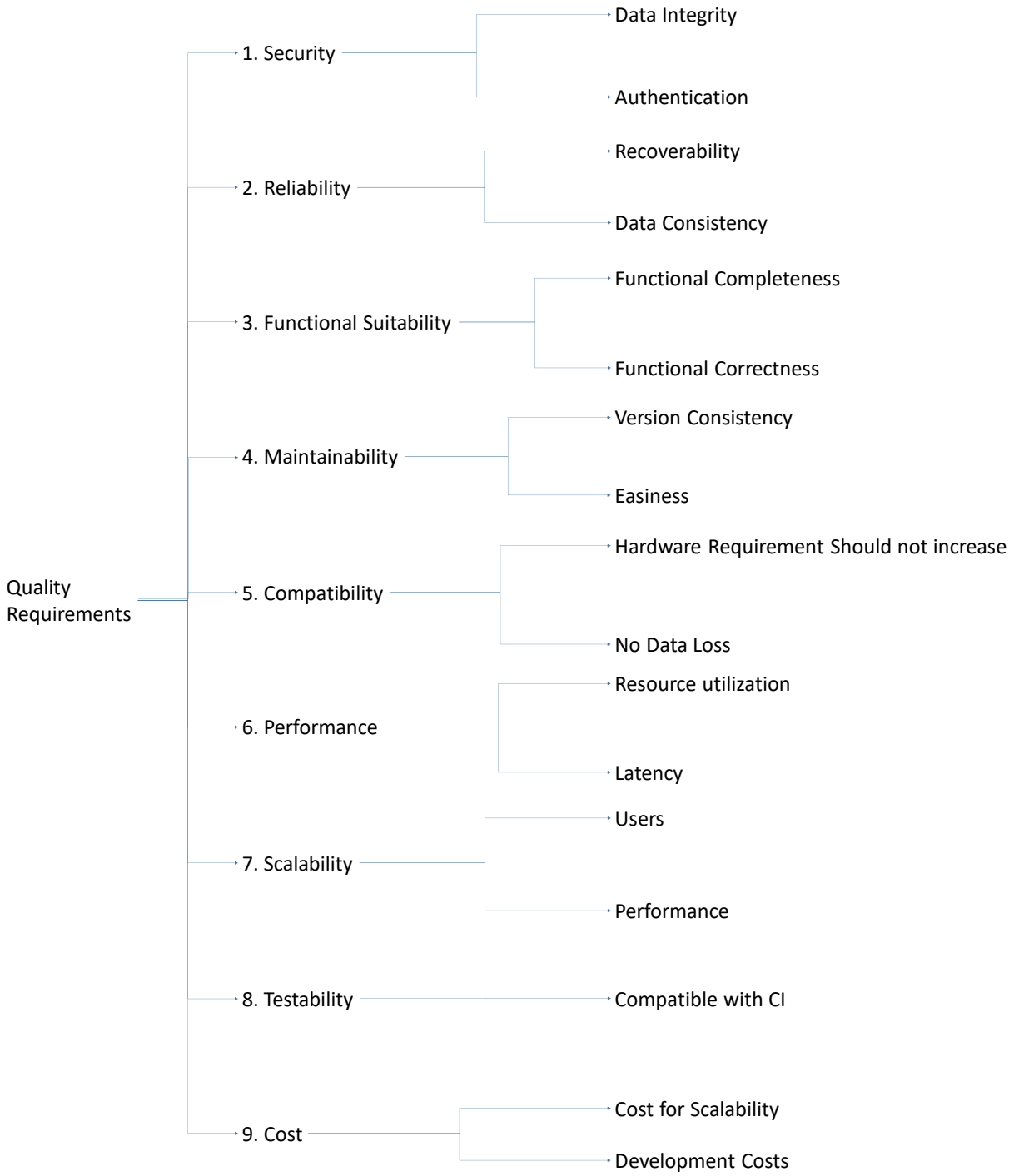


Abbildung 4.3.: Der Utility Tree mit den Qualitätsattributen der ersten Phase

Qualitäts Attribut	Sub-Attribut	Szenario
1. Security	Data Integrity	C2. Angreifer/Benutzer fügt bösartige Daten in das System ein. Die bösartige Daten werden vom System bereinigt/gesandboxt, bevor sie Schaden anrichten können.
		B3. Mehrere Benutzer bearbeiten denselben Eintrag im System. Die Architektur sollte implizit die korrekten Security-Attribute, ohne manuelle Interaktion, zuordnen.
	Authentication	A1. Benutzer will auf Daten außerhalb seiner Zugriffsrechte zugreifen. Das System verweigert den Zugriff/stellt sicher, dass die Zugriffskontrolle beim Speichern von Daten eingehalten wird.
		C1. Ein Entwickler implementiert ein Funktion/Bugfix, die Continuous Integration antwortet direkt nach dem Build mit einem Hinweis, wenn die Authentifizierung fehlt/unvollständig ist.
2. Reliability	Recoverability	A2. Bei Datenverlust sollte es dem Administrator möglich sein, das System innerhalb einer Minute auf einen konsistenten Zeitpunkt wiederherzustellen.
		A3. Wenn sich die zugrunde liegenden Daten des Systems ändern, sollten alle Subsysteme/Systemaspekte ordnungsgemäß und stillschweigend, beim nächsten Synchronisierungs-/Aktualisierungszyklus, auf den neuen Datensatz wiederhergestellt werden ohne dass der Benutzer eingreifen muss.
	Data Consistency	B1. Der Benutzer erwartet, dass die Daten auf dem Bildschirm, welche aus verschiedenen Subsystemen kommen, konsistent sind.
		A1. Auf der Serverseite sollten die Daten stark konsistent sein, wenn sie von mehreren Subsystemen gemeinsam genutzt werden.
3. Functional Suitability	Functional Completeness	C3. Der Kunde erwartet alle Funktionalitäten des alten Systems.
		C1. Die API-Funktionalität sollte mit der Funktionalität in der Anwendung identisch sein.
	Functional Correctness	A1. Alle Funktionen antworten mit den gleichen Ergebnissen wie zuvor, wenn die Eingabe gleich bleibt.
		A1. API-Ergebnisse führen zu denselben Daten wie die Anwendungsergebnisse, wenn die Eingabe gleich ist.

Tabelle 4.1.: Die Szenarien für die drei am höchsten priorisierten Qualitätsattribut Kategorien.

Wegen dieser Gründe wird die Arbeit an der Migration der L-mobile Applikation in Microservices im Rahmen eines PoC weitergeführt.

4.2. Phase 2

Die Phase 2 besteht, wie in Fritzsch et al. beschrieben [FBH+22], aus den Schritten: Migrationsstrategie definieren und Service-Identifikationsansatz definieren. Für die Auswahl der Migrationsstrategien und Service-Identifikationsansätze aus den über 90 Veröffentlichungen, welche das FMM zum Zeitpunkt der Verfassung dieser Arbeit vorschlägt, ist eine Filterung nötig. Diese Filterung ist in Kapitel 4.2.1 beschrieben.
Da in der späteren Umsetzung der Strategien und Ansätze mehrere Service-Identifikationsansätze und Migrationsstrategien zum Einsatz kommen können, ist in Kapitel 4.2.2 und Kapitel 4.2.3 beschrieben, wie sich die Service-Identifikation und die Migrationsstrategie aus den gefilterten Veröffentlichungen zusammenstellt.

4.2.1. Migrationsstrategie und Service-Identifikationsansatz auswählen

Für die Auswahl einer geeigneten Migrationsstrategie und eines Service-Identifikationsansatzes, stellt das FMM zum Zeitpunkt, zu dem diese Arbeit verfasst wird, eine Tabelle mit 90 Migrationsstrategien und Service-Identifikationsansätzen zur Verfügung. Damit die Strategien und Ansätze effizient gefiltert werden können, besitzt die zur Verfügung gestellte Tabelle bereits hilfreiche Informationen, die von Fritzsch et al. [FBZW19] und Koch [Koc22] erarbeitet wurden, über die Strategien und Ansätze. Diese Informationen sind übersichtlich über verschiedene Spalten aufgeteilt. Die Spalten und deren Bedeutung sind in Tabelle 4.2 aufgelistet.
Mithilfe der Taxonomie von Service-Identifikationsansätzen von Abdellatif et al. [ASM+21] konnten diese Ansätze gefiltert werden. Die für L-mobile relevanten Taxonomien sind in Abbildung 4.4 zu sehen. Für eine bessere Übersicht wurden in Abbildung 4.4 die Sub-Qualitätsattribute der Qualitätsanforderungen nicht mit aufgenommen, es sind aber dieselben Attribute, wie sie in Abbildung 4.3 dargestellt sind.
Mithilfe der Informationen aus Tabelle 4.2 und der Taxonomie aus Abbildung 4.4 konnte eine erste Filterung durchgeführt werden. In der ersten Filterung wurde nach Ansätzen Ausschau gehalten, welche die von L-mobile geforderten Qualitätsattribute, Abbildung 4.3, berücksichtigen. Außerdem wurde in dieser Filterung darauf geachtet, dass in den Notizen der Tabelle nicht die Anmerkung „Nur für Java“ auftaucht, da die L-mobile Applikation eine .Net Applikation ist. Das letzte Kriterium dieser Filterung war, dass sich der Ansatz als Refactoring ausführen lässt. Das Resultat dieser ersten Filterung, welche Publikationen verfügbar sind und wie viele der geforderten Qualitätsattribute und Sub-Qualitätsattribute diese berücksichtigen, ist in Tabelle 4.3 dargestellt.

Name	Bedeutung
ID	Eine eindeutige Identifikationsnummer für jede Publikation.
Year	Das Veröffentlichungsjahr der Publikation.
Publication	Titel der Publikation.
Refactor/ Rebuild/ New	Die Strategie, die bei der Entwicklung der Services angestrebt wird. Bei Refactor wird die bestehende Anwendung umstrukturiert, bei Rebuild wird die bestehende Applikation neu erstellt und bei New wird eine komplett neue Anwendung entwickelt.
Type	Der Typ der Migrationsstrategie, des Service-Identifikationsansatzes. Eine kurze Wiedergabe, wie genau die Veröffentlichung bei der Migration/Identifikation vorgeht.
QA	Die QAs, die durch die Strategie in der Publikation verbessert werden [Koc22].
SubA	Falls ein spezielles Sub-Qualitätsattribute (SQAs) der betroffenen QAs im Besonderen in der Publikation behandelt wird, ist es in dieser Spalte aufgelistet.
SP	System-Eigenschaften (SPs) wie Granularität, Kopplung, Zusammenhalt, Isolation, Autonomie, Komplexität, technische Heterogenität und ob sie durch die Strategie der Veröffentlichung erhöht(+) oder verringert(-) wird.
Tool Support	Angabe, ob für die beschriebenen Prozesse ein Tool existiert oder sich ein Tool in der Konzeptphase befindet.
Other Notes	Andere Notizen zu der Publikation, z. B. ob eine Strategie oder Ansatz nur für eine bestimmte Programmiersprache verfügbar ist.
Reference Application	Applikationen, die in der Publikation referenziert werden.
Classification	Die Klassifizierung der Strategien und Ansätze [FBZW19]. Es existieren Statische Code Analyse (SCA), Meta-Daten getrieben (MDA), Arbeitslast (Workload) Daten getrieben, Dynamische Microservice Komposition und verschiedene Hybridansätze dieser Klassifizierungen.

Tabelle 4.2.: Die Spaltennamen für die Tabelle mit den Migrationsstrategien und Service-Identifikationsansätze wie von Fritzsch et al. [FBZW19] erarbeitet.

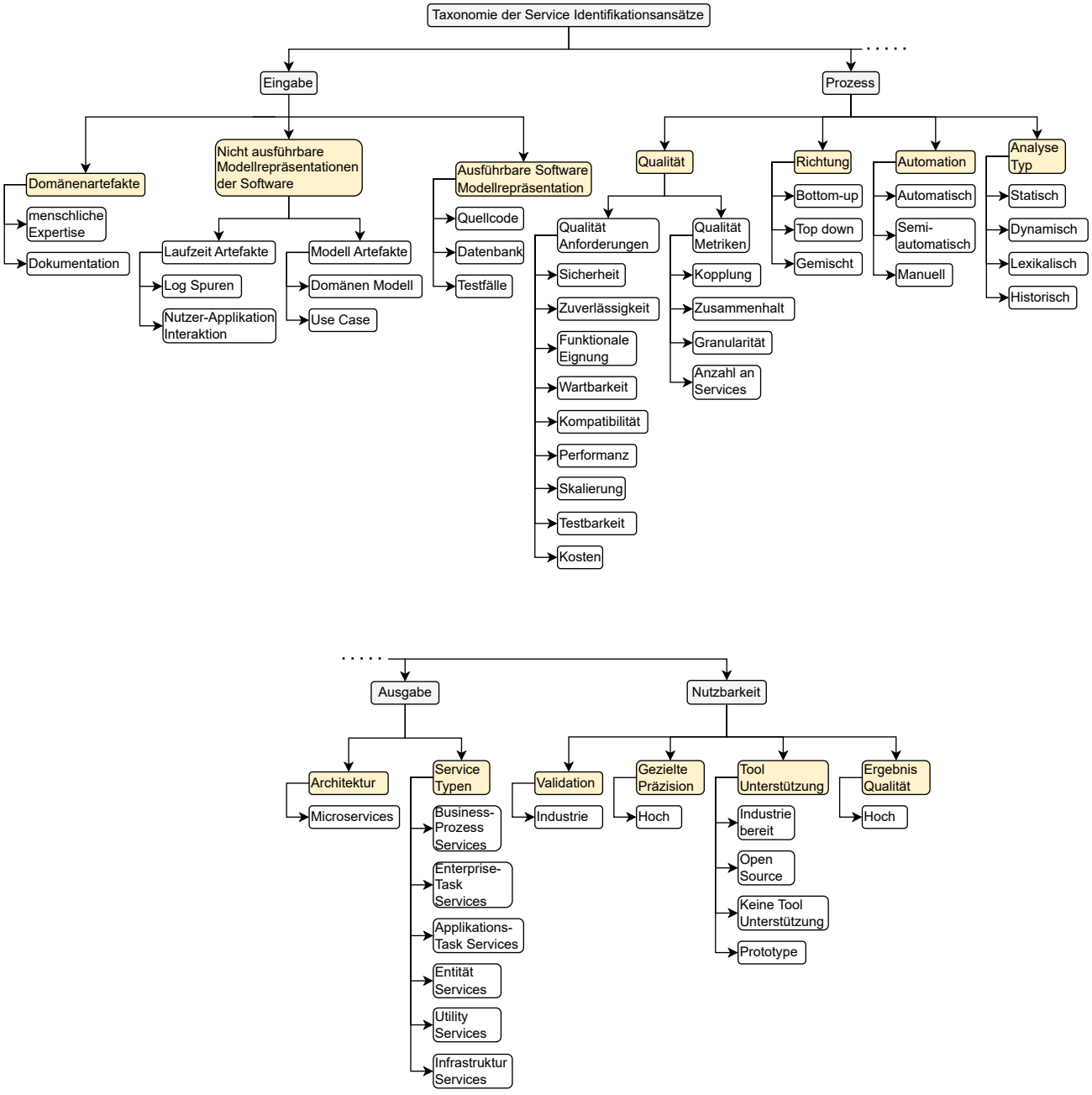


Abbildung 4.4.: Die Taxonomie von Serivce-Identifikationsansätzen für die L-mobile Applikation

Anzahl der berücksichtigten QAs	Publikationen
1	[RR22], [AM21], [DBT21], [FSC+21], [BBGG20], [BCS21], [DMF+20], [LO20], [BM20], [JLC+19], [EB18], [NUEH18], [CLL17], [HAB17], [DB22]
2	[KMK22], [AM22], [ACC+21], [CCRZ20], [DEF+21], [IT21], [VPAG21], [LML20], [SSR19], [BBM18], [DBFP18], [PP18], [RWW+18], [KVGJ17], [AI16], [ECA+16]
3	[ACC+22], [DBFP21], [ZSS+21], [CGC+20b], [CGC+20a], [PKY20], [De 19], [HLT18]
4	[DKTT22], [GCL+20], [AIE19]
7	[SK17]

Tabelle 4.3.: Die vom FMM vorgeschlagenen Migrationsstrategien und Service-Identifikationsansätze nach der Filterung nach Qualitätsattributen.

Anzahl der berücksichtigten QAs	Publikationen
1	[AM21], [DBT21], [FSC+21], [BBGG20], [LO20], [JLC+19], [EB18], [NUEH18], [HAB17], [DB22]
2	[AM22], [ACC+21], [CCRZ20], [IT21], [BBM18], [DBFP18], [RWW+18], [KVGJ17], [ECA+16]
3	[ACC+22], [DBFP21], [ZSS+21], [CGC+20b], [CGC+20a], [HLT18]
4	[DKTT22], [AIE19]
7	[SK17]

Tabelle 4.4.: Die vom FMM vorgeschlagenen Migrationsstrategien und Service-Identifikationsansätze nach der Filterung nach Qualitätsattributen und ohne reine MDA Analysen.

Metadaten wie verschiedene UML-Diagramme sind für die L-mobile Applikation kaum verfügbar und der Zustand dieser Metadaten ist unklar. Daher wurde in der zweiten Filterung darauf geachtet, dass Strategien und Ansätze, welche nur eine MDA-Analyse des Altsystems verfolgen, nicht mehr berücksichtigt werden. Die nach dieser zweiten Filterung verfügbaren Migrationsstrategien und Service-Identifikationsansätze sind in Tabelle 4.4 dargestellt.
Da nach der zweiten Filterung immer noch 28 Strategien und Ansätze verfügbar sind, musste eine weitere Filterung durchgeführt werden. In dieser Filterung wurden Strategien und Ansätze beibehalten, welche die von L-mobile gewünschten System-Eigenschaften, Kopplung, Zusammenhalt, Granularität oder Komplexität positiv beeinflussen. Eine Übersicht über die

nach der zweiten Filterung übrig gebliebenen Veröffentlichungen und ihr Einfluss auf die System-Eigenschaften kann in Tabelle 4.5 nachgelesen werden.
Die nach dieser dritten Filterung übrig gebliebenen Strategien und Ansätze sind in Tabelle 4.6 dargestellt. Von den übrig gebliebenen Strategien und Ansätzen verfolgen [AM21], [DBT21] und [HLT18] eine reine SCA, [DBFP18] eine statische Code Analyse und Arbeitslast (Workload) Daten getriebener Hybrid Analyse und [CGC+20b] eine statische Code Analyse und Meta-Daten getriebener Hybrid Analyse.
Neben den nach der dritten Filterung übrig gebliebenen Strategien und Ansätzen (Tabelle 4.6) werden für die Service-Identifikation und die Migration der Service/Sales-Applikation von L-mobile auch die Veröffentlichungen [LML20] und [SK17] berücksichtigt.
Die Veröffentlichung von Li et al. [LML20] wird weiterhin berücksichtigt, obwohl sie durch den Filterungsprozess aussortiert wurde, da sie das *Strangler Fig* Pattern berücksichtigt, welches eine gute Strategie bildet, um einen Monolithen nach und nach in eine MSA zu überführen. Da diese Veröffentlichung jedoch eine MDA Analyse zur Service-Identifikation vorschlägt, wird aus dieser Veröffentlichung nur die Migrationsstrategie berücksichtigt.
Die Veröffentlichung von Shashwat und Kumar [SK17] wird weiterhin berücksichtigt, obwohl sie durch den Filterungsprozess aussortiert wurde, da sie sieben der geforderten QAs und SQAs berücksichtigt. Außerdem werden in dieser Veröffentlichung Qualitätsattribute wie z. B. Sicherheit und Zuverlässigkeit berücksichtigt, welche in der Architekturanalyse als die wichtigsten Qualitätsattribute eingestuft wurden, aber von den anderen übrig gebliebenen Veröffentlichungen bisher noch nicht berücksichtigt werden. Da diese Veröffentlichung eine SCA verfolgt, kann sie uneingeschränkt angewendet werden.
Von den vorgeschlagenen Veröffentlichungen enthalten [DBT21], [CGC+20b], [DBFP18] und [SK17] einen reinen Service-Identifikationsansatz. [AM21], [LML20] und [HLT18] enthalten einen Service-Identifikationsansatz mit einer Migrationsstrategie.
Eine weitere Tabelle mit Publikationen, welche Tools für die Service-Identifikation vorstellen, wurde vom FMM bereitgestellt. In dieser neuen Tabelle existieren auch neue Eigenschaften der Tools, welche von Fritzsch et al. [FBZW19] erarbeitet wurden. Die neuen Eigenschaften, wie von Fritzsch et al. [FBZW19] beschrieben, sind: ein Link zu dem Tool, die Lizenz des Tools, der Ansatz den das Tool verfolgt, die Eingabe und Ausgabe des Tools und wie das Tool evaluiert wurde. Die Publikationen aus der neuen Tabelle sind [SSO22], [FFC21], [DBFP21], [KXK+21], [MBDT21], [SP21], [WYPZ20], [ZSS+21], [ZZ21], [BCS21], [BSG20], [KZH+20], [NSR19], [PAM19], [NUEH18], [THLL18], [BGD17], [MCL17], [GKGZ16], [KBAR21], [TAA+22] und [MCF+20].
Aus diesen Publikationen wurden [FFC21], [KXK+21], [SP21], [BCS21], [BSG20], [NSR19] und [PAM19] exkludiert, da sie nur für Java-Applikationen konzipiert sind.
Die übrig gebliebenen Publikationen, [SSO22], [DBFP21], [MBDT21], [WYPZ20], [ZSS+21], [ZZ21], [KZH+20], [NUEH18], [THLL18], [BGD17], [MCL17], [GKGZ16], [KBAR21], [TAA+22] und [MCF+20] wurden weiter analysiert.
Weiterhin wurden Publikationen nicht berücksichtigt, bei denen eine Ausführung oder Inspektion des Tools vom ISTE ESE nicht erfolgreich war. Nach diesem Schritt konnten die Publikationen, [SSO22], [DBFP21], [MBDT21], [WYPZ20], [ZSS+21], [ZZ21], [KZH+20], [NUEH18], [THLL18], [MCL17], [GKGZ16] und [MCF+20] extrahiert werden.

Publikationen	positiv beeinflusste SPs	negativ beeinflusste SPs	berücksichtigte SPs
[AM21]	Zusammenhalt	-	-
[DBT21]	Zusammenhalt, Kopplung	-	-
[FSC+21]	-	-	Granularität
[BBGG20]	-	-	-
[LO20]	-	-	Kopplung
[JLC+19]	-	-	-
[EB18]	-	-	Kopplung
[NUEH18]	-	-	-
[HAB17]	-	-	Granularität
[DB22]	-	-	-
[AM22]	-	-	-
[ACC+21]	-	-	Kopplung, Zusammenhalt
[CCRZ20]	-	-	Granularität
[IT21]	-	-	-
[BBM18]	-	-	-
[DBFP18]	Zusammenhalt, Kopplung	-	-
[RWW+18]	-	-	Kopplung, Zusammenhalt
[KVGJ17]	-	-	-
[ECA+16]	-	-	Kopplung, Zusammenhalt
[ACC+22]	-	-	Kopplung, Zusammenhalt
[DBFP21]	-	-	Granularität
[ZSS+21]	-	-	-
[CGC+20b]	Zusammenhalt, Kopplung	-	-
[CGC+20a]	-	-	Kopplung, Zusammenhalt
[HLT18]	Kopplung	-	-
[DKTT22]	-	-	-
[AIE19]	-	-	-
[SK17]	-	-	-

Tabelle 4.5.: Die Veröffentlichungen nach der zweiten Filterung und ihr Einfluss auf die System-Eigenschaften.

Anzahl der berücksichtigten QAs	Publikationen
1	[AM21], [DBT21]
2	[DBFP18]
3	[CGC+20b], [HLT18]

Tabelle 4.6.: Die vom FMM vorgeschlagenen Migrationsstrategien und Service-Identifikationsansätze nach der Filterung nach Qualitätsattributen, System-Eigenschaften und ohne reine MDA Analysen.

Weiterhin wurde nach allen Publikationen gefiltert, welche zumindest als Teil der Eingabe den Quellcode oder eine API Spezifikation verlangen. Nach diesem Schritt konnten die Publikationen [WYPZ20], [ZSS+21], [ZZ21], [KZH+20], [NUEH18], [MCL17] und [MCF+20] extrahiert werden.
Mazlami et al. [MCL17] wurde nicht berücksichtigt, da sie hauptsächlich die Versionskontrolldaten analysieren. Der Feature Table Ansatz von Wei et al. [WYPZ20] wurde nicht berücksichtigt, da die L-mobile Applikation sehr viele Features besitzt.
Die übrigen Publikationen [ZSS+21], [ZZ21], [KZH+20], [NUEH18] und [MCF+20] wurden eingehender analysiert, ob und wie sie sich auf die L-mobile Applikation anwenden lassen.
[MCF+20] wurde abgelehnt, da es nur für Anwendungen, die in Django implementiert sind, anwendbar ist.
Durch diese Filterung lässt sich nun Forschungsfrage 3: Welche Refactoring-Verfahren und Migrationsverfahren eignen sich am besten für die Dekomposition des L-mobile Monolithen in eine Microservices-Architektur, beantworten. Die laut dem FMM am besten geeigneten Migrationsverfahren und Service-Identifikationsansätze für die Dekomposition des L-mobile Monolithen sind [AM21], [DBT21], [DBFP18], [CGC+20b], [HLT18], [LML20], [SK17], [ZSS+21], [ZZ21], [KZH+20] und [NUEH18].

4.2.2. Resultierender Service-Identifizierungsansatz

Aus den elf Publikationen [AM21], [DBT21], [DBFP18], [CGC+20b], [HLT18], [LML20], [SK17], [ZSS+21], [ZZ21], [KZH+20] und [NUEH18] wurde eine Publikation, die für den weiteren Verlauf dieser Arbeit den Service-Identifikationsansatz definiert, ausgewählt. Da die Service/Sales-Applikation von L-mobile mit ca. 300.000 Lines of Code (LoC) und 4810 Klassen (C#, JavaScript, HTML, TypeScript, etc.), laut der Analyse von SonarQube[2], umfangreich ist, wird ein toolunterstützer Service-Identifizierungsansatz bevorzugt. Durch ein Tool ist der Schritt der Service-Identifizierung wiederholbar. Mit einer manuellen Identifikation der Services kann es

[2] https://www.sonarsource.com/products/sonarqube/

höhere Abweichungen bei Wiederholungen geben.
Da der Ansatz von Li et al. [LML20] ein MDA ist, wird diese Veröffentlichung bei der Definition des Service-Identifizierungansatzes nicht betrachtet. Das Tool von Nakazawa et al. [NUEH18] wird in ihrer Publikation nicht referenziert, daher fällt auch dieser Ansatz weg.
Aus den gefilterten Publikationen qualifizierten sich [KZH+20][3,4,5], [DBFP18][6], [TAA+22][7] und [DBT21][8], da diese mit einem frei verfügbaren Tool Services für verschiedene Programmiersprachen identifizieren können. Für die Tools von [DBFP18], [TAA+22] und [DBT21] existierte wenig Dokumentation, was die erfolgreiche Ausführung dieser Tools auf die Service/Sales-Applikation von L-mobile verhinderte. Daher war es schlussendlich möglich den Ansatz von Krause et al. [KZH+20] nach der Filterung durch das FMM zu extrahieren.
Um sicherzugehen, dass durch die Filterung des FMM keine potenziell interessanten Veröffentlichungen ausgeschlossen wurden, wurde eine manuelle Analyse der Tools vorgenommen. Ziel dieser Analyse war es, Tools zu identifizieren, die für die Migration der L-mobile Applikation eingesetzt werden können, aber von der Filterung des FMM ausgeschlossen wurden. Mit den identifizierten Tools kann eine Optimierung des Filterungsprozesses des FMM vorgenommen werden. Dabei wurden Veröffentlichungen berücksichtigt, die durch das FMM gefiltert wurden, aber nicht durch die Tatsache exkludiert wurden, da sie nur für Java-Applikationen implementiert wurden oder reine MDAs sind.
Wie in Kapitel 4.2.1 beschrieben, wurden die Publikationen [LML20] und [SK17] weiter betrachtet, obwohl sie bereits durch die Filterung des FMM exkludiert wurden. Weitere Veröffentlichungen, welche in diesem Schritt eingehender analysiert wurden sind [BGD17], [MBDT21], [NUEH18], [DBFP21], [ZSS+21], [WYPZ20], [LO20], [GKGZ16], [ZZ21] und [KBAR21].
Um auch weitere, vom FMM bisher noch nicht berücksichtigte Service-Identifikationsansätze zu betrachten, wurde aus der Veröffentlichung von Akkaya und Ovatman [AO22] weitere Tools betrachtet. Dazu zählen [GCD+17a, GCD+17b, Gra17] und [DBPF18, De 18]. Weitere Publikationen, die sowohl im FMM als auch in der Veröffentlichung von Akkaya und Ovatman [AO22] vorkommen, wie [Klo22, KVGJ17] wurden ebenfalls berücksichtigt.
MicADO von Klock et al. [Klo22, KVGJ17] ist ein Tool, welches eine bestehende Microservicesverteilung optimiert und wurde deshalb nicht berücksichtigt. Für Subtype von De Alwis et al. [DBPF18, De 18] existierte zu wenig Dokumentation, um das Tool korrekt anzuwenden.
MicroART von Granchelli et al. [GCD+17a, GCD+17b, Gra17] ist ein Tool, um die Architektur von Microservices Systemen zu regenerieren und benötigt GitHub Repositories, die Docker [9] Container verwenden als Eingabe. Da die Service/Sales-Applikation von L-mobile nicht in einem GitHub Repository gehalten wird und keine Docker Container verwendet, kam MicroART für die L-mobile Applikation nicht zur Anwendung. Wem von Kirby et al. [KBAR21]

[3] https://structure101.com/
[4] https://www.explorviz.dev/
[5] https://dbeaver.io/
[6] https://github.com/AnuruddhaDeAlwis/NSGAII
[7] https://drive.google.com/drive/folders/1UsHvl-A-eAi7V3WIgEaSDDFARzu5u_rN
[8] https://github.com/utkd/cogcn
[9] https://www.docker.com/

wurde ausgeführt, es wurde jedoch keine Spezifikation der Eingabe gefunden, weshalb diese Anwendung nicht eingesetzt werden konnte. Es gibt keinen Link oder Hinweis, der zur Implementierung der Ansätze von Zhao und Zhao [ZZ21], Zaragoza et al. [ZSS+21], Mathai et al. [MBDT21] und Nakazawa et al. [NUEH18] führt. Service Cutter von Gysel et al. [GKGZ16] benötigt ERM-Diagramme, die für die L-mobile Applikation nicht vorhanden sind. Steinmetz von Löhnertz und Oprescu [LO20] unterstützt derzeit nur die Serviceidentifizierung von Java Monolithen. Die URL, welche von de Alwis et al. [DBFP21] für ihren KMeans Algorithmus angegeben wurde, kann nicht aufgelöst werden.
Aus den oben genannten Publikationen eignet sich die Service-Identifikation von Krause et al. [KZH+20] für die Service-Identifikation für die L-mobile Applikation. Für die anderen Service-Identifizierungsansätze wurde entweder das beschriebene Tool nicht gefunden, der Prototyp war für eine Analyse von Java-Applikationen implementiert, oder es existierte zu wenig Dokumentation, um das Tool effektiv zum Einsatz zu bringen.

4.2.3. Resultierende Migrationsstrategie

Aus den selben elf Publikationen die in Kapitel 4.2.2 genannt wurden, wurde eine Publikation, welche für den weiteren Verlauf dieser Arbeit die Migrationsstrategie definiert, ausgewählt.
Die Migrationsstrategie von Li et al. [LML20] wurde für die in dieser Arbeit zum Einsatz kommende Migrationsstrategie ausgewählt, um in der weiteren Migration verwendet zu werden, da dieser nach dem *Strangler Fig* Pattern handelt. Sie migrieren einen Service aus dem Monolithen in die Strangler Applikation, der Teil der Applikation welcher die migrierten Services enthält, in sieben schritten. Als erstes identifizieren sie die Services mit dem Domain-Driven Design. Anschließend führen sie vorbereitende Arbeiten durch, Li et al. haben in diesem Schritt eine *Service Registry* und ein *API Gateway* implementiert. Als nächstes priorisieren sie die Services bevor sie die Services implementieren, integrieren und testen. Zum Schluss entfernen Li et al. das migrierte Modul aus dem Monolithen indem sie den Monolithen so rekonstruieren, dass er mit dem migrierten Service kommuniziert. Anschließend führen Li et al. [LML20] den Migrationsprozess, ab der Implementierung, für den nächsten Service aus. Der Migrationsprozess von Li et al. ist in Abbildung 4.5 dargestellt. Da die L-mobile Applikation in dieser Arbeit nicht vollständig migriert werden soll, ist es wichtig, den noch nicht umstrukturierten Monolith parallel mit den Microservices betreiben zu können.
Durch diese Auswahl lässt sich ein weiterer Teil der Forschungsfrage 4 beantworten. Zu den L-mobile spezifischen Herausforderungen der Migration zählt, dass es kaum ein Tool für die Serviceidentifizierung gibt, welches für .Net Applikationen implementiert wurde. Eine weitere Herausforderung sind die kaum dokumentierten Tools.

Phase 2 des FMM war hilfreich, da hier viele Migrationsstrategien und Service-Identifikationsansätze effizient gefiltert werden konnten. Dennoch fehlen in den vorhandenen Materialien Informationen zu den jeweiligen Veröffentlichungen, bzw. sind unvollständig. Eine Filterung der Migrationsstrategien nur durch die drei am höchsten priorisierten Qualitätsattribute kann jedoch dazu führen, dass geeignete Strategien ausgefiltert werden.

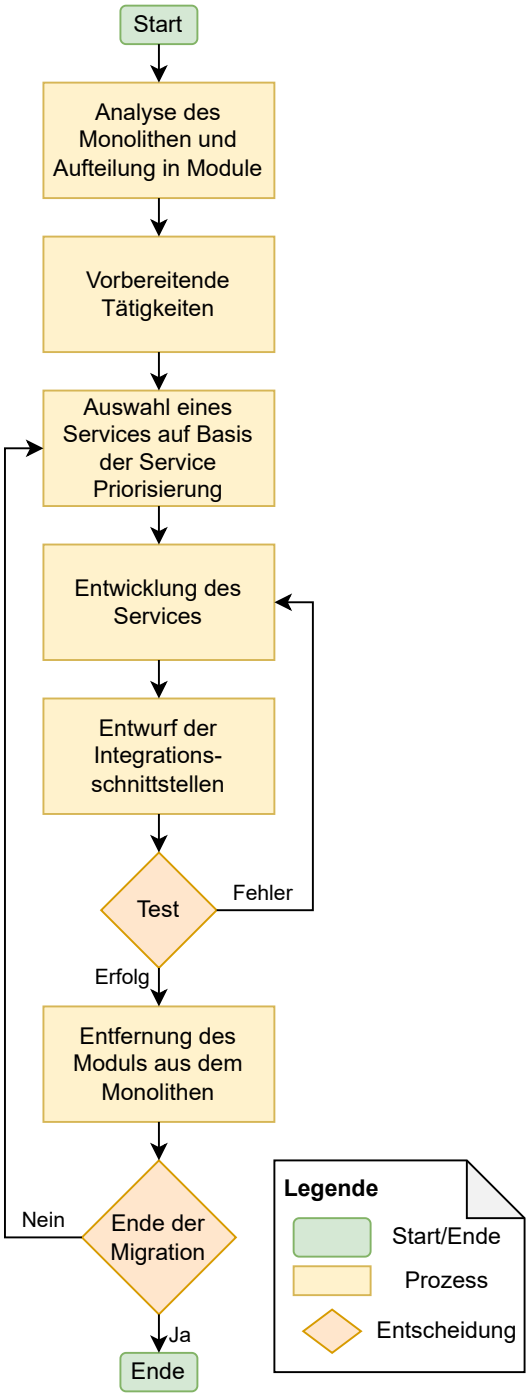


Abbildung 4.5.: Der Migrationsprozess von Li et al., übersetzt und umschrieben, original aus Li et al. [LML20].

Daher sollte in der Filterung auch der Kontext des zu migrierenden Systems und der Kontext der Migration betrachtet werden.

4.3. Phase 3

Phase 3 des FMM ist, wie Fritzsch et al. [FBH+22] beschreiben, in die Aufgaben Definition der Architektur und Service-Implementierung unterteilt. Dieses Kapitel ist für eine bessere Übersicht in Kapitel 4.3.3 Definition der Architektur und Kapitel 4.3.4 Service-Implementierung unterteilt. In Kapitel 4.3.3 Definition der Architektur befinden sich die Schritte der Service-Identifizierung und der Definition der Zielarchitektur, so wie Fritzsch et al. [FBH+22] die Unterteilung der Phase 3 durchgeführt haben. In Kapitel 4.3.4 Service-Implementierung befinden sich die Service-Priorisierung und die Service-Implementierung, so wie Fritzsch et al. [FBH+22] die Unterteilung der Phase 3 durchgeführt haben.
Ein Teil der vorbereitenden Aufgaben, wie von Li et al. [LML20] beschrieben, wurde schon vor der Service-Identifizierung durchgeführt und besteht, anders als bei Li et al., darin die Patterns und Best Practices zu filtern.

4.3.1. Filterung der Patterns

Die Hauptquellen für die Beschreibung der Microservices Patterns sind Microservice.io [Ric23i], so wie die Veröffentlichungen [TLP18, VLC19, VLL+20]. Das Ergebnis der Filterung ist in Tabelle A.1 in Anhang A dargestellt.
Das FMM stellt für die Analyse verschiedener Microservices Patterns eine Excel-Tabelle bereit. Um mithilfe dieser Tabelle die Patterns effizient zu filtern, wurden von Koch [Koc22] die Patterns kategorisiert. Daten wie die Qualitätsattribute, SQAs und System-Eigenschaften, welche durch das Anwenden dieses Patterns verbessert werden, aber auch Trade-offs, Qualitätsattribute und System-Eigenschaften, welche durch die Anwendung des Patterns verschlechtert werden können, wurden von Koch den Patterns zugeordnet und in der Tabelle angegeben. Als Quellen für die Beschreibung der Patterns wurden folgende Webseiten und Veröffentlichungen herangezogen: Microservices.io [Ric23i], so wie die Veröffentlichungen [TLP18, VLC19, VLL+20].
Um herauszufinden, welche Patterns für die Migration der Service/Sales-Applikation in eine MSs-Applikation infrage kommen, wurde die Tabelle mehrmals analysiert und gefiltert. Die verschiedenen Schritte der Filterung der Patterns sind in Tabelle 4.7 erklärt. In der ersten Filterung wurden Patterns entfernt, welche Qualitätsattribute, Sub-Qualitätsattribute und System-Eigenschaften verbessern, die bei der Systemanalyse, Kapitel 4.1.2, eine niedrige Priorität erhalten haben. Es wurden nur Patterns weiter betrachtet, welche mindestens eine der drei am höchsten priorisierten Qualitätsattribute oder eines ihrer SQAs verbessern. Sollte es zu einem Trade-off zwischen Qualitätsattributen und System-Eigenschaften kommen, wurde für jedes Pattern individuell entschieden, ob es weiter betrachtet oder nicht berücksichtigt

Schritt	Beschreibung
1	Es werden nur Patterns weiter betrachtet, welche mindestens eine der drei am höchsten priorisierten QAs, oder eines ihrer SQAs, verbessern. Individuelle Entscheidung bei Trade-Offs.
2	Es wird das zu migrierende System betrachtet. Patterns, welche nicht in den Kontext des Systems passen werden exkludiert. In Schritt 1 exkludierte Pattern können wieder betrachtet werden, sollten sie in den Kontext des Systems passen.
3	Durch firmeninterne Vorgaben werden Patterns, welche sich auf Messaging oder Containerization beziehen, zurückgestellt und evtl. zu einem späteren Zeitpunkt betrachtet.
4	Patterns, welche zu komplex für die Architektur des PoC sind, werden zurückgestellt und evtl. zu einem späteren Zeitpunkt betrachtet.

Tabelle 4.7.: Beschreibung der verschiedenen Schritte bei der Filterung der Patterns.

wird. Das Ergebnis, ob ein Pattern in diesem Schritt exkludiert oder weiter betrachtet wird, ist in Tabelle 4.8 dargestellt.
Da nach der ersten Filterung noch viele Patterns übrig geblieben sind, die bei genauerer Betrachtung für die Service/Sales-Applikation irrelevant erscheinen und so manches Pattern exkludiert wurde, welches für die Service/Sales-Applikation relevant sein könnte, wurde eine zweite Filterung durchgeführt, in der nicht nur die Qualitätsattribute aus Kapitel 4.1.2 ausschlaggebend sind, sondern auch der Kontext des Systems. Das Ergebnis dieses Filterungsschrittes ist in Tabelle 4.9 angegeben.
Da firmeninterne Vorgaben bestehen, welche besagen, dass *Messaging* und *Containerization* zu einem späteren Zeitpunkt diskutiert werden sollen, gab es eine dritte Filterung, welche diese Patterns herausfiltert, damit sie zu einem späteren Zeitpunkt betrachtet werden können. Das Ergebnis des dritten Filterungsschrittes ist in Tabelle 4.10 angegeben.
Da das System, welches in dieser Arbeit als PoC, implementiert wird, noch keine hohe Komplexität besitzt, können weitere Patterns für die spätere Implementierung zurückgestellt werden. Die Patterns, welche in dem vierten Filterungsschritt zurückgestellt oder weiter betrachtet werden, sind in Tabelle 4.11 dargestellt.
Die Patterns *Auth-service*, *Bulkhead* sowie *Service locator* wurden in der erstellten Microservices-Architektur noch nicht berücksichtigt, da hier ebenfalls noch eine Rücksprache erforderlich ist, ob diese Patterns eingesetzt werden sollen. Außerdem wurde von einem Softwarearchitekten vorgeschlagen, das *Log Aggregator* Pattern zu berücksichtigen.
Es werden auch Patterns berücksichtigt, welche nicht durch das FMM vorgeschlagen wurden. Um die Implementierung von Microservices zu vereinfachen, werden die Patterns *Microservice Chassis* und *Microservice Template* berücksichtigt. In Tabelle 4.12 sind die zusätzlich gefundenen Patterns aufgelistet.

exkludierte Patterns	**weiter betrachtet**
Aggregator	*API Gateway*
Ambassador	*Asynchronous Completion Token*
Anti-Corruption Layer	*Asynchronous Messaging*
Backend for Frontend	*Asynchronous Query*
Change Code Dependency to Service Call	*Auth-Service*
Competing Consumers	*Bulkhead*
Compute Resource Consolidation	*Circuit Breaker*
Consumer-Driven Contracts	*Container*
Correlation ID	*CQRS*
DB Cluster	*DB is the Service*
Enable Continuous Integration (CI)	*DB per Service*
Gateway Aggregation	*Deploy Cluster and Orchestrate Containers*
Health Check	*Edge Server*
Leader Election	*Event Notification*
Local Sharing-Based Router	*Event Sourcing*
Log Aggregator	*External Configuration Store*
Microservice DevOps	*External Load Balancer*
Monitor	*Externalized Configuration*
Mulitple Service per Host	*Gatekeeper*
Page Cache	*Gateway Offloading*
Priority Queue	*Gateway Routing*
Request-Reaction	*Internal Load Balancer*
REST Integration	*Key-Value Store*
Result Cache	*Load Balancer/Load-Balancing*
Self-Containment of Services	*Local Database Proxy*
Service Registry + Message Bus Hybrid	*Pipes and Filters*
Shared DB Server	*Scalable Store*
Sidecar	*Secure Channel*
Single Service per Host	*Self-Contained Systems*
Static Content Hosting	*Service Discovery*
Strangler	*Service Locator*
Tolerant Reader	*Service Registry*
	Service Registry Client

Tabelle 4.8.: Patterns, welche in dem ersten Filterungsschritt exkludiert wurden (linke Spalte) und weiter betrachtet werden (rechte Spalte).

exkludierte Patterns	**weiter betrachtet**	**wieder betrachtet**
DB is the Service	*API Gateway*	*Aggregator*
DB per Service	*Asynchronous Completion Token*	*Correlation ID*
Event Sourcing	*Asynchronous Messaging*	*Enable CI*
External Configuration Store	*Asynchronous Query*	*Health Check*
External Load balancer	*Auth-Service*	*Monitor*
Key-Value Store	*Bulkhead*	*Mulitple Service per Host*
Local Database Proxy	*Circuit Breaker*	*REST Integration*
Scalable Store	*Container*	*Shared DB Server*
Self-Contained Systems	*CQRS*	*Strangler*
	Deploy Cluster and Orchestrate Containers	
	Edge Server	
	Event Notification	
	Externalized Configuration	
	Gatekeeper	
	Gateway Offloading	
	Gateway Routing	
	Internal Load Balancer	
	Load Balancer/Load-Balancing	
	Pipes and Filters	
	Secure Channel	
	Service Discovery	
	Service Locator	
	Service Registry	
	Service Registry Client	

Tabelle 4.9.: Patterns, welche in dem zweiten Filterungsschritt exkludiert wurden (linke Spalte), weiter betrachtet werden (mittlere Spalte) und wieder betrachtet werden (rechte Spalte).

zurückgestellt	**weiter betrachtet**
Asynchronous Completion Token	*Aggregator*
Asynchronous Messaging	*API Gateway*
Asynchronous Query	*Auth-Service*
Container	*Bulkhead*
Deploy Cluster and Orchestrate Containers	*Circuit Breaker*
Event Notification	*Correlation ID*
Pipes and Filters	*CQRS*
Secure Channel	*Edge Server*
	Enable CI
	Externalized Configuration
	Gatekeeper
	Gateway Offloading
	Gateway Routing
	Health Check
	Internal Load Balancer
	Load Balancer/Load-Balancing
	Monitor
	Mulitple Service per Host
	REST Integration
	Service Discovery
	Service Locator
	Service Registry
	Service Registry Client
	Shared DB Server
	Strangler

Tabelle 4.10.: Patterns, welche in dem dritten Filterungsschritt zurückgestellt wurden (linke Spalte) und weiter betrachtet werden (rechte Spalte).

Manche dieser Patterns, wie das *Externalized configuration* Pattern und *Enable CI* Pattern, sind bereits im aktuellen System implementiert. Es kann nicht garantiert werden, dass alle ausgewählten Patterns für die Implementierung nach dieser Arbeit relevant bleiben und es kann ebenfalls nicht ausgeschlossen werden, dass die ausgefilterte Patterns zu einem späteren Zeitpunkt relevant werden. Da die Services erst für die lokale Nutzung implementiert werden, werden nicht alle Patterns, die nach der vierten Filterung übrig geblieben sind, implementiert.

zurückgestellt	weiter betrachtet
Aggregator	*Circuit Breaker*
API Gateway	*Enable CI*
Auth-Service	*Externalized Configuration*
Bulkhead	*Internal Load Balancer*
Correlation ID	*Load Balancer/Load-Balancing*
CQRS	*REST Integration*
Edge Server	*Service Discovery*
Gatekeeper	*Service Locator*
Gateway Offloading	*Service Registry*
Gateway Routing	*Service Registry Client/Server*
Health Check	*Shared DB Server*
Monitor	*Strangler*
Mulitple Service per Host	

Tabelle 4.11.: Patterns, welche in dem vierten Filterungsschritt zurückgestellt wurden (linke Spalte) und weiter betrachtet werden (rechte Spalte).

Pattern	exkludiert (Schritt)/ verwendet	Begründung
Microservice Chassis	verwendet	Da es als Sinnvoll erachtet wird ein Framework für die Grundlagen der Implementierung eines MS zu haben.
Microservice Template	verwendet	Da es als Sinnvoll erachtet wird eine Code Vorlage für die Implementierung eines Microservice zu haben.

Tabelle 4.12.: Zusätzlich identifizierte Patterns und eine Begründung, warum die Patterns verwendet wurden.

4.3.2. Filterung der Best Practices

Das FMM stellt ebenfalls eine Exceltabelle mit Best Practices für die Implementierung zur Verfügung. Als Quellen für die Beschreibung der Best Practices wurden folgende Veröffentlichungen herangezogen: [LW22, PS22, SB21]. Der Filterungsprozess der Best Practices ist übersichtlich in Tabelle A.2 in Anhang A dargestellt.
Um mithilfe dieser Tabelle die Best Practices effizient zu filtern, hat Koch [Koc22] Daten wie die Qualitätsattribute, SQAs und System-Eigenschaften, welche durch das Anwenden der Best Practice verbessert werden, den Best Practices zugeordnet.
Um herauszufinden, welche Best Practices für die Migration der Service/Sales-Applikation in eine MSs-Applikation infrage kommen, wurde die Tabelle mehrmals analysiert und gefiltert.

Schritt	Beschreibung
1	Es wurden nur Best Practices weiter betrachtet, welche mindestens eine der drei am höchsten priorisierten QAs, oder eines ihrer SQAs, verbessern.
2	Es wird das zu migrierende System betrachtet. Best Practices, welche nicht in den Kontext des Systems passen werden exkludiert. In Schritt 1 exkludierte Best Practices können wieder betrachtet werden, sollten sie in den Kontext des Systems passen.
3	Durch firmeninterne Vorgaben werden Best Practices, welche sich auf Messaging oder Containerization beziehen, zurückgestellt und evtl. zu einem späteren Zeitpunkt betrachtet.
4	Best Practices, welche zu komplex für den PoC sind, werden zurückgestellt und evtl. zu einem späteren Zeitpunkt betrachtet.

Tabelle 4.13.: Beschreibung der verschiedenen Schritte bei der Filterung der Best Practices.

Die verschiedenen Schritte der Filterung der Best Practices sind in Tabelle 4.13 erklärt. In der ersten Filterung wurden Best Practices entfernt, welche Qualitätsattribute, Sub-Qualitätsattribute und System-Eigenschaften verbessern, die bei der Systemanalyse, Kapitel 4.1.2, eine niedrige Priorität erhalten haben. Es wurden nur Best Practices weiter betrachtet, welche mindestens eine der drei am höchsten priorisierten Qualitätsattribute oder eines ihrer SQAs verbessern. Das Ergebnis des ersten Filterungsschritt ist in Tabelle 4.14 dargestellt.
Da auch hier nach der ersten Filterung viele Best Practices übrig geblieben sind, die bei genauerer Betrachtung für die Service/Sales-Applikation irrelevant erscheinen und so manche Best Practice exkludiert wurde, welche für die Service/Sales-Applikation relevant sein könnte, wurde ebenfalls eine zweite Filterung durchgeführt, in der nicht nur die Qualitätsattribute aus Kapitel 4.1.2 ausschlaggebend sind, sondern auch der Kontext des Systems. Das Ergebnis der zweiten Filterung kann in Tabelle 4.15 betrachtet werden.
Auch für die Best Practices haben die firmeninterne Vorgaben Einfluss auf die Filterung. Das Resultat nach der Filterund der firmeninternen Vorgaben ist in Tabelle 4.16 dargestellt.
Unter den verbliebenen Best Practices wurden ebenfalls Best Practices zurückgestellt, siehe Tabelle 4.17, welche für die Implementierung des PoC als irrelevant angesehen wurden.
Es wurden noch nicht alle der zurückgestellten Best Practices in der erstellten Microservices-Architektur berücksichtigt.
Manche dieser Best Practices, wie *Access restriction*, *Account Separation*, *Configuration management* und *Guarded ingress* sind in dem vorhandenen Monolithen bereits implementiert und wurden deshalb nicht nochmal in der erstellten Microservices-Architektur behandelt. Weitere Best Practices wie *Manageable Connections*, *Manageable Standards*, *Service Independence*, *Standardization* und *Standardized deployment unit* wurden nicht explizit in der Architekturbeschreibung erwähnt, da diese implizit berücksichtigt werden.

exkludierte Best Practices	**weiter betrachtet**
API-based Communication	*Access Restriction*
Appropriate Service Relationship	*Account Separation*
Built-in Autoscaling	*Acyclic Calls*
Cloud Vendor Abstraction	*Authentication Delegation*
Coarse-Grained Microservices	*Automated Infrastructure*
Communication Indirection	*Automated Monitoring*
Cost Variability	*Automated Restarts*
Immutable Artifacts	*Autonomous Fault Handling*
Infrastructure Abstraction	*Configuration Management*
Isolated State	*Data Encryption in Transit*
Loose Coupling	*Dynamic Scheduling*
Manageable Connections	*Guarded Ingress*
Manageable Standards	*Persistent Communication*
Non-ESB Microservices	*Seamless Upgrades*
Operation Outsourcing	*Secrets Management*
Replication	*Separation by Gateways*
Resolved Endpoints	
Right Cuts	
Separate Libraries	
Separate Persistency	
Service Independence	
Service-Orientation	
Sparsity	
Standardization	
Standardized Deployment Unit	
Use Infrastructure as Code	
Versioned APIs	

Tabelle 4.14.: Best Practices, welche in dem ersten Filterungsschritt exkludiert wurden (linke Spalte) und weiter betrachtet werden (rechte Spalte).

exkludierte Best Practices	weiter betrachtet	wieder betrachtet
Automated Infrastructure	*Access Restriction*	*API-Based Communication*
	Account Separation	*Built-in Autoscaling*
	Acyclic Calls	*Cloud Vendor Abstraction*
	Authentication Delegation	*Coarse-Grained Microservices*
	Automated Monitoring	*Loose Coupling*
	Automated Restarts	*Manageable Connections*
	Autonomous Fault Handling	*Manageable Standards*
	Configuration Management	*Separate Libraries*
	Data Encryption in Transit	*Service Independence*
	Dynamic Scheduling	*Standardization*
	Guarded Ingress	*Standardized Deployment Unit*
	Persistent Communication	*Versioned APIs*
	Seamless Upgrades	
	Secrets Management	
	Separation by Gateways	

Tabelle 4.15.: Best Practices, welche in dem zweiten Filterungsschritt exkludiert wurden (linke Spalte), weiter betrachtet werden (mittlere Spalte) und wieder betrachtet werden (rechte Spalte).

Bei den Best Practices kann ebenfalls nicht garantiert werden, dass alle ausgewählten Best Practices für die Implementierung nach dieser Arbeit relevant bleiben und es kann ebenfalls nicht ausgeschlossen werden, dass die ausgefilterte Best Practices irgendwann relevant werden. Da die Services erst für die lokale Nutzung implementiert werden, werden nicht alle Best Practices, die nach der vierten Filterung übrig geblieben sind, implementiert.

4.3.3. Phase 3a Definition der Architektur

Zu der Definition der Microservices-Architektur gehört, nach Fritzsch et al. [FBH+22], die Service-Identifizierung, beschrieben in Kapitel 4.3.3 und das Formulieren der Zielarchitektur, beschrieben in Kapitel 4.3.3. In der Service-Identifizierung wird beschrieben, wie mit den ausgewählten Service-Identifikationsansätzen aus Kapitel 4.2.2, die Services identifiziert werden. In der Definition der Zielarchitektur wird erklärt, wie mit den identifizierten Services, Patterns und Best Practices die Zielarchitektur beschrieben wird.

zurückgestellt	weiter betrachtet
Acyclic Calls	*Access Restriction*
Persistent Communication	*Account Separation*
	API-Based Communication
	Authentication Delegation
	Automated Monitoring
	Automated Restarts
	Autonomous Fault Handling
	Built-in Autoscaling
	Cloud Vendor Abstraction
	Coarse-Grained Microservices
	Configuration Management
	Data encryption in Transit
	Dynamic Scheduling
	Guarded Ingress
	Loose Coupling
	Manageable Connections
	Manageable Standards
	Seamless Upgrades
	Secrets Management
	Separate Libraries
	Separation by Gateways
	Service Independence
	Standardization
	Standardized Deployment Unit
	Versioned APIs

Tabelle 4.16.: Die Best Practices, welche in dem dritten Filterungsschritt zurückgestellt wurden (linke Spalte) und weiter betrachtet werden (rechte Spalte).

Service-Identifizierung

Wie bei Li et al. [LML20] wird als erstes der Monolith analysiert und die Services identifiziert. Für die Serviceidentifikation wurde die Publikation von Krause et al. [KZH+20] verwendet. Der erste Schritt bei ihrer Serviceidentifikation ist, die Bounded Contexts der Applikation zu identifizieren. Es folgt bei ihnen eine statische Codeanalyse unter Verwendung des Tools Structure 101[10]. Sie beschreiben, dass die Klassen in die analysierten Bounded Contexts auf-

[10] https://structure101.com/

zurückgestellt	**weiter betrachtet**
Authentication Delegation	*Access Restriction*
Automated Monitoring	*Account Separation*
Automated Restarts	*API-Based Communication*
Autonomous Fault Handling	*Coarse-Grained Microservices*
Built-in Autoscaling	*Configuration Management*
Cloud Vendor Abstraction	*Guarded Ingress*
Data Encryption in Transit	*Loose Coupling*
Dynamic Scheduling	*Manageable Connections*
Seamless Upgrades	*Manageable Standards*
Secrets Management	*Separate Libraries*
Separation by Gateways	*Service Independence*
	Standardization
	Standardized Deployment Unit
	Versioned APIs

Tabelle 4.17.: Die Best Practices, welche in dem vierten Filterungsschritt zurückgestellt wurden (linke Spalte) und weiter betrachtet werden (rechte Spalte).

geteilt werden. Mit einer dynamischen Analyse unter Verwendung von Exploreviz[11] werden diese Bounded Contexts, nach Krause et al., nochmals verfeinert. Die Datenbank wird bei ihnen mithilfe des Tools DBeaver[12] ebenfalls in die Bounded Contexts aufgeteilt. Die Autoren beschreiben, falls eine Datenbanktabelle von mehreren Bounded Contexts verwendet werden sollte, so wird diese Tabelle für die verschiedenen Bounded Contexts separiert [KZH+20].
Für die Serviceidentifikation bei der L-mobile Service/Sales-Applikation wurden ebenfalls erst die Bounded Contexts analysiert. Für die grafische Darstellung des Codes wurde ebenfalls Structure 101 verwendet. Da der monolithische Code bereits sehr ordentlich strukturiert und in zusammengehörige Module aufgeteilt ist, lief die Analyse der Bounded Contexts schnell und ohne Probleme. Viele der bereits existierenden Module waren bereits eigene Bounded Contexts und manche Module mussten zusammengelegt werden, um einen Bounded Context zu ergeben. Es gab zwei Module, die mehrere Bounded Contexts in sich vereinen.
Eine dynamische Analyse mit Exploreviz konnte nicht durchgeführt werden, da die verfügbare allgemeine Version während der Ausführung beim Anlegen eines neuen Software-Landscape für die Applikation den Fehler „Something went wrong“ ausgibt. Eine weitere Version von Exploreviz war nur für Java Anwendungen verfügbar.
Ein ähnliches Tool für die Visualisierung dynamischer Analysen für .Net Anwendungen wurde nicht gefunden. Da der Code jedoch bereits in die Bounded Contexts aufgeteilt ist und die Expertenbefragung bei L-mobile ebenfalls ergeben hat, dass die Bounded Contexts gut arrangiert

[11] https://www.explorviz.dev/
[12] https://dbeaver.io/

sind, wurde entschieden, auf die dynamische Analyse zu verzichten.
Die Datenbank wurde ebenfalls mit DBeaver analysiert. Bei der Aufteilung der Datenbank muss pro Service vorgegangen werden, da die Tabellen eine hohe Referenzierung untereinander aufweisen und die Module Daten eines anderen Moduls benötigen. Hierfür bietet sich das Shared Database Pattern [Ric23p] an.
Die resultierenden Services sind in Abbildung 4.6 dargestellt. Es gibt fünf übergeordnete Kontexte:

- Main enthält 16 Basisfunktionalitäten und wurde daher in 16 Bounded Contexts unterteilt:
  - Offline, enthält die Funktionalitäten für den Offlinebetrieb
  - Replication
  - AccountManagement
  - Task
  - Flow
  - SmtpDropbox
  - Multitenant
  - VideoCall
  - AttributeForms
  - Documentation
  - Documents
  - Login
  - Posting
  - PushNotification
  - Company
  - Notes
- Crm, der Crm Kontext besteht aus 14 weiteren Kontexten:
  - Campains
  - MarketInsights
  - Project
  - Order
  - Service, welcher wiederum in 6 verschiedene Kontexte aufgeteilt ist:
    - Checklists
    - ReplenishmentOrder
    - ServiceOrder
    - ServiceCase
    - ServiceContract
    - Installation
  - VisitReport
  - Configurator
  - DynamicForms
  - PerDiem
  - Article

  - ErpExtension
  - InforExtension
  - PdfGeneration
  - Web
- Sms, das Service Management System enthält drei weitere Kontexte:
  - Einsatzplanung
  - TimeManagement
  - ZeitIntervalle
- Integration enthält Basisklassen, welche für die Anbindung an ein ERP-System wichtig sind.
- Customer enthält Plugins für die Kundenanpassungen.

Die Kontexte PdfGeneration, Zeitintervalle und Login wurden von den Product Managern vorgeschlagen und sind in der ursprünglichen Codestruktur nicht enthalten. Diese Kontexte besitzen eine eindeutige Funktion (SRP) und stehen nicht in Abhängigkeit mit den anderen Kontexten, weshalb diese sich gut in eigene Services auslagern lassen.

Definition der Zielarchitektur

Anders als bei Li et al. [LML20] besteht in dieser Arbeit die vorbereitenden Aufgaben, neben der Filterung der Patterns und Best Practices, darin eine Architektur zu erstellen. Die Microservices-Architektur der Service/Sales-Applikation Version 9.1 von L-mobile basiert auf den gleichen Basis-Technologien wie der Monolith. Die schrittweise Migration den Monolithen in Microservices findet unter Verwendung des *Strangler Fig* Patterns statt [Fow04, LML20, Ric23q, YM20]. Dies erlaubt es, nach Yoder et al. [YM20], neue Funktionen als Microservices der Applikation hinzuzufügen und bestehende Funktionen aus dem Monolithen als Microservices zu extrahieren, während der restliche Monolith parallel zu den Microservices betrieben werden kann. Für den Rahmen dieser Arbeit kann dadurch ein einzelnes Repository verwendet werden. Wie in Kapitel 4.3.3 bereits beschrieben, kommt für die Datenbankmigration das *Shared Database* Pattern zum Einsatz [Ric23p]. Tabellen, die von mehreren Microservices verwendet werden, aufzuteilen und in andere Microservice spezifische Tabellen zu integrieren, wie in Krause et al. beschrieben [KZH+20], kommt für die Service/Sales-Applikation nicht infrage, da dies für Bestandskunden, die zu der Microservices Lösung migrieren wollen, eine zu komplexe Datenmigration nach sich ziehen würde. Daher wurde mit Stakeholdern bei L-mobile abgesprochen, Datenbanktabellen entweder zu duplizieren oder das *Shared Database* Pattern [Ric23p] einzusetzen. Diese Entscheidung wird für jeden Service separat getroffen.
Die Kommunikation zwischen den Microservices untereinander und zwischen dem Monolithen und den Microservices findet im Rahmen dieser Arbeit implementierten Umfang per *REST API* statt [Ric23k]. Zu einem späteren Zeitpunkt, wenn bereits mehrere Microservices extrahiert wurden, kann eine Diskussion über die Verwendung von *Messaging* [Ric23g] für die Kommunikation geführt werden.

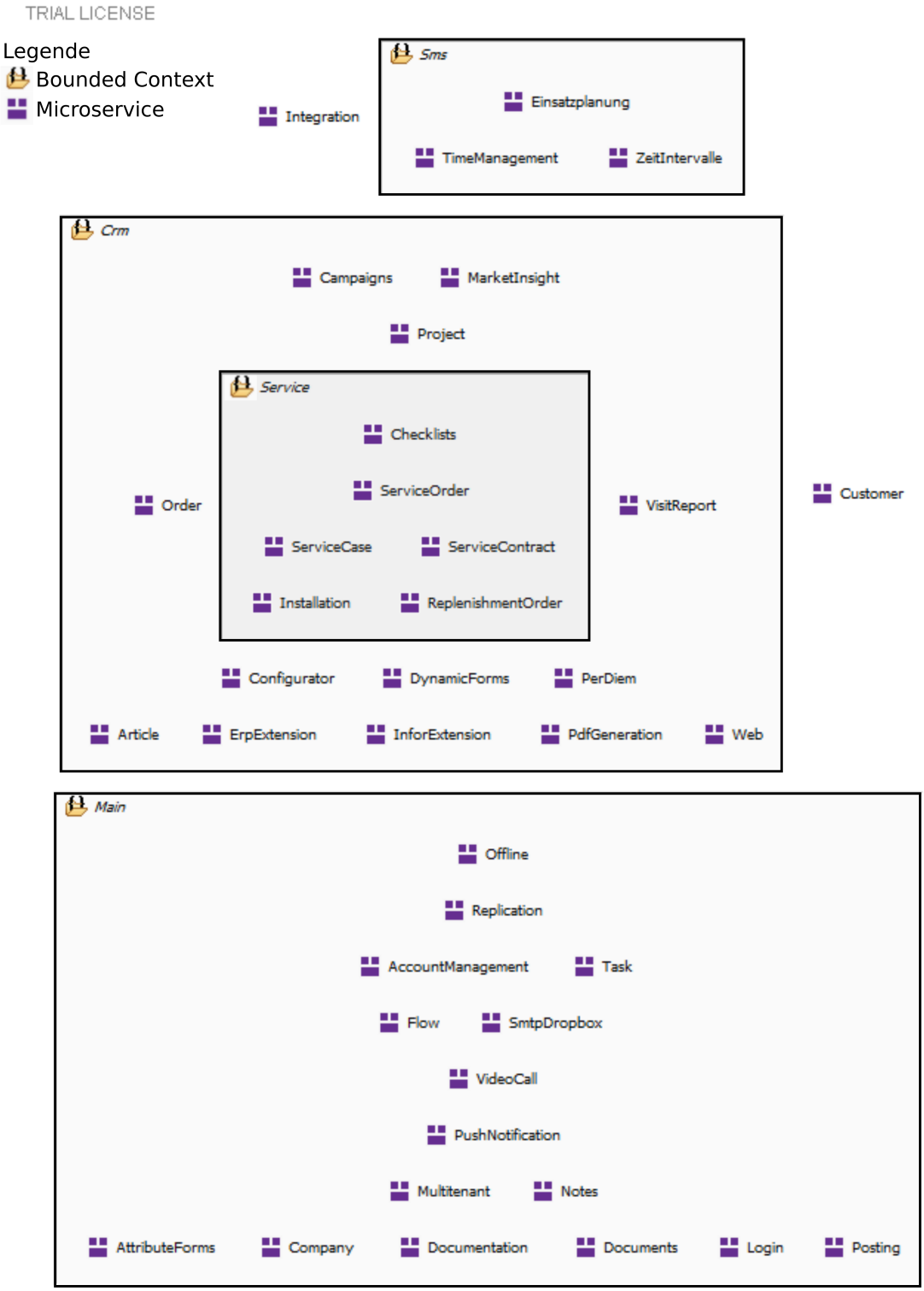


Abbildung 4.6.: Die identifizierten Services in ihren Bounded Contexts. Bild exportiert aus Structure 101.

Da bei der Migration nach dem *Strangler Fig* Pattern [Fow04, LML20, Ric23q, YM20] vorgegangen wird, kommunizieren die Endgeräte mit dem verbliebenen Monolithen. Dieser Monolith enthält für die bereits extrahierten Microservices Schnittstellen, welche die jeweilige *REST API* des Microservice aufruft [YM20]. So wird nach dem *Strangler* Pattern, wie in Yoder et al. beschrieben [YM20], das Modul an der Stelle des extrahierten Service umstrukturiert, so dass es die *REST API* des extrahierten Services aufruft. Bei einer vollständigen "Strangulierung„ des Monolithen sollte der Monolith durch ein *API Gateway* [Ric23b] ersetzte werden. Dieses *API Gateway* agiert als Einstiegspunkt für die Endgeräte und leitet die Anfragen an die entsprechenden Services weiter. Damit das *API Gateway* die korrekten Services ansprechen kann, wird eine *Service Registry* [Ric23n], in der sich die aktiven Microservices registrieren können, implementiert.
Einen Überblick über die Microservices-Architektur, wie sie in dieser Arbeit implementiert wird, ist in Abbildung 4.7 zu sehen. Abbildung 4.8 gibt einen architektonischen Überblick über die aktuelle implementierte Service/Sales-Applikation mit extrahiertem Microservice. In Abbildung 4.9 ist ein Überblick über die Architektur einer zukünftigen Version der Service/Sales-Applikation angegeben, in welcher der Monolith vollständig in Microservices aufgegangen ist. Die konkreten Kommunikationsmuster der in dieser Arbeit implementierten MSA ist in Abbildung 4.10 zu sehen. Zur Übersicht werden Module innerhalb des Monolithen, welche keine Kommunikation zu einem extrahierten Microservice aufweisen, nicht gezeigt. Die Kommunikationsmodule, welche die Aufrufe an die Microservices weiterleiten, wurden in *Library*, *Crm.Web* und *Main* implementiert.
Die mögliche Verwendung des *Messaging* Patterns [Ric23g] wird zu einem späteren Zeitpunkt entschieden, eine mögliche Implementierung ist jedoch in Abbildung 4.9 bereits angegeben. Da von den Stakeholdern gewünscht wurde, dass die gesamte Applikation eine Log Datei verwendet, anstatt dass jeder Microservice seinen eigenen Log generiert, wird zu einem späteren Zeitpunkt das *Log Aggregation* Pattern [Ric23f] implementiert. Damit sichergestellt werden kann, dass nur autorisierte Nutzer eine Anfrage an einen Microservice senden können, sollten *Access Tokens* [Ric23a] verwendet werden. Außerdem sollte darüber nachgedacht werden, *Health Checks* und *Monitoring* [Ric23e, VLL+20] einzubauen. Es sollten mehrere Instanzen der Microservices auf einem Host [Ric23j] betrieben werden.
Damit keine Abhängigkeiten von anderen Services oder dem Monolithen entstehen, verwaltet jeder Microservice seine Bibliotheken selbst. *Circuit Breaker* [Ric23c] und *Load Balancer* [Pra22, VLC19, VLL+20] werden ebenfalls implementiert, um zu verhindern, dass Ausfälle durch die Microservices kaskadieren. Damit sichergestellt werden kann, dass alle Microservices die korrekten Grundlagen jedes Microservice implementieren, wird eine *Microservice Chassis* [Ric23h] und ein *Microservice Template* [Ric23o] generiert. Um Einstellungen, welche mehrere Microservices betreffen können oder die nicht hard-gecoded sein sollten, wird eine externe Konfigurationsdatei [Ric23d] angelegt, von der aus die Microservices ihre Einstellungen einlesen können.
Die Architektur der im Rahmen dieser Arbeit zu implementierenden Microservices wird nicht so umfangreich wie die oben beschriebene Architektur ausfallen. Mittels des *Strangler Fig* Patterns [Fow04, LML20, Ric23q, YM20] wird der Service *Crm.PdfGeneration* extrahiert. An die Stelle, in der dieses Modul im Monolithen war, wird das Modul durch ein Modul ergänzt,

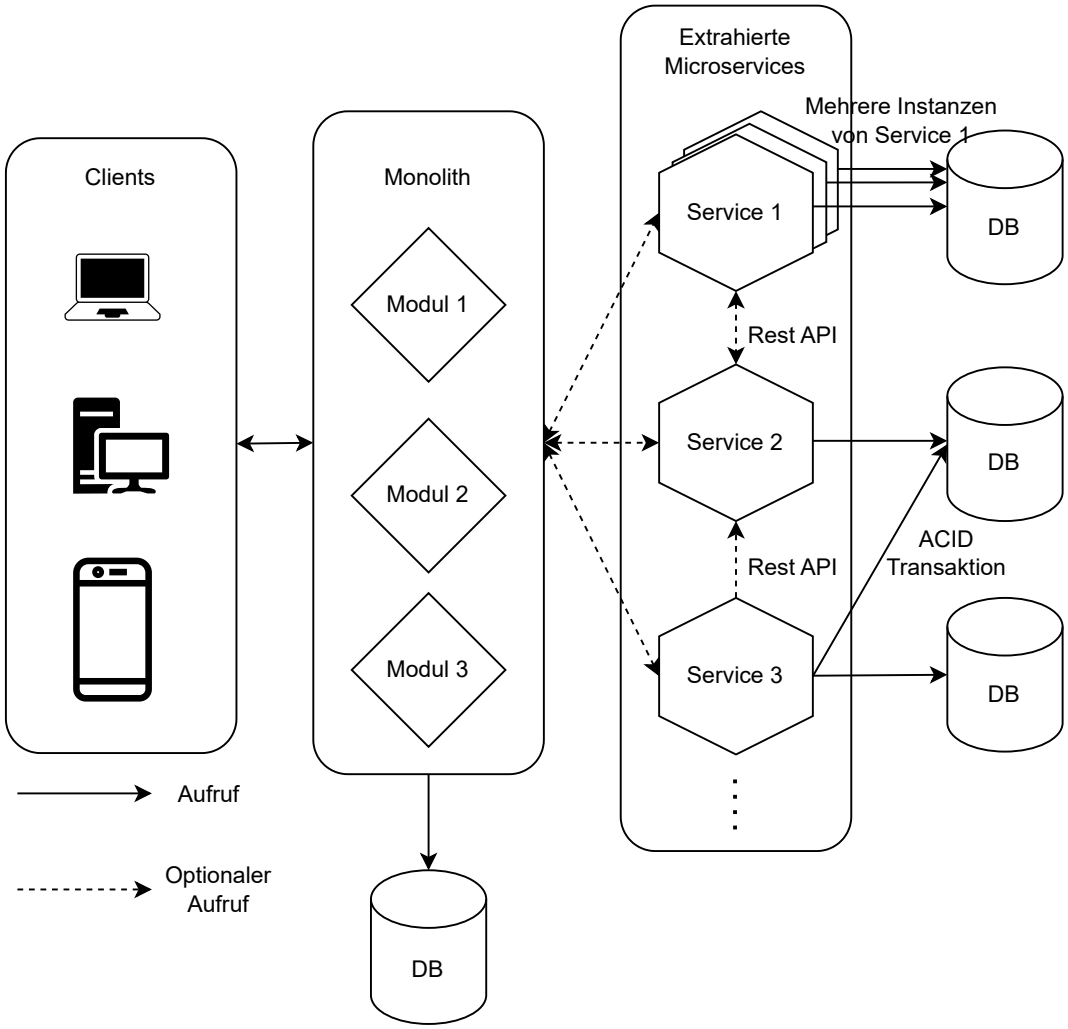


Abbildung 4.7.: Überblick über die Microservices-Architektur, wie sie in dieser Arbeit für frühe Versionen der Migration vorgeschlagen wird.

welches die Anfrage an den Service weiterleitet [YM20]. Die Kommunikation zwischen dem Monolithen und dem Microservice findet per *REST API* [Ric23k] statt. Im Rahmen dieser Arbeit wird der Service für die lokale Nutzung mit fester IP-Adresse und Port implementiert. Ein *Circuit Breaker* [Ric23c] wird prototypisch für den Service erstellt. Es wurde entschieden *Container* [Ric23m], *Messaging* [Ric23g] und Aussagen über Skalierung nicht durchzuführen. Eine Implementierung einer *Service Registry* [Ric23n] mit *serverseitiger Discovery* [Ric23l] wird in dieser Arbeit nicht durchgeführt. Eine *Microservices Chassis* [Ric23h] und ein *Microservices Template* [Ric23o] wird nicht erstellt. Die Patterns *Access Token* [Ric23a] und *API Gateway* [Ric23b] werden in dieser Arbeit nicht implementiert. Eine Service-Datenbank muss für den Service nicht extrahiert werden, da dieses Modul im Monolithen keine Zugriffe auf die Datenbank tätigt.

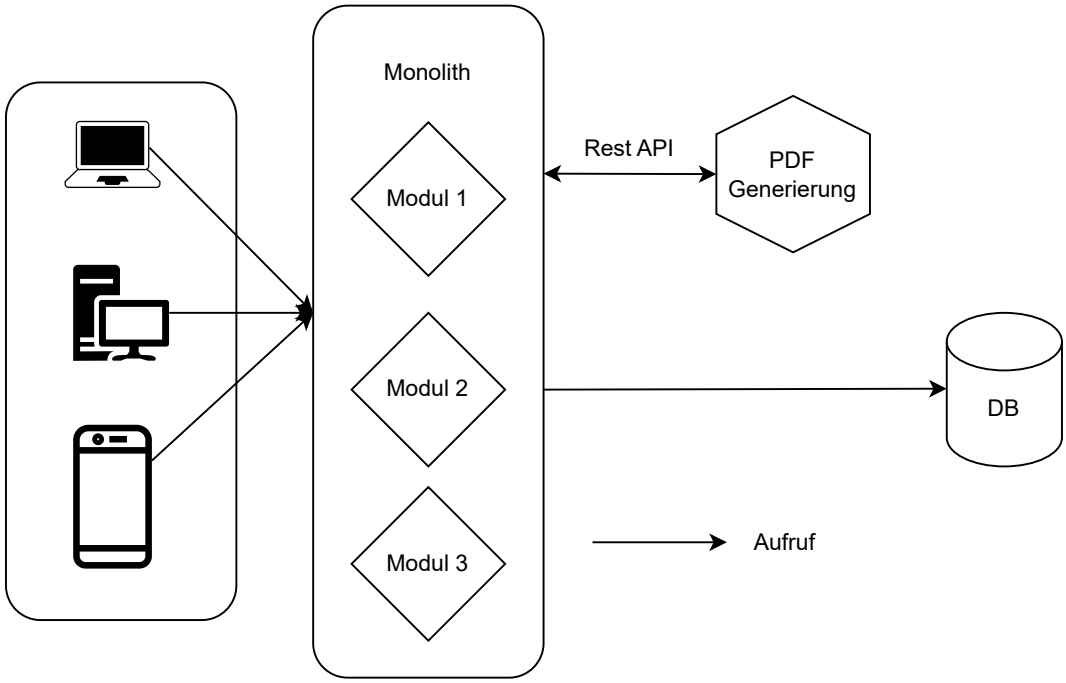


Abbildung 4.8.: Überblick über die aktuell implementierte Microservices-Architektur.

4.3.4. Phase 3b Service-Implementierung

Im zweiten Teil der Phase 3 wird, wie in Fritzsch et al. [FBH+22] und Li et al. [LML20] beschrieben, zuerst eine Service-Priorisierung durchgeführt. Anschließend wird der am höchsten priorisierte Service, wie die Autoren vorsehen, implementiert.

Service-Priorisierung

Für die Implementierung des PoC wurden Meetings mit dem Head of Development und Product Manager abgehalten, in denen die identifizierten Services priorisiert wurden, um einen Service zu finden, der für den PoC implementiert werden soll.
Die höchste Priorität hat dabei der PDF-Generierungs-Service bekommen, da dies ein kleiner Service ist, der nicht viel mit anderen Modulen kommuniziert und nach den Stakeholdern verursacht diese Komponente im Monolithen Probleme.
Yoder et al. [YM20] empfehlen, mit kleinen Schritten bei der Migration zu beginnen, deshalb eignet sich ein kleiner Service für den Start der Extraktion. Li et al. [LML20] haben viele Kriterien für die Priorisierung der Services vor der Implementierung. Ein Service sollte nach ihren Kriterien eine hohe Priorität erhalten, wenn er u. a. Performance Probleme aufweist und separiert von dem Monolithen ist. Beide Kriterien treffen auf den PDF-Generierungs-Service zu.

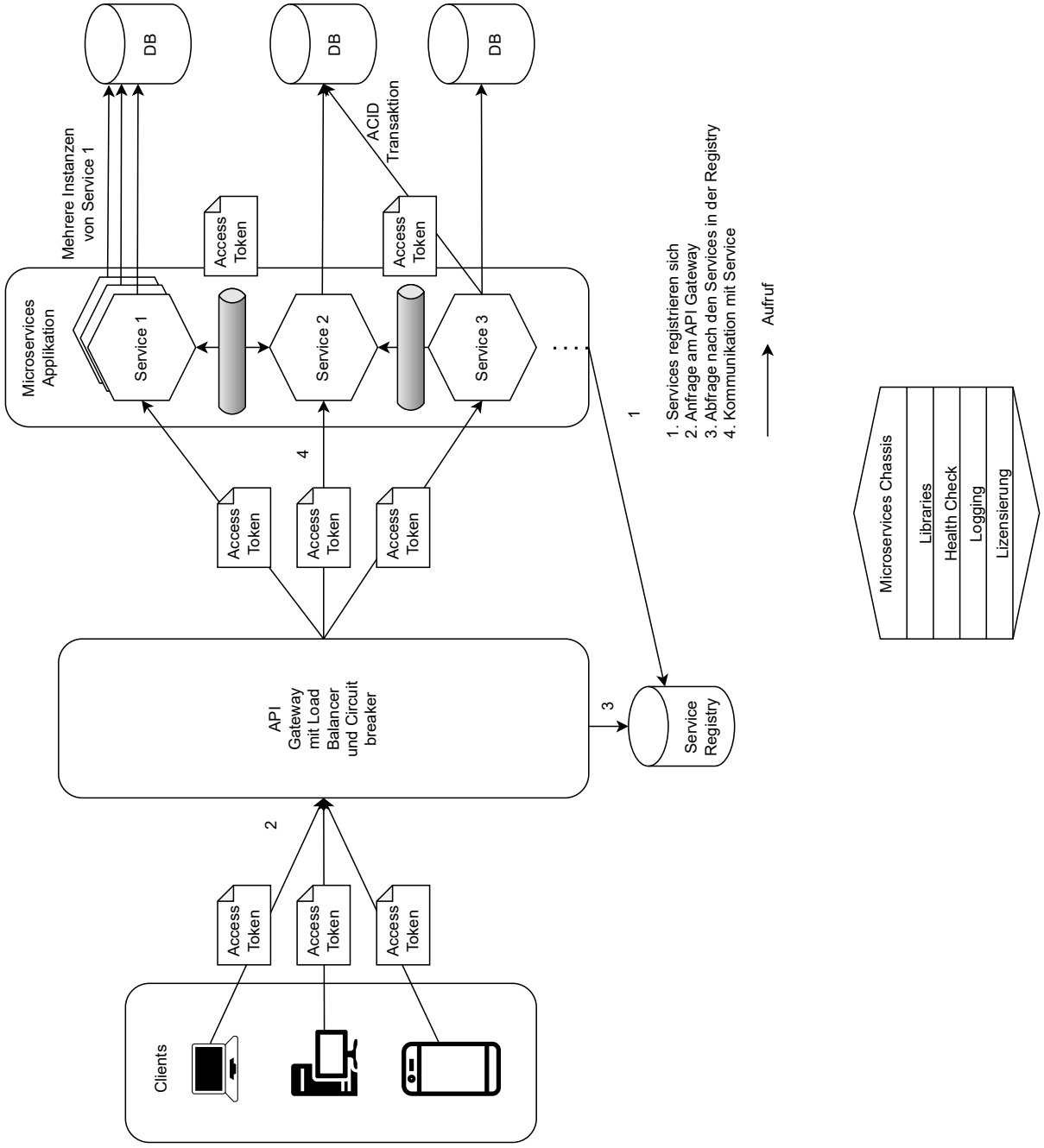


Abbildung 4.9.: Überblick über die angestrebte Microservices-Architektur. Service Registry Schritte aus [LML20].

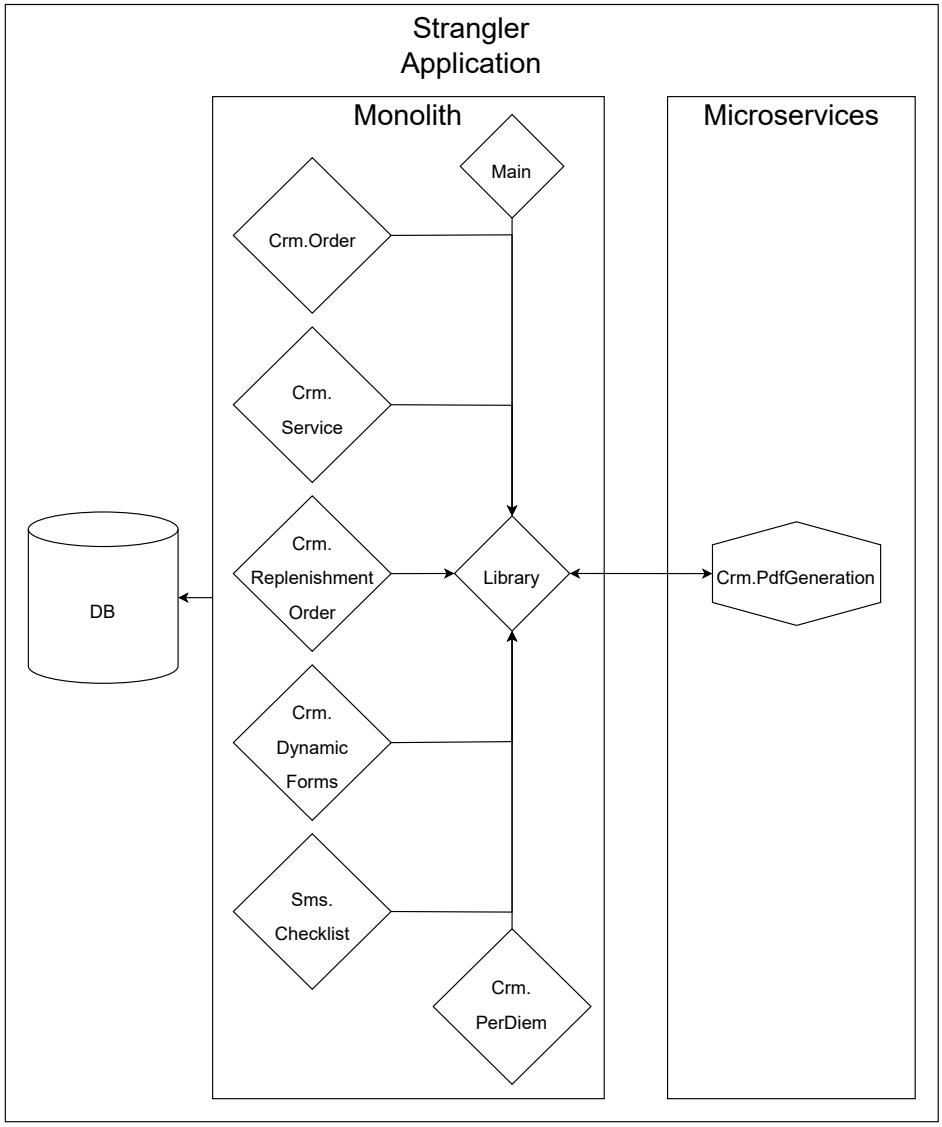


Abbildung 4.10.: Überblick über die in dieser Arbeit implementierte MSA mit ihren Kommunikationswegen.

Die Performance Probleme wurden von den Stakeholdern direkt angesprochen, und da er wenig Kommunikation und Abhängigkeiten zu anderen Modulen und Klassen des Monolithen, hat kann er als separiert vom Monolithen betrachtet werden. Auch Taibi et al. [TLP17] geben an, dass eine Priorisierung u. a. durch den Kunden geschieht, in diesem Fall die Stakeholder. Außerdem wird bei ihnen u. a. durch die Anzahl an Bugs oder von Abhängigkeiten im Monolithen die Priorität erhöht. Sie geben an, dass eine höhere Anzahl an Bugs oder eine niedrige Anzahl an Abhängigkeiten für eine höhere Priorität sorgt. Die niedrige Anzahl an Kommunikation

zu dem Monolithen lässt darauf schließen, dass der PDF-Generierungs-Service eine niedrige Abhängigkeit vom Monolithen hat, was ihn für die Implementierung hoch priorisiert.

Services-Implementierung

In diesem Abschnitt wird die Implementierung des Crm.PdfGeneration Microservice beschrieben.

Crm.PdfGeneration
Als erstes wurde in der Solution des Crm Repositories ein neues „ASP.NET Core Web API" Projekt angelegt. Anschließend wurden die Klassen, aus denen der Microservice bestehen wird, in das Projekt kopiert. In diesem Fall wurden die Klassen *IPdfService.cs* und *PdfService.cs* aus *Crm.Library* kopiert. Da diese Klassen Abhängigkeiten aus dem *Crm.Library* und diese Abhängigkeiten weitere Abhängigkeiten hatten, wurden die Abhängigkeiten ebenfalls in das *Crm.PdfGeneration* Projekt kopiert, da jeder Microservice seine Bibliotheken und Abhängigkeiten selbst verwalten soll. Anschließend wurden externe Abhängigkeiten über den NuGet Package Manager[13] installiert. Um das Projekt korrekt zu initialisieren, wurde neben der *Program.cs* auch eine *Startup.cs* Klasse generiert. In der *Startup.cs* Klasse werden u. a. die Konfigurationsdateien eingelesen, der PdfService registriert sowie HttpClient und der API Controller hinzugefügt, zu sehen in Listing B.1.
Um zu garantieren, dass alle benötigten Werte mit der Kommunikation geliefert werden, werde DTO Klassen erstellt, welche die benötigten Informationen enthalten. Die verschiedenen DTOs für den PdfGeneration-Microservice werden im Klassendiagrammen in Abbildung 4.11 gezeigt. Das *PdfDTO* wird verwendet, um die generierte PDF als `byte` Array zurück an den Aufrufer zu übergeben. Die restlichen DTOs enthalten Parameter, welche für die verschiedenen Methodenaufrufe von *PdfService.cs* von Bedeutung sind.
Als nächstes wurde für die Kommunikation über die REST API eine *PdfController.cs* Klasse erzeugt. In dieser Klasse wurde eine Get-Methode für jede öffentliche Methode der *PdfService.cs* Klasse, welche in der früheren Monolith Klasse von anderen Klassen aufgerufen wurde, generiert. Jede dieser Get-Methoden bekommt als Parameter ein spezielles DTO übergeben, welches über den Anfragekörper des REST API Aufrufes übergeben wird. Ursprünglich wurde der API-Aufruf über Parameter geregelt, `/Html2Pdf/{html}/{headerMargin}/{footerMargin}`. Da dieser Aufruf jedoch für die generierten HTML Strings zu lang wurde und in einem *414 URI Too Long* Error endete, wurde entschieden, die Parameter per Anfragekörper zu übergeben. Anschließend ruft die Get-Methode die jeweilige Methode des PdfService mit den Parametern des DTO auf. In Listing B.2 wird eine dieser Get-Methoden gezeigt. Sie bekommt *SendPdfDTO* mit dem String der HTML-Seite, welche als PDF konvertiert werden soll, und Kopf- und Fuß-Rand übergeben, Zeile 18. Diese Werte leitet sie an den PdfService weiter, Zeile 20, und liefert ein *PdfDTO* mit der vom PdfService erzeugten PDF als `byte`

[13] https://www.nuget.org/

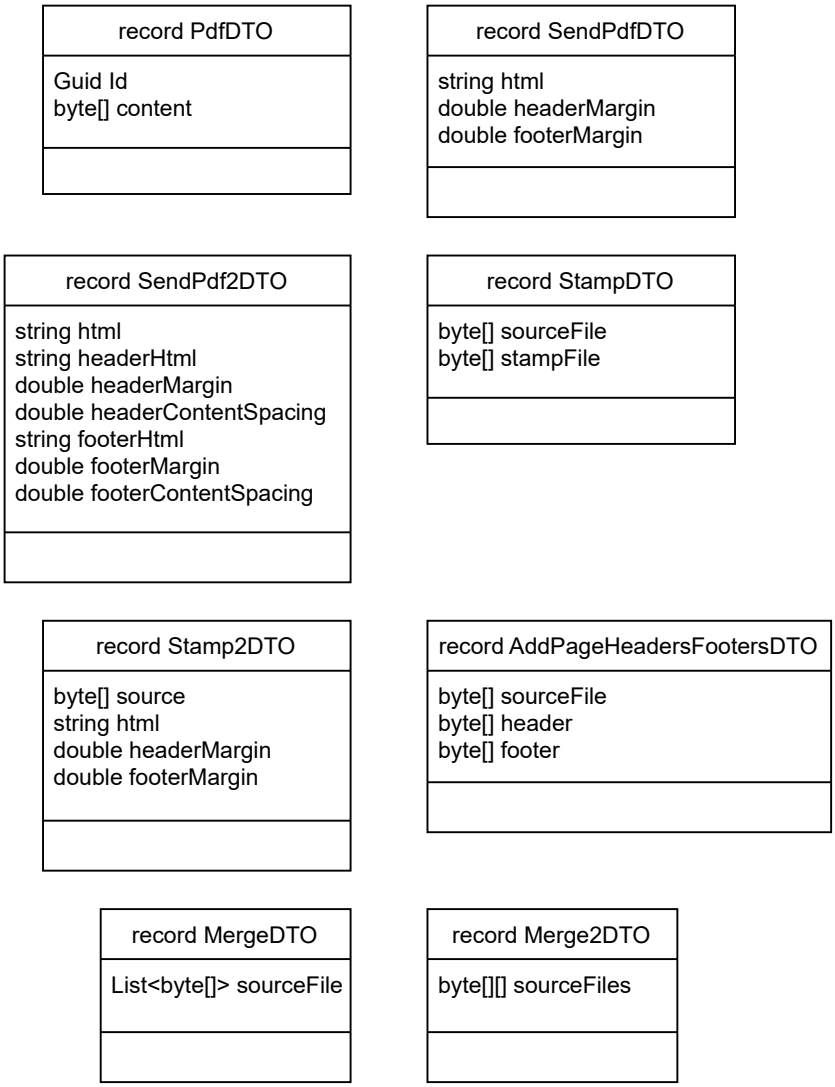


Abbildung 4.11.: Überblick über die verschiedenen DTOs.

Array zurück, Zeilen 25f. Diese Methode wird durch den Aufruf der Uniform Resource Identifier (URI) `https://localhost:7114/api/Pdf/Html2Pdf/dto` aufgerufen. Dabei bezieht sich `/api/Pdf/` auf den Pfad des Controllers, Zeile 9. `Html2Pdf/dto`, Zeile 17, bezieht sich auf die Get-Methode welche aufgerufen werden soll, wobei `dto` auf das DTO im Aufrufkörper bezieht. Damit sichergestellt werden kann, dass die Klassen ihre Abhängigkeiten korrekt übergeben bekommen, wird in dem Crm Repository der inversion of control container AutoFac[14] verwendet. Damit die korrekten Instanzen von Objekten an die Klassen per dependency injection übergeben werden können, wurde eine *PdfModule.cs* Klasse erstellt. In dieser Klasse werden Objekte registriert, welche über dependency injection an Klassen übergeben werden. In Lis-

[14] https://autofac.org/

ting B.3 ist die *PdfModule.cs* Klasse, zur besseren Übersicht stark vereinfacht, dargestellt. Es ist dargestellt, wie eine Konfigurationsinstanz registriert wird, Zeilen 40 - 51. In Zeile 52 wird die aktuelle Seite registriert. Der CefToPdfConverter wird in den Zeilen 53 - 61 registriert.
Der Pdfservice, Listing B.4, nimmt die Werte des Controllers entgegen, Zeile 18. Die Generierung findet durch den CefToPdfConverter in Zeilen 25 - 29 statt. Zum Schluss wird die PDF als `byte` Array zurückgegeben, Zeile 30, bzw. es wird im Falle einer Timeout Exception, Der Wert `null` zurückgegeben, Zeile 35.
Um den PdfGeneration-Microservice aufzurufen, wurde die Kommunikation in dem *Crm.Library* Projekt implementiert. Da der PdfGeneration-Microservice auf einer anderen IP-Adresse mit einem anderen Port betrieben wird, wurde eine *PdfServiceClient.cs* Klasse, vereinfacht dargestellt in Listing B.5, in *Crm.Library* implementiert, welche den Aufruf an eine externe REST API regelt. Das *SendPdfDTO* wird aus übergebenen Parametern generiert, Zeile 23. Das DTO wird im Anschluss als JSON-Objekt serialisiert, Zeile 24. Anschließend wird die URI für den API Aufruf generiert, Zeile 25. Dafür wird die BaseAddress, in dem Fall einer lokalen Ausführung mit definiertem Port `https://localhost:7114`, aus der Konfigurationsdatei ausgelesen und mit dem Pfad der API Get-Methode verknüpft. Daraufhin wird die Anfrage mit samt Aufrufkörper generiert, Zeilen 26 - 31, und gesendet, Zeile 32. Zum Schluss wird das zurückgelieferte *PdfDTO* extrahiert und zurückgegeben, Zeilen 33 und 34.
Diese *PdfServiceClient.cs* Klasse musste in der *Startup.cs* Klasse des Startup-Projekts des Monolithen initialisiert werden, Zeile 99-102 in Listing B.7.
Damit Anfragen, welche an die ursprüngliche *PdfService.cs* Klasse des *Crm.Library* Projekts gesendet werden, an den PdfGeneration-Microservice weitergeleitet werden, wurden die Methodenkörper der ursprünglichen Klasse so angepasst, dass sie die entsprechende Methode der *PdfServiceClient.cs* Klasse aufrufen, welche die Anfrage an die entsprechende REST API des PdfGeneration-Microservice weiterleitet. In Listing B.6 ist dargestellt, wie sich der ursprüngliche Methodenkörper ändert. Es wird nun die `GetPdfAsync` Methode des pdfServiceClient aufgerufen und die erhaltenen Parameter weiter gegeben, Zeile 14. Da sich bei der Implementierung für einen asynchronen API Aufruf entschieden wurde, Listing B.5 Zeile 32, muss auf die Antwort gewartet werden, da der Prozess sonst ohne Antwort weiter läuft und die Antwort des Microservices ins Leere läuft, Listing B.6 Zeile 14. Bevor die Antwort zurückgeliefert werden kann, muss die PDF in Form eines `byte` Array extrahiert werden, Zeile 15.
Klassen und Module des Monolithen, welche eine Kommunikation zu der *PdfService.cs* Klasse des Monolithen haben, müssen ihre Implementierung nicht ändern und haben keine Ahnung davon, dass die Funktionalität in einen Microservice ausgelagert wurde.
Da die Kommunikation zum PdfGeneration-Microservice und wieder zurück zum Monolithen über DTOs geregelt wird, wurden auch in dem *Crm.Library* Projekt die entsprechenden DTOs angelegt. Diese DTOs sind identisch mit den DTOs aus dem Microservice, welche in Abbildung 4.11 gezeigt werden. Ein prototypischer *Circuit Breaker* wurde für den REST API Aufruf des PdfGeneration-Microservice in der *Startup.cs* Klasse des Monolithen implementiert, Listing B.7. Insgesamt steuert der Monolith den Microservice 5 Mal bei einer fehlgeschlagenen Anfrage erneut an, Zeile 104. Jedes Mal mit einer anderen Wartezeit bis zum nächsten Versuch, Zeilen 105 und 106. Nach 3 Versuchen öffnet sich der *Circuit Breaker* und wartet 15 Sekunden

bis er sich wieder schließt, Zeile 115-128.
Eine beispielhafte Kommunikation zwischen Monolith und Microservice für die PDF-Generierung, ist in Algorithmus 4.1 und Abbildung 4.12 dargestellt.

Algorithmus 4.1 Algorithmus, der die Kommunikation zwischen Monolith und Microservice beschreibt.

User will eine PDF generieren und klickt auf die Schaltfläche in der Benutzeroberfläche.
Klasse im Monolith, für diese Oberfläche generiert einen HTML-String für die entsprechende Seite, welche in eine PDF konvertiert werden soll.

Klasse ruft die Methode der PdfService Klasse im Monolith für die Generierung einer PDF auf.
| PdfService leitet den Aufruf an die PdfServiceClient Klasse weiter und wartet auf eine Antwort.
| | PdfServiceClient generiert ein DTO aus den übergebenen Werten und ruft die API des Microservice auf.
| | | Controller des Microservice nimmt den API Aufruf entgegen, entpackt das übergebene DTO und ruft mit den DTO Werten die entsprechende Methode der PdfService Klasse im Microservice auf.
| | | | Die PdfService Klasse Microservice ruft mit den übergebenen Werten den CefToPdfConverter auf, welcher mit den übergebenen Werten eine PDF generiert.
| | | | **return** PDF an PdfController
| | | **return** PDF an Monolith (PdfServiceClient)
| | **return** PDF an PdfService (im Monolith)
| **return** PDF an aufrufende Klasse
End

Schlussendlich wurde von der Firma L-mobile entschieden, für den PoC nur einen Microservice zu extrahieren. Diese Entscheidung wurde getroffen, damit im Zeitraum dieser Arbeit mehr Zeit in die Ausbesserung des einen Microservice investiert werden kann. Dies bedeutet die Implementierung so gut wie es geht, auf den Einsatz vorzubereiten. Mehrere Code-Reviews und Tests für den Anwendungsfall des Microservice durchzuführen. Ein Review durchzuführen, in dem das Vorgehen/Ergebnis des PoC an sich bewertet wurde, ob es sich für die weitere Migration der Service/Sales-Applikation eignet. Eventuell den Microservice so zu gestalten, dass er als Vorbild für zukünftige Microservices dienen kann.
Für die letzten Schritte gab es eine Abweichung der Reihenfolge wie sie von Li et al. [LML20] vorgesehen ist. In dieser Arbeit wurden die Services erst implementiert, anschließend wurde das Modul aus dem Monolithen entfernt, bzw. durch die Kommunikation zu dem Microservice ergänzt und dadurch der Service integriert. Zum Schluss wurde der Service getestet.

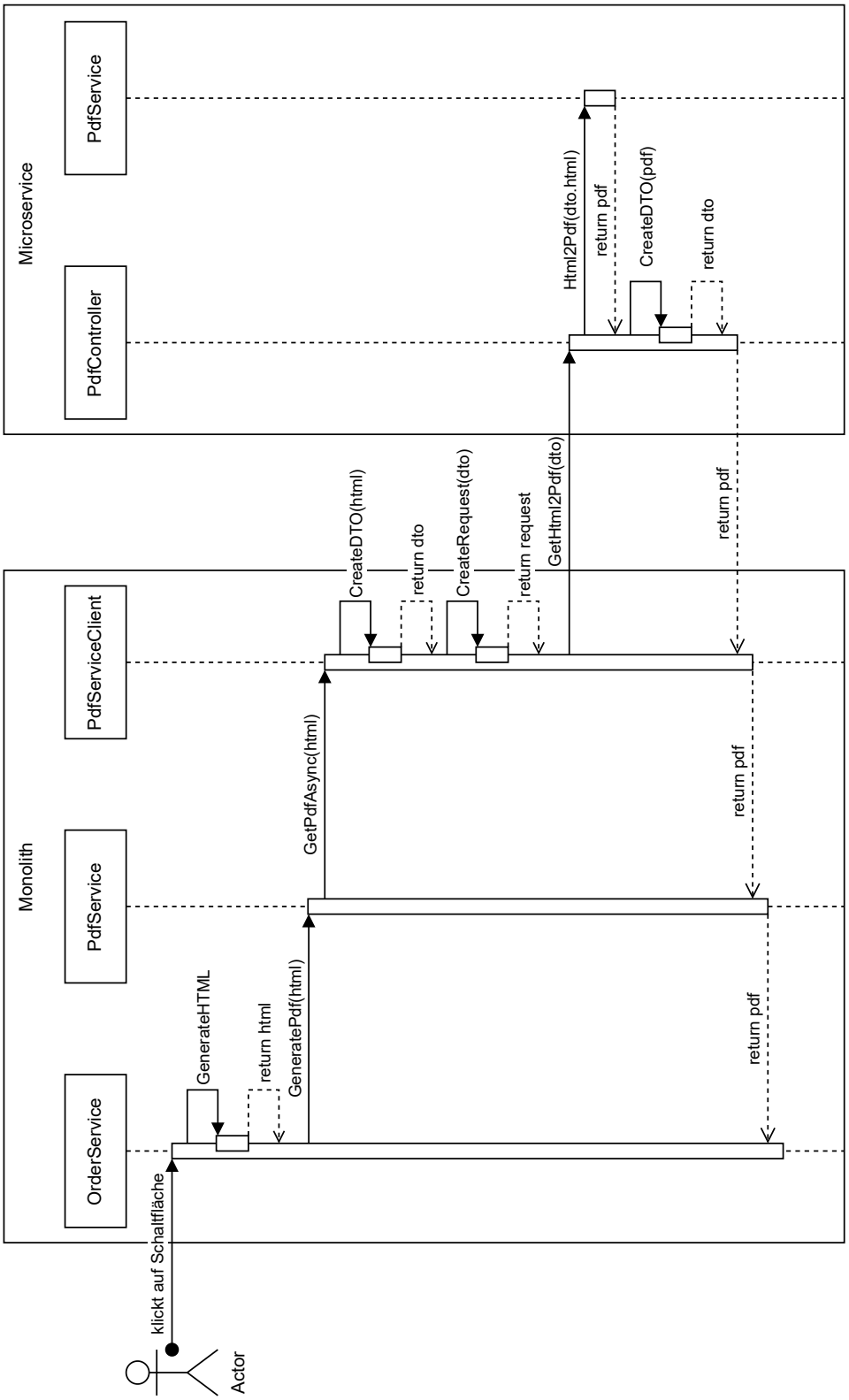


Abbildung 4.12.: Überblick über die Kommunikation zwischen Monolith und Microservice.

5. Reviews und automatisierte Tests

Es wurde ein Architektur-Review durchgeführt, um sicher zu stellen, dass die erstellte Architektur verständlich ist und keine Mängel aufweist. Das Architektur-Review kann in Kapitel 5.1 nachgelesen werden. Um sicher zu stellen, dass nach der Extraktion eines Microservices die Codequalität erhalten bleibt und sich die Funktionalität der Applikation nicht verändert hat, wurden Code-Reviews und automatisierte Tests durchgeführt. Die Kapitel 5.2 und 5.3 gehen näher darauf ein, wie diese Code-Reviews und automatisierte Tests durchgeführt wurden und welche Ergebnisse erzielt wurden. Außerdem soll in einem Review herausgefunden werden, ob die Arbeitsweise und die Kommunikation des Microservices und des Monolithen, so wie der PoC für den produktiven Betrieb und die weitere Migration der Service/Sales-Applikation in Microservices dienlich sind. Kapitel 5.4 beschreibt das Vorgehen und das Ergebnis des Reviews.

5.1. Architektur-Review

Für die erstellte Architektur wurde von dem Softwarearchitekten, welcher bereits in Phase 1 Teilnehmer bei der Architekturbewertung war, ein Review durchgeführt. Das Review wurde von dem Softwarearchitekten alleine und ohne Beobachtung durchgeführt, was die Ausführlichkeit der Feldnotiz, welche für dieses Review erstellt wurde, einschränkt. In diesem Fall war es nicht möglich u. a. Empfindungen des Softwarearchitekten während des Reviews, Kommentare die er während des Reviews gesagt aber nicht notiert hat, etc. zu erheben. Zusätzlich zum durchgeführten Review des Softwarearchitekt, wurde ihm das Dokument zur Verfügung gestellt, welches die Architekturbeschreibung, die verschiedenen Architekturentscheidungen, Patterns und Best Practices, so wie die Architekturdiagramme und das Service-Diagramm enthielt. Diese Diagramme sind in den Abbildungen 4.7, 4.8, 4.9, 4.10 und 4.6 dargestellt. Für das Architektur-Review hat der Softwarearchitekt ca. zwei Stunden benötigt. Auf Basis des Architektur-Reviews wurde die Feldnotizen nachträglich erstellt.

Ergebnisse
Die Ergebnisse des Architektur-Reviews sind:

- Die ganzen verwendeten Begriffe, wie „Bounded Contexts“ etc. sollten erklärt werden.
- „Jeder Microservice verwaltet seine Bibliotheken, Logging und Licensing Einheit selbst.“ Wenn nur die Architektur betrachtet wird, ist das schon sinnvoll, wenn jedes Modul eine

eigene Lizenzierung hat, aber in unserem Fall stelle ich mir das nicht so praktikabel vor, wenn pro Service eine Lizenz erstellen und hinterlegen werden muss.
- Ein zentrales Log sollte auch auf jeden Fall implementiert werden.
- Services Diagram: was sind ZeitIntervalle?
- Services Diagram: „Main“ , „Service“ , „Library“ sind keine wirklichen Services.
- *Crm.PdfGeneration*: die PDF Generierung ist doch bereits ausgelagert in CefToPdf mit separatem Repository.
- *Crm.ProjectOrders* ist vielleicht nicht die beste Wahl für einen Microservice, da das kein Service ist, sondern über die Plugin Extensions die beiden Plugins *Crm.Order* und *Crm.Project* miteinander verknüpft.

Umsetzung der Ergebnisse
Die Begriffe wurden in dem entsprechenden Architekturdokument beschrieben. Der Hinweis zur Lizenzierung wurde ebenfalls berücksichtigt. Das Pattern *Log Aggregation* wurde berücksichtigt. Der Bounded Context ZeitIntervalle wurde erklärt. Die Bounded Contexts „Main“, „Service“ und „Library“ wurden weiter in die vorhandenen Bounded Contexts aufgeteilt. Klassen aus „Main“, „Service“ und „Library“, welche in mehrere Bounded Contexts eingefügt werden müssten, wurden für dieses Diagramm ausgelassen, da Structure 101 kein kopieren der Klassen zugelassen hat. In der späteren Implementierung werden diese Klassen in die jeweiligen Bounded Contexts importiert. Es wurde erklärt, dass für die PDF Generierung noch Klassen im Monolithen existieren und aufgerufen werden, aus diesen Klassen soll der identifizierte MS bestehen. *Crm.ProjectOrders* wurde in den *Crm.Orders* Bounded Context integriert.
Bei der in dieser Arbeit beschriebenen Architektur, handelt es sich bereits um die angepasste Architektur nach dem Review.

5.2. Code-Review

Zwei Softwareentwickler von L-mobile haben sich dazu bereit erklärt, den Code des extrahierten MS und die Anpassungen im Monolithen zu reviewen. Der erste Reviewer ist der Softwarearchitekt, welcher bereits in Phase 1 ein Teilnehmer bei der Architekturbewertung war und bereits ein erstes Architektur-Review der MSA durchgeführt hat. Der zweite Reviewer ist ein Information Security Engineer und konzentriert sich in seinem Review auf den Security-Aspekt. Für das Code-Review haben die zwei Reviewer jeweils den aktuellen Stand des Migrations-Repositories, Code des Monolithen und des Microservices, erhalten. Sowohl der Softwarearchitekt als auch der Information Security Engineer haben für das Review im Durchschnitt zwei Stunden benötigt.
Für diese Arbeit wurden nicht alle Befunde der Review-Ergebnisse umgesetzt. Darüber hinaus ist für die Weiterentwicklung geplant, diese Befunde zu berücksichtigen.

5.2.1. Ergebnisse Code-Review

Hier werden die Ergebnisse der Code-Reviews der jeweiligen Person dargestellt.

Softwarearchitekt:

1. Die Anwendung baut im CI (TeamCity[1]) nicht.
2. Der Microservice akzeptiert jetzt Get-Requests mit einem JSON im Request Body, das ist nicht richtig. Ein Get-Request sollte keinen Request Body haben. Sobald da irgendeine Art Cache zwischen Monolith und Microservice ist, gibt es Probleme.
3. PdfDTOs und *PdfServiceClient.cs* gehören ins *Crm.PdfGeneration* Projekt, bzw. ist jetzt sogar doppelt vorhanden.
4. In *PdfServiceClient.cs* ist viel duplizierter Code enthalten.
5. *PdfController.cs* ohne Autorisierung, ist das gewollt? Im HTML zu PDF Fall ist das wahrscheinlich egal, aber wie sähe das bei Microservices aus, bei denen eine Autorisierung benötigt wird?
6. *ReplaceRelativeWithAbsolutePaths* ⇒ diese Logik gehört eher nicht in den Microservice.
7. *ReplaceVariables* ⇒ das muss nicht von außen als Serviceaufruf erreichbar sein.
8. Wofür wird *CommunicationToMonolith/Host CommunicationToMonolith/Port* in den Appsettings genutzt? Der Monolith sollte dem Microservice doch nicht bekannt sein, der Microservice kann ja von verschiedenen Anwendungen heraus aufgerufen werden.
9. Viel kopierter Code von *Crm.Web/Crm.Library/Main* in den Microservice statt eine Referenz auf die benötigten Projekte. Bei dem zweiten Microservice würde das dann eher auffallen, dass das sehr viel boilerplate Code ist. Der Code gehört in ein separates Projekt, welches dann referenziert wird.
10. Da scheint überhaupt mehr drin zu sein als nötig. Der Microservice kennt jetzt die Plugin Infrastruktur, Multitenancy, initialisiert NHibernate mit einem eigenen Connectionstring, etc. Dinge, die um HTML in PDF umzuwandeln ja eigentlich nicht benötigt werden.

Information Security Engineer:

1. Web API allgemein:
   a) Authentifizierung: Nutzung des API ist derzeit ohne Authentifizierung möglich. Bei jedem Aufruf eines API-Endpunkts muss die Identität des jeweiligen Nutzers sichergestellt werden. Hierfür sollte der vorhandene Token basierte Authentifizierungsmechanismus genutzt werden.
   b) Autorisierung: Das API ist ohne Überprüfung der Autorisierung des jeweiligen Nutzers verwendbar. Die CRM und Service Anwendung verfügt über ein granulares Berechtigungsmanagementsystem, das auf den Microservice ausgeweitet werden sollte.

[1] https://www.jetbrains.com/teamcity/

   c) HTTP Verbindungen werden akzeptiert, in produktiven Umgebungen sollte diese Möglichkeit zur unverschlüsselten Übertragung von Informationen unterbunden werden.
2. DTO-Klasse
   a) Positiv zu bewerten: grundsätzlicher Einsatz von DTO. Datenkapselung ist wichtig für die Sicherheit von Microservice Architekturen, bietet hier Typsicherheit.
   b) Negativ: DTO hat in diesem Fall keine Möglichkeit zur Inhaltsvalidierung, DTOs werden direkt mit clientseitig generierten Inhalten befüllt, hier besteht prinzipiell Risiko diverser Injections oder Denial of Service (DoS) Angriffe.
   c) `byte[]` ohne Größenbegrenzung kann bei Überlastung potenziell für DoS Angriffe missbraucht werden.
3. Controller
   a) Übersichtliche Trennung einzelner Funktionen in verschiedenen Endpunkten, gut wartbar, einfach testbar.
   b) Kein Auth vorgelagert (siehe oben).
   c) Kritisch: Eingabedaten werden nicht validiert, es erfolgt lediglich eine Transformation in DTO.
   d) Eingaben landen direkt in pdfService.Html2Pdf, es wird keine Validation oder Sanitizion ausgeführt.
   e) Risiko von Code Injections in generierte PDFs, besonders kritisch da PDFs von dem Nutzer vertrauter Quelle (CRM) stammen, folglich wird den generierten PDFs implizit vertraut.
   f) Einsatz von Log4Net sehr gut, Logging ist elementar zur Nachverfolgung und Behebung potentieller Sicherheitslücken.
   g) Prüfen ob Logging richtig funktioniert, ggf. weiter ausbauen.
   h) Prüfen ob beim Loggen sensible Daten abfließen können, ggf. ausschließen.
   i) Prüfen ob TimeoutExceptions korrekt arbeiten und ggf. anpassen, kann DoS Angriffe durch lange Verzögerungen oder mutual concurrency verhindern.
   j) ReplaceRelativeWithAbsolutePaths kümmert sich zwar um relative Pfadangaben, stellt allerdings keine Absicherung gegen Path-Vulnerabilities dar, prüfen und ggf. Schutz umsetzen.
   k) Stamp und MergeFiles verarbeiten Daten ungefragt ohne Inhaltsprüfung, ggf. auf bestimmte Datentypen begrenzen.

5.2.2. Umsetzung der Code-Review Ergebnisse

Für die Umsetzung der Ergebnisse der Code-Reviews sind 22 Befunde vom Umfang und der Abgrenzung der Arbeit nicht abgedeckt. Daher werden in dieser Arbeit nicht alle Befunde der Reviews umgesetzt. Für die Befunde, welche in dieser Arbeit nicht umgesetzt werden konnten, ist geplant diese im Anschluss zu bewerten und umzusetzen.

Umgesetzte Befunde:
Auf Basis einer Aufwandsabschätzung wurde entschieden Folgende Befunde der Reviews im Rahmen dieser Arbeit umzusetzen. Vom Code-Review des Architekten wurden folgende Befunde umgesetzt: Befund 4, Befund 6 und Befund 7. Es war erforderlich die Änderungen aus Befund 6 zu verwerfen, da die automatisierten Tests nach der Umsetzung fehlschlugen. Zudem hat sich herausgestellt, dass diese Funktionalität für den Microservice erforderlich ist.
Aus dem Code-Review des Information Security Engineer werden folgende Befunde umgesetzt: 1 b) und 3 d). Nach der Umsetzung des Befundes 3 d) fiel jedoch auf, dass die externe Komponente nicht mit sanitized Strings umgehen kann, weshalb diese Umsetzung wieder verworfen wurde. Nach dieser Arbeit werden weitere Methoden recherchiert, um den String zu bereinigen mit denen auch die externe Komponente arbeiten kann. Auf Basis einer Aufwandsabschätzung wurde entschieden, die verbliebenen Befunde der Reviews im Rahmen dieser Arbeit nicht mehr umzusetzen.

5.3. Automatisierte Tests

Für die Funktionalitätsprüfung der PDF-Generierung existierten bereits sieben automatisierte Tests, welche die PDF-Generierungs-Methoden des Monolithen testen. Da die Klasse des Monolithen so abgeändert wurde, dass die ursprüngliche *PdfService.cs* Klasse die Anfrage über den PdfServiceClient an den Microservice weiterleitet, mussten die sieben automatisierte Tests in ihrer Kommunikation nicht angepasst werden. Um die automatisierte Tests auszuführen, muss eine Instanz des Monolithen und eine Instanz des Microservices gestartet werden.
Die automatisierte Tests rufen die API des Monolithen auf, welche eine Anfrage zur Generierung einer PDF erzeugen. In dieser Anfrage wird zunächst die *PdfService.cs* Klasse und daraufhin die *PdfServiceClient.cs* Klasse des Monolithen angesteuert. Die *PdfServiceClient.cs* Klasse sendet die Anfrage weiter an die API des Microservices, welche die Anfrage weiter an die *PdfService.cs* Klasse des Microservice leitet, in der schließlich die PDF generiert wird.
Alle sieben existierenden automatisierte Tests für die Generierung einer PDF wurden erfolgreich ausgeführt, ohne dass weitere Änderungen im Monolith oder Microservice vorgenommen werden mussten. Anschließend wurden drei weitere automatisierte Tests, welche Funktionen des PDF-Generierungs-Microservice testeten, die durch die existierenden automatisierte Tests noch nicht abgedeckt wurden, implementiert und erfolgreich ausgeführt. Von den zehn Tests werden zwei von sieben Methoden des PdfControllers angesteuert. Durch diese zwei API Aufrufe werden in den zehn Tests sechs der zehn Methoden der *PdfService.cs* Klasse des Microservices ausgeführt.
Wenn dieselben sieben automatisierte Tests, die bereits enthalten waren, auf einer reinen monolithischen Implementierung durchgeführt werden, laufen sie auch erfolgreich durch. Es werden auch die selben sechs von zehn Methoden der *PdfService.cs* Klasse getestet. Die selbst geschriebenen Tests lassen sich nicht im Monolithen ausführen, da diese den Microservice direkt aufrufen.
Die Tests wurden auf einem Lenovo ThinkPad mit einem 11th Gen Intel(R) Core(TM) i7-1165G7

@2.80GHz 2.80GHz Prozessor, 32GB RAM, 64-Bit-Betriebssystem Windows 10 Enterprise durchgeführt.
Durch die erfolgreiche Ausführung der automatisierten Tests, sowohl in der monolithischen als auch in der Microservices Implementierung lässt sich schließen, dass die Funktionserhaltung der PDF Generierung durch das Refactoring in einen Microservice gegeben ist.

5.4. Prozess Review

In diesem Kapitel wird das Review des PoC beschrieben. Dabei soll herausgefunden werden, ob die Arbeitsweise und das Vorgehen während der Migration des Proof of Concept für den produktiven Betrieb und die weitere Migration der Service/Sales-Applikation in Microservices dienlich sind. Dafür wurden zwei halbstündige Interviews einmal mit dem Head of Development und einmal mit dem Product Manager durchgeführt. In diesen Interviews wurden zuerst die verwendete Migrationsstrategie und das Vorgehen während des Refactorings erklärt. Anschließend wurde auf die Kommunikation zwischen Monolith und Microservice eingegangen und eine Demonstration des Zusammenspiels zwischen Monolith und Microservices präsentiert. Zum Schluss wurden die Teilnehmer danach gefragt, wie sie das Vorgehen der Migration und der Implementierung einschätzen. Die Ergebnisse der Interviews sind in Kapitel 5.4.1 für den Head of Development und Kapitel 5.4.2 für den Product Manager, angegeben.

5.4.1. Interview Head of Development

Der Head of Development fand die Ergebnisse des PoC gut. Die angewendete Migrationsstrategie und das Vorgehen während des Refactorings machen auf ihn einen geeigneten Eindruck, um dieses Vorgehen in der weiteren Migration beizubehalten. Für ihn hat der PoC sein Ziel zu beweisen, dass die Service/Sales-Applikation von L-mobile in Microservices migriert werden kann, erreicht. Ob der erstellte Code und die erstellte Architektur in der weiteren Migration der Applikation verwendet werden kann, ist für ihn noch nicht sicher, da diesbezüglich noch Unklarheiten in den Rahmenbedingungen existieren. Einige genannte Unklarheiten sind z. B. die Verwendung von Containern oder der Betrieb im Internet Information Service (IIS) sowie der Einsatz von Messaging. Außerdem ist noch nicht sicher, wie die Entwicklung von Microservices in den Entwicklungsprozess und die Strategie von L-mobile integriert werden kann. Hierfür kann der PoC und speziell das Vorgehen während der Migration des PoC als Basis für weitere Arbeiten, in denen herausgefunden werden soll, wie der Entwicklungsprozess von Microservices integriert werden kann, dienen. Weiterhin kann der PoC dafür verwendet werden, um herauszufinden, wie der spätere Betrieb der Microservices in einem Produktivsystem umgesetzt werden soll.

5.4.2. Interview Product Manager

Auch der Product Manager hat beschrieben, dass der PoC sein Ziel erreicht hat und ist mit den Resultaten zufrieden. Die ausgewählte Migrationsstrategie hat sich für den PoC geeignet und ist höchst wahrscheinlich auch für die weitere Migration der Applikation geeignet. Bevor er jedoch die Entscheidung treffen würde, diese Migrationsstrategie auch für die Migration der gesamten Service/Sales-Applikation einzusetzen, findet er es vorteilhaft und notwendig, ein oder zwei andere Migrationsstrategien in weiteren PoCs einzusetzen. Damit soll evaluieren werden, welche Strategie für die Migration der kompletten Anwendung am geeignetsten ist. Der Product Manager würde es begrüßen, wenn ein Leitfaden für die Generierung von idealen Microservices aus der Service/Sales-Applikation, welche auch die Ergebnisse der Code-Reviews berücksichtigt, erstellt wird. Dieser Leitfaden und auch der PoC können genutzt werden, um fünf bis sechs weitere Microservices verschiedener Komplexitätsklassen zu extrahieren, um auf dieser Grundlage eine Aufwandsabschätzung der vollständigen Migration anzugehen.

5.4.3. Diskussion

Beide Review Teilnehmer haben beschrieben, dass der PoC sein Ziel erreicht hat. Auch die angewendete Migrationsstrategie wurde akzeptiert. Jedoch wurde von beiden beschrieben, dass, bevor eine Migration der ganzen Service/Sales-Applikation durchgeführt werden kann, noch weitere Unklarheiten geklärt werden müssen. Beide geben an, dass für diese Unklarheiten der PoC als Vorlage dienen kann. So sollte z. B. durch die Durchführung weiterer PoCs geklärt werden, ob noch weitere mögliche Migrationsstrategie für die Service/Sales-Applikation existieren könnten. In weiteren Projekten kann der PoC auch dazu dienen, um herauszufinden wie die Entwicklung von Microservices in die Entwicklungsstrategie von L-mobile integriert werden kann und wie der Betrieb von Microservices mit Containern oder im IIS, mit REST API oder Messaging, etc., idealerweise realisiert werden kann. Außerdem sollte nach dieser Arbeit der vom Product Manager gewünschte Leitfaden für die Implementierung idealer MSs erstellt werden. Dieser Leitfaden kann als Artikel im Intranet in dem Entwicklungsbereich veröffentlicht werden. Er sollte das Vorgehen der Migration, wie in Kapitel 4.3.4 beschrieben enthalten. Außerdem sollten die Schritte in dem Leitfaden durch die Befunde der Code-Reviews ergänzt werden. Mit diesem Leitfaden können anschließend, wie beschrieben, erfahrene Entwickler weitere Microservices implementieren. Mit dieser Implementierung kann eine aussagekräftige Aufwandsabschätzung gewährt werden, welche eine Aussage darüber trifft, wie hoch der Aufwand für die gesamte Migration der Service/Sales-Applikation in Microservices sein wird.

Fazit:
Die Durchführung des PoC war den Reviews des Head of Development und des Product Managers nach zu urteilen, ein voller Erfolg. Auf Basis des umgesetzten und in dieser Arbeit

beschriebenen PoC können weitere Projekte realisiert werden, mit denen weitere Entscheidungen für die Migration der Service/Sales-Applikation in Microservices getroffen werden können.

5.5. Zusammenfassung

In diesem Kapitel wurden die Ergebnisse beschrieben, die durch die verschiedenen Reviews des PoC zustande gekommen sind.
In Kapitel 5.1 wurden die Ergebnisse des Architekturreviews aufgezeigt. Die Resultate des Architekturreviews enthielten vier größere architektonische Veränderungen:

- Lizenzverwaltung für gesamte Applikation, nicht für jeden Microservice einzeln,
- *Log Aggregator* verwenden,
- *Main*, *Service* und *Library* sind keine Microservices und
- *Crm.ProjectOrders* ist kein Service, sondern eine Plugin Extension für *Crm.Order* und *Crm.Project*.

Es wurden alle Befunde des Architekturreviews umgesetzt und die Architektur wurde für den PoC akzeptiert.

Die Code-Reviews, durchgeführt von dem Softwarearchitekten und dem Information Security Engineer, brachten insgesamt 27 Befunde hervor. Fünf Befunde wurden in dieser Arbeit umgesetzt, von denen jedoch zwei wieder verworfen werden mussten. 22 Befunde benötigen einen zusätzlichen Recherche- und Implementierungs-Aufwand, weshalb diese in dieser Arbeit nicht mehr durchgeführt werden konnten. Bevor der PoC als Vorlage für die Implementierung von Microservices genutzt werden kann, sollten diese noch nicht umgesetzten Befunde vollständig begutachtet und berücksichtigt werden.

Es existieren bereits sieben automatisierte Tests für die PDF Generierung. Diese Tests wurden für die monolithische Implementierung der Service/Sales-Applikation erfolgreich ausgeführt. Auch für die Microservices Implementierung wurden diese Tests ausgeführt und ihre Ergebnisse mit den Ergebnissen der monolithischen Ausführung verglichen. Die Testdurchläufe resultieren für die monolithische als auch für die Microservices Implementierung in der erfolgreichen Durchführung der Tests. Daraus lässt sich schließen, dass die Funktionserhaltung der PDF Generierung durch das Refactoring in einen Microservice gegeben ist.

Zum Abschluss des Reviewprozesses wurden ein Review des gesamten PoC einmal mit dem Head of Development und einmal mit dem Product Manager durchgeführt. Dafür wurde das Vorgehen während der Migration und die Ergebnisse des PoC betrachtet. Dieses Review resultierte darin, dass der PoC sein Ziel erreicht hat. Bevor die Migration der Service/Sales-Applikation in Microservices angegangen werden kann, sollten weitere Projekte durchgeführt werden, die folgende offene Punkte beantworten sollen:

- Eignet sich die Verwendung von Containern und Messaging für die L-mobile Applikation?
- Wie gestaltet sich die Integration der Implementierung von Microservices in den L-mobile Entwicklungsprozess?
- Wie gestaltet sich die Umsetzung der Microservices im Produktivbetrieb?
- Existieren weitere geeignete Migrationsstrategien?
- Welche Resultate liefert eine Aufwandsabschätzung für die gesamte Migration der Service/Sales-Applikation?

Für mögliche Folgeprojekte kann der in dieser Arbeit erstellte PoC als Grundlage dienen.

6. Auswertung der Feldnotizen

In diesem Kapitel werden die erfassten strukturierten Feldnotizen, die während der Durchführung der Migration (wie in Kapitel 4 beschrieben) erstellt wurden, ausgewertet. Dafür werden die Resultate in Kapitel 6.1 vorgestellt und eine Diskussion der Ergebnisse in Kapitel 6.2 geführt. Vor der Auswertung der strukturierten Feldnotizen wurde eine offene Codierung, zu sehen in Tabelle C.33, Zeile-für-Zeile, wie von Khandkar [Kha09] und von Seaman [Sea99] beschrieben, durchgeführt. Für die Auswertung der strukturierten Feldnotizen wurden die Methoden Anomalien, [MH94, Sea08] eingesetzt. Die unkodierten Feldnotizen sind in Anhang C angegeben.

6.1. Ergebnisse

Während der Anwendung des FMM, im Zeitraum vom 01.03.2023 bis zum 12.07.2023, wurden insgesamt 32 strukturierte Feldnotizen erhoben. Alle Feldnotizen wurden von dem Autor dieser Arbeit erstellt. Davon beziehen sich neun auf die erste Phase des FMM, sieben auf die zweite Phase und 16 auf die dritte Phase. Die Befunde in Tabelle 6.4 beziehen sich auf Themen, die sich für alle Phasen aufzeigen lassen. Befunde in den Tabellen 6.1 - 6.3 beziehen sich auf phasenspezifische Aussagen oder zeigen Aspekte auf, die in Tabelle 6.4 zusammengefasst sind. In den Tabellen 6.1 - 6.3 werden die Anzahl der Nennungen von positiven und negativen Empfindungen ihrer Aktivität, in der sie aufgetreten sind, zugeordnet. Die Anzahl an Kommentaren und sonstigen Bemerkungen werden ebenfalls ihren Aktivitäten zugeordnet. Die letzten Abschnitte der Tabelle fassen zusammen, in welchen Feldnotizen mehr Aktivitäten gut als schlecht empfunden wurden, vice versa und in welchen Feldnotizen gleich viele Aktivitäten gut wie schlecht empfunden wurden.

6.1.1. Phase 1

In Phase 1 wurden insgesamt vier Mal keine Einträge in der Kategorie „Was lief schlecht“ erhoben. Die Kategorie „Was lief gut“ hat in jeder Feldnotiz der ersten Phase einen Eintrag. Sieben Mal wurde kein Eintrag in der Zeile „Sonstiges “ hinzugefügt. Vier Mal war kein Teil der Software betroffen und zwei Mal konnte keine Aussage zur Uhrzeit erhoben werden, da sich die Aufgabe über mehrere Tage erstreckte. Zwei Mal wurde in Phase 1 von privaten Räumlichkeiten aus gearbeitet, die restliche Zeit wurde in Sulzbach an der Murr gearbeitet. Ein

Mal wurde in den „Kommentaren“ angegeben, dass der Autor bei der Architekturbewertung die Teilnehmer darauf hinweisen musste, dass die Bewertung der aktuellen Architektur und nicht der MSA gilt. Ein Mal wurde in den „Kommentaren“ angegeben, dass es interessant zu wissen wäre, wie das Ergebnis mit einer alternativen Architekturbewertungsmethode ausgefallen wäre, Feldnotiz C.9. In der Kategorie „Sonstiges“ wurde insgesamt ein Mal angegeben, was unternommen wurde, wenn der Autor in einem Schritt/bei einer Aufgabe nicht weiter wusste und weitere Quellen für die Aufgabe benötigt hat, Feldnotiz C.1. Insgesamt wurden drei Mal Verbesserungsvorschläge in den Kategorien „Kommentare“ oder „Sonstiges“ angegeben, davon zwei Mal auf das ARH Tool bezogen, Feldnotiz C.10. Fünf Mal wurde in den Feldnotizen angegeben, dass die Architektur im Fokus stand. In Feldnotiz C.7 wurde noch positiv angemerkt, dass es keine Fragen zu der Architekturpräsentation gibt. In den Feldnotizen C.9, C.8, C.6 und C.5 wurde positiv angemerkt, dass es während der Architekturbewertung gute Diskussionen gab. In den Feldnotizen C.6 und C.8 wurde jedoch auch als schlecht angemerkt, dass die Teilnehmer erst zu den Diskussionen angeregt werden mussten. Weitere Befunde der ersten Phase werden in Tabelle 6.1 zusammengefasst.

Positiven Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
1	Terminfindung ging schnell, Feldnotiz C.1.
1	Product Manager und Head of Development waren schnell überzeugt, Feldnotiz C.1.
1	Eine geeignete Architekturbewertungsmethode finden, ging gut, Feldnotiz C.1.
1	Projektbeschreibung war einfach und gut, Feldnotiz C.2.
1	Gut auf die Projektbeschreibung vorbereitet, Feldnotiz C.2.
1	Strategische Ziele ließen sich einfach identifizieren, Feldnotiz C.3.
1	Identifikation und Priorisierung der QAs war gut, Feldnotiz C.5.
3	Es gab Diskussionen zwischen den Teilnehmern der Architekturbewertung, Feldnotizen C.5, C.6, C.9.
1	Szenarien wurden gut erhoben, Feldnotiz C.6.
1	Ausführliche Präsentation der aktuellen Architektur, Feldnotiz C.7.
Negative Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
1	Suche nach Beispielen einer Architekturbewertung, Feldnotiz C.1.
2	Unklar wie mit Szenarien die Architektur bewertet

	werden soll, Feldnotizen C.1, C.7.
1	Nervös, weil die Architekturbewertung ansteht und der Vorgang einer Architekturbewertung noch nicht klar ist, Feldnotiz C.1.
1	Nervös und unsicher ob die Architekturbewertung genehmigt wird, Feldnotiz C.1.
1	Unsicher, ob alle strategischen Ziele genannt wurden, Feldnotiz C.3.
1	Es wurde während der Architekturbewertung zu oft an Microservices anstatt die aktuelle Architektur gedacht, Feldnotiz C.5.
3	Nervös und unsicher während der Architekturbewertung, Feldnotiz C.5.
2	Den Teilnehmern fiel die Aufgabe in der Architekturbewertung schwer, Feldnotizen C.5, C.6.
2	Im Stillen überlegt, keine Diskussion, Feldnotiz C.6.
2	Wenig Erfahrung des Autors und der Teilnehmer was eine Architekturbewertung angeht, Feldnotizen C.1, C.8.
1	Genervt bei der Eingabe der Szenarien ins ARH, Feldnotiz C.10.
Kommentare	
# Feldnotiz mit Kommentare	**Aktivität**
1	Planung der Architekturbewertung, Feldnotiz C.1.
1	Projekt Planung, Feldnotiz C.2.
1	Definition strategischer Ziele, Feldnotiz C.3.
5	Architekturbewertung, Feldnotizen C.5 - C.9.
1	Eingabe der Ergebnisse ins ARH, Feldnotiz C.10.
Sonstiges	
# Feldnotiz mit Sonstiges	**Aktivität**
1	Planung der Architekturbewertung, Feldnotiz C.1.
1	Eingabe der Ergebnisse ins ARH, Feldnotiz C.10.
Mehr Aktivitäten waren gut	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
3 / 2	Feldnotiz C.1
1 / 0	Feldnotiz C.2
1 / 0	Feldnotiz C.3
4 / 1	Feldnotiz C.5
2 / 0	Feldnotiz C.7

3 / 0	Feldnotiz C.9
Mehr Aktivitäten waren schlecht	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
3 / 5	Feldnotiz C.6
3 / 4	Feldnotiz C.8
Gleich viel war gut wie schlecht	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
2 / 2	Feldnotiz C.10

Tabelle 6.1.: Befunde der Feldnotizen in der ersten Phase und in welcher Aktivität diese genannt wurden. # bedeutet Anzahl.

6.1.2. Phase 2

In der Kategorie „Was lief schlecht“ wurde in den Feldnotizen von Phase 2 drei Mal kein Eintrag getätigt und zwei Mal wurde in „Was lief gut“ kein Eintrag getätigt. In Phase 2 konnte in den Feldnotizen nie eine Aussage darüber getroffen werden, welcher Teil der Applikation betroffen war. Zwei Mal wurde kein Kommentar in der Feldnotiz angegeben, drei Mal keine Empfindungen und drei Mal wurde nichts in „Sonstiges“ angegeben. Insgesamt wurden in den Kategorien „Kommentare“ und „Sonstiges“ zwei Verbesserungsvorschläge beschrieben. In Feldnotiz C.12 wurde explizit erwähnt, dass es als gut empfunden wird, dass das FMM mehrere Migrationsstrategien vorstellt, aus denen eine passende Strategie ausgewählt werden kann. Die „Kommentare“ und der Eintrag in „Sonstiges“ aus Feldnotiz C.4 sind mit der Tabelle des FMM obsolet geworden. Außerdem wurde ein Mal angegeben, dass er sich während der Ausführung der Aufgabe sicher fühlte. Weitere Befunde der zweiten Phase werden in Tabelle 6.2 zusammengefasst.

Positiven Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
5	Gute Analyse der Service-Identifikationsansätze und Migrationsstrategien, Feldnotizen C.4, C.11 - C.14.
2	Vorhandene Tabelle für die Methoden war übersichtlich, Feldnotizen C.11, C.14.
Negative Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
1	Nicht alle Veröffentlichungen sind frei oder mit Open Access verfügbar, Feldnotiz C.4.
2	Unsicherheit bei der Filterung der Methoden, Feldnotizen C.11, C.16.

1	Unkonzentriert bei der Filterung, Feldnotiz C.12.
1	Missinterpretation der SPs Filter, Feldnotiz C.14.
1	Viele Tools sind nur für Java Applikationsanalysen verfügbar, Feldnotiz C.16.
Kommentare	
# Feldnotiz mit Kommentare	**Aktivität**
1	Erste Service-Identifikationsansätze und Migrationsstrategien Analyse im ARH, Feldnotiz C.4.
2	Methoden Analyse im FMM, Feldnotizen C.11, C.14.
1	Tool-Analyse im FMM, Feldnotiz C.15.
1	Methoden Filterung im FMM, Feldnotiz C.16.
Sonstiges	
# Feldnotiz mit Sonstiges	**Aktivität**
1	Erste Service-Identifikationsansätze und Migrationsstrategien Analyse im ARH, Feldnotiz C.4.
2	Methoden Analyse im FMM, Feldnotizen C.11, C.12.
1	Methoden Filterung im FMM, Feldnotiz C.16.
Mehr Aktivitäten waren gut	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
3 / 1	Feldnotiz C.11
3 / 1	Feldnotiz C.12
3 / 0	Feldnotiz C.13
2 / 1	Feldnotiz C.14
Gleich viel lief gut wie schlecht	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
1 / 1	Feldnotiz C.4
0 / 0	Feldnotiz C.15
0 / 0	Feldnotiz C.16

Tabelle 6.2.: Befunde der Feldnotizen in der zweiten Phase und in welcher Aktivität diese genannt wurden. # bedeutet Anzahl.

6.1.3. Phase 3

Sieben Mal enthielt die Kategorie „Was lief schlecht“ der Feldnotizen aus Phase 3 keine Angaben. In den 16 Feldnotizen der Phase 3 waren die Architektur des Monolithen fünf Mal, die MSA fünf Mal, der Code sieben Mal und die Tests ein Mal in den Aufgaben der Feldnotizen betroffen. In manchen Aufgaben waren mehrere Teile der Applikation betroffen. Insgesamt wurde drei

Mal keine Aussage darüber getätigt, welcher Teil der Applikation von der jeweiligen Aufgabe betroffen war. Es wurde insgesamt sechs Mal keine Aussage zur Uhrzeit getroffen, drei Mal keine Aussage zu den Empfindungen erhoben, sechs Mal keine „Kommentare“ angegeben und elf Mal keine sonstigen Aussagen getätigt. Insgesamt wurden vier Verbesserungsvorschläge beschrieben. Interne Probleme sind vier Mal vorgekommen. Davon beziehen sich eines der internen Probleme auf das Testen, eines auf die Implementierung und zwei auf die Auswahl eines Tools für die Service-Identifizierung. In den Feldnotizen der Phase 3 wurden insgesamt 13 Mal Formulierungen/Wiederholungen verwendet, die schon mal in einer anderen Feldnotiz beschrieben waren. Viele der Wiederholungen in der Formulierung treten in Feldnotizen auf, die entweder ein Meeting zwischen Autor und Head of Development oder Product Manager behandeln, z. B. Feldnotizen C.20, C.21 und C.22 oder die während der Implementierung entstanden sind, Feldnotizen C.27, C.28 und C.29. Bei insgesamt zwei Programmieraufgaben wurde nach Hilfe von anderen Softwareentwicklern gefragt und die Hilfe wurde auch gewährt, dadurch konnten die aufgetretenen Probleme gelöst werden. In Phase 3 wurde in neun Feldnotizen angegeben, dass von Sulzbach an der Murr aus gearbeitet wurde und in elf Feldnotizen steht, dass von privaten Räumlichkeiten aus gearbeitet wurde. Bei der Anwendung der Tools für die Service-Identifikation bezieht sich die Kritik bzw. die Tätigkeiten, die schlecht liefen, nicht auf das FMM, Feldnotiz C.17. Die internen Probleme, die aufgetreten sind, haben hauptsächlich damit zu tun, dass die angebotenen Tools für die Service-Identifizierung insbesondere für Java-Applikationen implementiert wurden oder schlecht dokumentiert sind, siehe hierfür z. B. Feldnotiz C.18. Weitere Befunde der dritten Phase werden in Tabelle 6.3 zusammengefasst.

Positiven Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
1	Es gab Tools für die Service-Identifizierung, die einfach verwendet werden konnten, Feldnotiz C.17.
1	Gut identifizierte Services, Feldnotiz C.19.
1	L-mobile Stakeholder vom Fortschritt überzeugt, Feldnotiz C.19.
3	Gute Struktur des Monolithen in Bounded Contexts, Feldnotizen C.20 - C.22.
3	Generierte MSA wurde akzeptiert, Feldnotizen C.21, C.22, C.26.
1	MSA-Generierung lief gut, Feldnotiz C.23.
1	Gutes Gefühl bei der MSA-Generierung, Feldnotiz C.23.
1	Gute Filterung der Patterns, Feldnotiz C.24.
1	Gute Filterung der Best Practices, Feldnotiz C.25.
3	Gutes Review erhalten, Feldnotizen C.26, C.31, C.32.
3	Gutes Gefühl bei der Implementierung, Feldnotizen C.27 - C.29.
1	Tests für den implementierten Microservice ließen sich

	erfolgreich ausführen, Feldnotiz C.30.
Negative Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
3	Es gab Tools für die Service-Identifizierung, die nicht in Betrieb genommen werden konnten, Feldnotizen C.17, C.18, C.20.
2	Unklarheit und Unsicherheit während der MSA-Generierung, Feldnotizen C.20, C.23.
1	Unklarheit bei der Filterung der Patterns, Feldnotiz C.24.
1	Unklarheit bei der Filterung der Best Practices, Feldnotiz C.25.
3	Schwierigkeiten bei der Implementierung, Feldnotizen C.27 - C.29.
1	Schwierigkeiten bei der Testausführung, Feldnotiz C.30.
Neutrale Empfindungen/Notizen	
# Feldnotiz mit Nennungen	**Aktivität**
2	Services können, falls nötig, in späteren Iterationen noch verfeinert werden, Feldnotizen C.21, C.22.
Kommentare	
# Feldnotiz mit Kommentare	**Aktivität**
1	Testen der Tools, Feldnotiz C.17.
1	Generierung der MSA, Feldnotiz C.20.
1	Pattern-Filterung, Feldnotiz C.24.
1	Best Practices Filterung, Feldnotiz C.25.
1	MSA Review, Feldnotiz C.26.
3	Implementierung Microservice, Feldnotizen C.27 - C.29.
2	Review des implementierten Microservice, Feldnotizen C.31, C.32.
Sonstiges	
# Feldnotiz mit Sonstiges	**Aktivität**
1	Testen der Tools, Feldnotiz C.17.
1	Generierung der MSA, Feldnotiz C.23.
2	Pattern-Filterung, Feldnotizen C.24, C.25.
1	Implementierung Microservice, Feldnotiz C.27.
1	Review des implementierten Microservice, Feldnotiz C.31.
Mehr Aktivitäten waren gut	
Anzahl gut / Anzahl schlecht	**Feldnotiz**

1 / 0	Feldnotiz C.19
2 / 0	Feldnotiz C.21
2 / 0	Feldnotiz C.22
1 / 0	Feldnotiz C.23
2 / 0	Feldnotiz C.26
2 / 1	Feldnotiz C.27
2 / 1	Feldnotiz C.30
2 / 0	Feldnotiz C.31
1 / 0	Feldnotiz C.32
Mehr Aktivitäten waren schlecht	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
1 / 2	Feldnotiz C.17
1 / 2	Feldnotiz C.18
1 / 2	Feldnotiz C.24
1 / 2	Feldnotiz C.28
1 / 4	Feldnotiz C.29
Gleich viel war gut wie schlecht	
Anzahl gut / Anzahl schlecht	**Feldnotiz**
1 / 1	Feldnotiz C.20
1 / 1	Feldnotiz C.25

Tabelle 6.3.: Befunde der Feldnotizen in der dritten Phase und in welcher Aktivität diese genannt wurden. # bedeutet Anzahl.

Tabelle 6.4 fasst zusammen, welche Befunde in den Feldnotizen, aufgeteilt in ihre Phasen, aufgetreten sind.

6.2. Diskussion der Feldnotizen

In diesem Kapitel werden die Ergebnisse der strukturierten Feldnotizen diskutiert. Dabei wird zuerst auf die Architekturbewertung eingegangen. Anschließend werden die Empfindungen der beteiligten Personen diskutiert. Die Diskussion der Tätigkeiten, die gut bzw. schlecht liefen, findet anschließend statt. Danach wird diskutiert, welche Einflüsse die erfasste Uhrzeit und der Arbeitsplatz haben. Welche anderen Quellen während der Durchführung des FMM verwendet wurden und welche interne Probleme aufgetreten sind, bilden den Abschluss der Diskussion.

Die negativen Gefühle, wie Unsicherheit und Nervosität, die in den strukturierten Feldnotizen oft beschrieben werden, können damit begründet werden, dass der Autor in vielen Situationen noch keine Erfahrungen hatte. Die Empfindung Unsicherheit wurde in den Feldnotizen der

Befund	**Anzahl Phase 1** von 9	**Anzahl Phase 2** von 7	**Anzahl Phase 3** von 16	**Summe** 32
Nichts lief gut	0 (0%/0%)	2 (29%/6%)	0 (0%/0%)	2 (6%)
Nichts lief schlecht	4 (44%/13%)	3 (43%/9%)	7 (44%/22%)	14 (44%)
Mehr Dinge gut wie schlecht	6 (67%/19%)	4 (57%/13%)	9 (56%/28%)	19 (59%)
Mehr Dinge schlecht wie gut	2 (22%/6%)	0 (0%/0%)	4 (25%/13%)	6 (19%)
Gleich viele Dinge gut wie schlecht	1 (11%/3%)	3 (43%/9%)	3 (19%/9%)	7 (22%)
Keine Uhrzeit	2 (22%/6%)	0 (0%/0%)	6 (38%/19%)	8 (25%)
Keine Kommentare	0 (0%/0%)	2 (29%/6%)	6 (38%/19%)	8 (25%)
Kein Eintrag in Sonstiges	7 (78%/22%)	3 (43%/9%)	11 (69%/34%)	21 (66%)
Keine Teile der Software betroffen	4 (44%/13%)	7 (100%/22%)	3 (19%/9%)	14 (44%)
Keine Empfindungen	1 (11%/3%)	3 (43%/9%)	3 (19%/9%)	7 (22%)
Positive Empfindungen	4 (44%/13%)	2 (29%/6%)	10 (63%/31%)	16 (50%)
Negative Empfindungen	9 (100%/28%)	3 (43%/9%)	13 (81%/41%)	25 (78%)
Unsicher	8 (89%/25%)	2 (29%/6%)	4 (25%/13%)	14 (44%)
Wenig Erfahrung	2 (22%/6%)	0 (0%/0%)	2 (13%/6%)	4 (13%)
Von privaten Räumlichkeiten aus gearbeitet	2 (22%/6%)	3 (43%/9%)	11 (69%/34%)	16 (50%)
In Sulzbach an der Murr gearbeitet	7 (78%/22%)	4 (57%/13%)	9 (56%/28%)	20 (63%)
Andere Quellen	1 (11%/3%)	0 (0%/0%)	3 (19%/9%)	4 (13%)
Interne Probleme	3 (33%/9%)	1 (14%/3%)	4 (25%/13%)	8 (25%)
Wiederholungen	0 (0%/0%)	3 (43%/9%)	13 (81%/41%)	16 (50%)
Verbesserungs-vorschläge	3 (33%/9%)	2 (29%/6%)	4 (25%/13%)	9 (28%)

Tabelle 6.4.: Befunde und die Anzahl an Feldnotizen, in denen sie in den verschiedenen Phasen auftreten. In Klammern steht der prozentuale Anteil in der Phase und für die gesamte Anzahl an Feldnotizen, auf ganze Zahlen gerundet.

ersten Phase sieben Mal erwähnt. Der Hauptteil dieser Phase befasst sich mit der Architekturbewertung. Da hier weder der Autor noch die Stakeholder von L-mobile Erfahrungen in diesem Thema hatten, könnte das der Grund dafür sein, dass hier die meisten negativen Empfindungen aufgekommen sind. Die Ursache, warum die Vorbereitung auf die Architekturbewertung so zeitintensiv war (vom 01.03.23 bis 23.03.23), könnte ebenfalls damit begründet werden, dass keine Erfahrung mit Architekturbewertungen vorhanden war. Durch den Mangel an Erfahrung war es dem Autor unklar, ob die Vorbereitung für diese Aufgabe ausreichend war. Dies wurde in der Feldnotiz C.1 in den „Kommentaren“ angemerkt. Der Zeitaufwand wurde hauptsächlich für die Auswahl einer geeigneten Bewertungsmethode und der Recherche nach Erklärungen und Beispielen für Architekturbewertungen erbracht. Deshalb beziehen sich die externen Quellen dieser Phase hauptsächlich auf die Suche nach Beispielen für eine Architekturbewertung, da hier unklar war, wie die Architekturbewertung angegangen werden kann, bzw. wie eine Architektur mit Szenarien bewertet werden kann. In dieser Phase wurde vom Autor der Vorschlag unterbreitet, im FMM Beispiele und genauere Beschreibungen für Architekturbewertungen hinzuzufügen.

Positive Empfindungen kamen in Phase 3 am häufigsten vor, sowohl von der Anzahl als auch prozentual. Positive Empfindungen treten am häufigsten während der Implementierung in Phase 3 auf, Feldnotizen C.27 - C.29. Dies deutet darauf hin, dass Resultate der vorherigen Aufgaben, wie die Architekturgenerierung, Filterung der Migrationsstrategien, Patterns und Best Practices von guter Qualität sind. In Phase 3 kommen ebenfalls die meisten negativen Empfindungen vor, 13 von der Anzahl. In dieser Phase wird am häufigsten in den Feldnotizen, welche die MSA-Generierung betreffen, von Unsicherheit und wenig Erfahrung berichtet, Feldnotizen C.17 - C.25. Die Empfindung Unsicherheit kommt Phase 3 sieben Mal vor. Wenig Erfahrung in der Generierung einer MSA wurden auch in der Studie von Fritzsch et al. [FBWZ19] in der Industrie häufiger beschrieben. Jedoch können die meisten negativen Empfindungen in Phase 1 beobachtet werden. In 100% der Feldnotizen der Phase 1 stehen negative Empfindungen. Hier könnte mangelnde Erfahrung in der Implementierung von Microservices der Grund sein. Weil in Phase 2 lediglich zwei Unsicherheiten vorkommen, lässt darauf schließen, dass das FMM gut durch den Filterungsprozess der verschiedenen Service-Identifikationsansätze und Migrationsstrategien führt.
Insgesamt wurden sieben Mal keine Empfindungen in den Feldnotizen eingetragen, ein Mal in der Besprechung strategischer Ziele, zwei Mal bei den Service-Identifikationsansatz und Migrationsstrategie Filterungen, zwei Mal während der Tool-Analyse, ein Mal bei der Best Practice Filterung, ein Mal im MSA Review. Die Besprechung der strategischen Ziele hat über einen Microsoft-Teams-Anruf stattgefunden, daher konnten hier keine Empfindungen gesammelt werden. Das MSA Review wurde von dem Softwarearchitekten alleine durchgeführt und nur die Ergebnisse mitgeteilt, daher konnten hier keine Empfindungen analysiert werden. Die restlichen Feldnotizen ohne Empfindungen beziehen sich auf die Filterung der Service-Identifikationsansätze und Migrationsstrategien, der Tool-Filterung für die Service-Identifikation und der Best Practices. Da sich mehrere Feldnotizen auf die Filterung der Service-Identifikationsansätze und Migrationsstrategien beziehen und die Feldnotizen ohne Empfindungen gegen Schluss dieser Aufgabe auftreten, könnte daran liegen, dass nach ein paar Tagen

die Filterung gut eingearbeitet war und sich gut ausführen ließ. Da die Tool-Analyse ohne Empfindungen direkt im Anschluss an die Methodenfilterung stattgefunden hat, war dem Autor, der die Analyse durchgeführt hat, bewusst, auf welche Aspekte er achten muss. Die Filterung der Best Practices hat im Anschluss an die Pattern-Filterung stattgefunden und die fehlenden Empfindungen können hier ebenfalls damit begründet werden, dass der Teilnehmer mit dem Filterungsschema schon vertraut war und wusste, wie er diese durchzuführen hat.

Es kam ausschließlich in Phase 2 vor, dass in der Kategorie „Was lief gut“ kein Eintrag getätigt wurde. Dies war insgesamt zwei Mal in den Feldnotizen der Phase 2 der Fall. Diese zwei Feldnotizen beziehen sich auf die letzten Schritte der Filterung der Service-Identifikationsansätze und Migrationsstrategien und auf die Tool Analyse. Davor wurde bereits ein Teil der Filterung der Service-Identifikationsansätze und Migrationsstrategien durchgeführt. In diesen Feldnotizen kommt die Filterung zu einem Ergebnis. Die Kategorie „Was lief schlecht“ hat in diesen Feldnotizen ebenfalls keine Einträge. Eine mögliche Ursache dafür könnte sein, dass abweichend von den vorher erstellten Feldnotizen für die Filterung keine Tätigkeiten besser oder schlechter abgelaufen sind und deshalb auf diese Einträge, in den letzten beiden Feldnotizen, verzichtet wurde. Es kam nur zwei Mal vor, dass kein Eintrag in die Kategorie „Was lief gut“ getätigt wurde. Da in den restlichen 30 Feldnotizen mindestens ein Eintrag in dieser Kategorie steht und im Vergleich dazu die Kategorie „Was lief schlecht“ insgesamt 14 Mal keinen Eintrag besitzt, lässt sich darauf schließen, dass die Anwendung des FMM gut durch die Migration führt.
Ein weiterer Punkt, der dafür spricht, ist, dass in jeder Phase des FMM mehr Tätigkeiten pro Feldnotiz gut wie schlecht waren, als dass mehr Tätigkeiten pro Feldnotiz schlecht wie gut waren. In Phase 2 waren sogar nie mehr Tätigkeiten pro Feldnotiz schlecht wie gut. Es kam auch in keiner Phase vor, dass öfters gleich viele Tätigkeiten pro Feldnotiz gut wie schlecht waren, als dass mehr Tätigkeiten pro Feldnotiz gut wie schlecht waren.
Die Kategorie „Was lief schlecht“ hat in den verschiedenen Phasen prozentual gesehen keinen Unterschied darin, dass sie einmal öfter keinen Eintrag hat. Es gibt also keinen wirklichen Unterschied der Häufigkeit in den Kategorien „Was lief gut“ und „Was lief schlecht“ zwischen den Phasen.

Die Uhrzeit hat keinen Einfluss auf Empfindungen oder darauf, ob mehr Tätigkeiten pro Feldnotiz gut wie schlecht liefen. Die einzige Anomalie ist, dass in den Aufgaben, die nur vormittags durchgeführt wurden, mehr positive als negative Empfindungen aufkamen und dass in diesen Aufgaben nie mehr Tätigkeiten pro Feldnotiz schlecht als gut liefen. In Phase 3 wurden in den Feldnotizen, welche die Implementierung und das Testen behandeln, keine Uhrzeit angegeben. In Phase 1 haben die Feldnotizen, welche Vorbereitungen für die Architekturbewertung und die Projektbeschreibung dokumentieren, keine Uhrzeit. Es ist normal, dass diese Schritte mehrere Tage in Anspruch nehmen und dadurch in einer Feldnotiz zusammengefasst werden können. Üblicherweise wurden in den Feldnotizen nur keine Angaben zur Uhrzeit notiert, wenn sich die in den Feldnotizen behandelte Aufgabe über mehrere Tage erstreckte. Dies kam nur in Phase 1 und Phase 3 vor.
Der Arbeitsplatz hatte ebenfalls keinen Einfluss auf Empfindungen oder darauf, dass mehr Tätigkeiten pro Feldnotiz gut wie schlecht liefen vice versa. In beiden Orten gab es mehr

negative als positive Empfindungen und es sind in beiden Orten mehr Tätigkeiten gut als schlecht gelaufen.

Weiterhin spricht dafür, dass das FMM gut durch den Migrationsprozess führt, dass es insgesamt 21 Mal keine Einträge in der Kategorie „Sonstiges“ gab. Diese Kategorie war ursprünglich dafür gedacht zu dokumentieren, welche Methoden außerhalb des FMM verwendet wurden und Verbesserungsvorschläge zu unterbreiten. Da in der Kategorie „Sonstiges“ aber insgesamt nur elf Einträge erhoben und insgesamt nur sieben Verbesserungsvorschläge unterbreitet wurden, lässt sich daraus schließen, dass das FMM bereits umfangreiche Unterstützung für den Migrationsprozess bereitstellt.
Es kam nur in Phase 1 und Phase 3 vor, dass insgesamt fünf Mal andere Quellen (in vier Feldnotizen) außer dem FMM konsolidiert werden mussten. Davon bezieht sich eine Quelle auf das YouTube-Video mit dem sich die Grundlagen von der Microservices-Implementierung in .Net angeeignet wurden. Dieser Punkt ist kein Nachteil/Fehler des FMM, da dieses Programmiersprachen- und technologieunabhängig durch den Migrationsprozess führen soll. Dennoch könnte dieses Video als Beispiel dem FMM hinzugefügt werden. Ein weiterer Eintrag, der andere Quellen erwähnt, ist, dass weitere Beispiele für MSs in .Net gesucht wurden. Diese weiteren Quellen kamen jedoch nicht zum Einsatz. Eine weitere Quelle aus den Feldnotizen aus Phase 3 bezieht sich auf weitere Microservices Patterns, welche im Internet recherchiert wurden. Es könnten weitere MS Patterns recherchiert und dann dem FMM hinzugefügt werden. Eine letzte externe Quelle in Phase 3 war eine erweiterte Internetrecherche nach weiteren Tools für die Service-Identifizierung und Migrationsstrategien. Diese erweiterte Recherche wurde vom Betreuer der Uni vorgeschlagen, um im Idealfall das FMM zu erweitern, jedoch lieferte diese Recherche keine Resultate. Die Tatsache, dass Informationen aus externen Quellen eingesetzt wurden, kann als Anomalie angesehen werden und deutet darauf hin, dass das FMM ausreichend Informationen zur Verfügung stellt, um gut durch den Migrationsprozess zu leiten.

Interne Probleme sind in allen Phasen aufgetreten. In Phase 1 beziehen sie sich hauptsächlich darauf, dass die Teilnehmer der Architekturbewertung bereits zu sehr auf die MSA als auf die aktuelle Architektur eingehen oder dass die Teilnehmer während der Architekturbewertung zu wenig kommunizieren und wenig Erfahrung haben. In Phase 2 und Phase 3 beziehen sich zwei der internen Probleme darauf, dass die meisten der angebotenen Tools für die Service-Identifizierung nur für Java-Applikationen implementiert sind. Dies ist kein Problem des FMM, da die Umsetzung der Tools und Frameworks für die Service-Identifizierung und Migration nicht im Bereich des FMM liegt. Außerdem wurden passende Methoden für die Service-Identifizierung und Migration aus den vom FMM vorgeschlagenen Methoden herausgefiltert. In Phase 3 werden auch vier interne Probleme während der Implementierung oder des Testens erwähnt. Zwei dieser internen Probleme kamen nur dadurch zustande, dass die Anwendung nicht korrekt gestartet wurde.

Aus der Evaluation der erhobenen Feldnotizen lässt sich schließen, dass alle Phasen des FMM gut durch die Migration leiten.

7. Diskussion

In diesem Kapitel findet die Diskussion für die einzelnen Forschungsfragen statt. Dieses Kapitel ist nach den Forschungsfragen strukturiert. Am Ende dieses Kapitels findet sich die Gefährdung der Geltung.

7.1. Forschungsfrage 1

Welche Frameworks werden in der wissenschaftlichen Literatur für die Umstrukturierung eines Monolithen in Microservices vorgeschlagen und welche Herangehensweisen bzw. Prozesse verfolgen diese?

In Kapitel 2.5 wurden zwei Frameworks und drei Tools aus der wissenschaftlichen Literatur, so wie zwei Frameworks aus der Industrie beschrieben. Diese Sammlung von Frameworks und Tools ist keinesfalls vollständig, zumal das FMM über 90 Frameworks und Tools für die Service-Identifizierung und Migration vorschlägt. Außerdem wurden die Beschreibungen der Herangehensweisen der Frameworks und Tools nur angerissen und auf das Nötigste beschränkt. Jedoch gibt diese Zusammenfassung der Frameworks und Tools einen Einblick, wie die Arbeitsweisen dieser Frameworks und Tools aussehen kann.

So verwendet das Framework Log2MS von Liu et al. [LXR+22] Ausführungs-Logs um Microservices in einem Monolithen zu identifizieren. Das Framework von Taibi und Systä [TS19] verwendet ebenfalls Ausführungs-Logs, verwendet aber andere, Schritte um die Microservices zu identifizieren.
Abgesehen von der dynamischen Analyse eines Monolithen, z. B. durch Ausführungs-Logs existieren noch andere Wege Microservices in einer monolithischen Architektur zu identifizieren. So teilen z. B. Pigazzini et al. [PAM19], mit ihrem Tool Arcan, Monolithen durch eine statische Analyse in Microservices auf, indem sie Architekturmängel (Architektur-Smells) identifizieren.
Eine weitere Methode, einen Monolithen in Microservices zu migrieren, stellen Zaragoza et al. [ZSS+21] mit ihrem Tool MonoToMicro vor. Sie Clustern die Klassen eines Monolithen und beseitigen anschließend Verkapselungsverstöße sogenannter Randklassen, um Microservices-Kandidaten zu finden.
Das letzte in dieser Arbeit, unabhängig von den Vorschlägen des FMM, betrachtete Tool aus der wissenschaftlichen Literatur, „Architect“ von Volynsky et al. [VMK22], migriert Monolithen in einem sogenannten Greenfield Projekt, während die anderen Methoden einen Brownfield

Ansatz verfolgen. Im Gegensatz zu den anderen hier vorgestellten Methoden benutzt „Architect“ keinen Code oder Ausführungs-Logs, sondern identifiziert die Microservices mit der Architekturbeschreibung.
Die zwei Tools, die in der Industrie verwendet werden, sind Mono2Micro, beschrieben von Gulati et al. und Kalia et al. [IBM20, KXK+21], und der AWS Microservice Extractor für .Net [AWS23a, AWS23b]. Mono2Micro verwendet künstliche Intelligenz, um Laufzeit-Aufrufspuren zu analysieren und dadurch die Microservices zu identifizieren [IBM20, KXK+21]. AWS Microservice Extractor für .Net hingegen visualisiert die Komponenten des Quellcodes samt ihrer Kommunikation und unterstützt den Softwarearchitekten bei der Gruppierung der Klassen in Microservices [AWS23a, AWS23b].

Ein Framework wie das FMM [FBH+22, Ins23], welches mehrere Methoden für die Service-Identifizierung und die Migration von Monolithen in Microservices vorschlägt, wurde in der wissenschaftlichen Literatur und in der Industrie nicht gefunden.

7.2. Forschungsfrage 2

Inwieweit eignet sich das FMM des ISTE ESE, um den L-mobile Monolithen in Microservices umzustrukturieren?

Um diese Forschungsfrage zu beantworten, wird auf die einzelnen Phasen separat eingegangen. In Phase 1 befindet sich die größte Differenz zwischen negativen Empfindungen, insgesamt zehn und positiven Empfindungen, insgesamt zwei. Dies liegt nicht unbedingt am FMM, sondern viel mehr daran, dass der Autor und die Stakeholder bei L-mobile noch keine Erfahrungen mit Architekturbewertungen, wie sie in dieser Arbeit durchgeführt wurde, hatten. Dieser Punkt wird auch in den Feldnotizen zwei Mal erwähnt, Feldnotiz C.1 und Feldnotiz C.8. Vielleicht gibt es Möglichkeiten, dass das FMM Unternehmen, in denen niemand eine Architekturbewertung durchgeführt hat, in Phase 1 besser unterstützt. Die Tatsache, dass das FMM im ersten Schritt eine Architekturbewertung vorsieht und die Resultate der Architekturbewertung, die in dieser Arbeit durchgeführt wurde, das Unternehmen weiter bringt, ist auf jeden Fall ein guter Start in die Migration.
Damit diese Forschungsfrage für Phase 2 beantworten werden kann, muss auch auf Forschungsfrage 3 eingegangen werden. In Forschungsfrage 3 wurde beantwortet, welche Migrationsverfahren für die L-mobile Applikation geeignet sind. Es kam heraus, dass die durch die Anwendung des FMM in Phase 2 inkludierten Verfahren von Krause et al. [KZH+20] und von Li et al. [LML20] für den Proof of Concept die richtige Wahl waren und wahrscheinlich auch die richtige Wahl für die Migration der kompletten Applikation sind, Kapitel 7.3. Demnach eignet sich das FMM auf jeden Fall für die Auswahl geeigneter Service-Identifikationsansätze und Migrationsstrategien.
In Phase 3 unterstützt das FMM die Generierung der MSA gut durch die Filterung von Patterns und Best Practices. Jedoch sollte der Filterungsprozess etwas angepasst werden, da sonst manche Patterns und Best Practices ausgefiltert werden, die evtl. relevant sein können bzw.

werden Patterns und Best Practices nicht gefiltert, die evtl. für das zu migrierende System nicht relevant sind, Kapitel 4.3.1 und Kapitel 4.3.2 bzw. Tabelle A.1 und Tabelle A.2. Für die Service Implementierung enthält das FMM zum Zeitpunkt, zu dem diese Arbeit verfasst wird, noch keine Unterstützung. Hier könnte auf Microservices Beispiele für verschiedene Programmiersprachen, verwiesen werden wie z. B. GitHub Repositories oder YouTube-Videos, wie das in dieser Arbeit verwendete Video[1].

Insgesamt lässt sich sagen, dass das FMM für die Migration der Service/Sales-Applikation von L-mobile sehr gut geeignet ist. Dies wird auch daran erkannt, dass sowohl der Head of Development und der Product Manager der Meinung sind, dass der PoC sein Ziel erreicht hat und die ausgewählten Methoden für die Migration geeignet sind, Kapitel 5.4.

7.3. Forschungsfrage 3

Welche Refactoring-Verfahren und Migrationsverfahren eignen sich am besten für die Dekomposition des L-mobile Monolithen in eine Microservices-Architektur?

In Kapitel 4.2 wurde ausführlich die Filterung der vom FMM vorgeschlagenen Service-Identifikationsansätze und Migrationsstrategie besprochen. Nach dieser Filterung kamen elf Publikationen [AM21], [DBT21], [DBFP18], [CGC+20b], [HLT18], [LML20], [SK17], [ZSS+21], [ZZ21], [KZH+20] und [NUEH18] für die Migration des L-mobile Monolithen in Frage. Aus diesen elf Publikationen wurden zwei Methoden, eine für die Service-Identifizierung und eine für die Migration, ausgewählt. Die hier ausgewählten Methoden für die Service-Identifikation von Krause et al. [KZH+20] und die Migration von Li et al. [LML20] eignen sich für die Dekomposition des L-mobile Monolithen sehr gut.
Obwohl bei der Service-Identifikation das Tool Exploreviz nicht ausgeführt werden konnte und keine Aufteilung der Datenbank durchgeführt wurde, wurden die identifizierten Microservices mit wenig Anpassungen vom Softwarearchitekten, Head of Development und dem Product Manager akzeptiert, siehe Feldnotizen C.19, C.21, C.22 und C.26. Das spricht dafür, dass sich mit dem Tool Structure 101 bereits geeignete Microservices-Kandidaten identifizieren lassen. Es sollte hier jedoch beachtet werden, dass die Applikation von L-mobile bereits gut in ihre Bounded Context aufgeteilt ist und diese mithilfe der Service-Identifikationsansatz von Krause et al. [KZH+20] nur noch verfeinern ließen. Trotzdem oder gerade deshalb eignet sich das Vorgehen von Krause et al. [KZH+20] sehr gut für die Identifizierung von Microservices-Kandidaten für die Service/Sales-Applikation von L-mobile.
Die Migrationsstrategie von Li et al. [LML20] eignet sich ebenfalls sehr gut für die Migration der Service/Sales-Applikation von L-mobile. Während der Anwendung dieser Strategie kam es zu keinen großen Problemen. Der ausgewählte Microservice-Kandidat ließ sich einfach und schnell extrahieren. Der Monolith und der Microservice konnten gleichzeitig betrieben

[1] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg

werden und ihre Kommunikation miteinander funktioniert. Der Monolith kann auch ohne den extrahierten Microservice betrieben werden. Außerdem wurde von dem Product Manager im Review des Proof of Concept, Kapitel 5.4.2 gesagt, dass die ausgewählte Migrationsstrategie für den Proof of Concept gepasst hat und wahrscheinlich auch die geeignetste Strategie für die Migration der kompletten Applikation ist. Der Head of Development hat sich geäußert, dass die Migrationsstrategie für den Proof of Concept geeignet erscheint, Kapitel 5.4.1.

7.4. Forschungsfrage 4

Welche allgemeine und L-mobile Monolith-spezifische Herausforderungen treten bei der Umstrukturierung auf?

Die allgemeinen Herausforderungen, die bei einer Migration von Monolithen in Microservices auftreten können, wurden in Kapitel 2.3.1 besprochen.
Hier wurden Herausforderungen wie Mehrmandantenfähigkeit, Zustandsfestigkeit und Datenkonsistenz, welche von Furda et al. [FFZ+17] beschrieben wurden, genannt.
Weiterhin wurden Herausforderungen, die von Velepucha und Flores [VF21] identifiziert wurden, genannt. Dazu gehören die Auswahl eines geeigneten Tools, Frameworks, Methode oder Modell für die Migration, die Änderung der hierarchischen Organisationsstruktur, die Abhängigkeit von Frameworks, die Priorisierung der Funktionalitäten für die Migration, die Generierung von Logs, eine adäquate Analyse des zu migrierenden Systems und gute Entscheidungen zu treffen [VF21].
Die Migration der Datenbank, die Zerlegung der Geschäftsfähigkeiten, der Bedarf an fähigen Entwicklern, Ressourcenmanagement, Expertenbetrachtung, komplexe Umgebungseinstellungen, Implementierung von Transaktionen und die Adaption der Arbeit mit Microservices sind Herausforderungen, die von Ponce et al. [PMA19] identifiziert wurden.
Zuletzt wurden noch Herausforderungen betrachtet, die von Fritzsch et al. [FBWZ19] identifiziert wurden. Dazu gehören die Service-Identifizierung bzw. den korrekten Servicecut zu finden, die Integration der Kommunikation, eine gewisse Fehlertoleranz zu erreichen, die Synchronität des Alt- und Neusystems während der Migration, das Orchestrieren der Microservices, das Integrieren von externen Systemen, das Aufsetzen der neuen Umgebung, die Datenbankmigration, der Wechsel von einem zustandsfesten zu einem zustandslosen System, die Verteilung der Aufgaben an Entwickler, der Wissenstransfer und das Erwerben der benötigten Fähigkeiten [FBWZ19].
Hervorzuheben ist, dass die Migration der Datenbank eine der am häufigsten genannte Herausforderung bei einer Migration eines Monolithen in Microservices ist, genannt von [FFZ+17, PMA19].

Herausforderungen, die bei der Migration der L-mobile Applikation in Microservices aufgetreten sind, lassen sich in der Auswertung der Feldnotizen, siehe Kapitel 6, nachlesen. Dazu gehören wenig Erfahrung des Autors und der Stakeholder von L-mobile mit der Architekturbewertung, dass viele der Tools für die Service-Identifizierung nur für Java-Applikationen

implementiert oder schlecht dokumentiert sind und dass für jeden Service selbst entschieden werden muss, wie die Datenbank migriert werden soll. Die Herausforderungen, welche während der Implementierung aufgetreten sind, wie dass das Problem mit dem Logging erst nach 16 Tagen gelöst wurde, dass es zu Problemen mit der *DomainExtension* und mit der Kommunikation zwischen Monolith und Microservice kam, werden hier nicht berücksichtigt, da dies Herausforderungen sind, die in jeder Implementierung auftreten können. Außerdem wurden diese Herausforderungen schnell gelöst. Die Befunde aus den Code-Reviews werden hier ebenfalls nicht aufgelistet, da diese ebenfalls in jeder anderen Implementierung auftreten können. Die Anzahl der Befunde aus den Code-Reviews kam wahrscheinlich dadurch zustande, dass der Autor wenig Erfahrung mit Microservices Entwicklung hat und noch nicht sicher entscheiden konnte, welche Abhängigkeiten aus dem Microservices-Code entfernt werden können. Außerdem hatte er wenig Erfahrungen mit den in der Service/Sales-Applikation verwendeten Security-Mechanismen.

7.5. Gefährdung der Geltung

Zu beachten ist, dass der Großteil dieses Projekts, Vorbereitung der Architekturbewertung, Auswahl der Service-Identifikationsansätze und Migrationsstrategie, Architekturerstellung, Implementierung nur von einem Softwareentwickler mit wenig professioneller Erfahrung durchgeführt wurde. Um hier den Softwareentwickler zu unterstützen, wurde deshalb Feedback und Reviews von erfahreneren Softwareentwicklern, Softwarearchitekten, Stakeholdern und Betreuern eingeholt. Viele der beteiligten Personen haben ein Interesse an einer guten Bewertung dieser Arbeit und könnten dadurch geneigt sein, vermehrt positives Feedback als ehrliches Feedback zu geben. Diesem Punkt wurde entgegengewirkt, indem klar gesagt wurde, dass ein negatives Ergebnis der Evaluation nicht zu einer schlechten Bewertung dieser Arbeit führen würde. Außerdem wirkt der Eintrag „Empfindungen“ der Feldnotizen diesem Punkt entgegen, da hier auch negative Empfindungen erfasst wurden. Die Erfassung und Evaluierung der strukturierten Feldnotizen wurde von einer Person ausgeführt, welche noch keine Erfahrung mit dieser Art der Evaluation hatte. Um die Erfassung und Auswertung der Feldnotizen aussagekräftig durchzuführen, wurden etablierte Methoden für die Erfassung und Auswertung sowie Beispiele für Feldnotizen und ihre Auswertung recherchiert. In der Implementierungsphase wurde nur ein kleiner Service mit wenig Kommunikation zum Monolithen und ohne Datenbankanbindung migriert. Die Ergebnisse könnten anders ausfallen, wenn größere Services migriert werden und die Datenbank aufgeteilt werden muss. Da die Service/Sales-Applikation bereits in Module aufgeteilt ist, dürfte die Migration weiterer Services auf der Code-Ebene ähnlich ablaufen wie in dieser Arbeit. Für die Migration der Datenbank wurden etablierte Methoden recherchiert und das Vorgehen, für den Fall dass eine Datenbanktabelle aufgeteilt werden muss, mit Stakeholdern von L-mobile besprochen. Außerdem wurde nur eines der drei Tools aus Krause et al. [KZH+20] ausgeführt. Die identifizierten Services könnten anders ausfallen, wenn Exploreviz ebenfalls zum Einsatz kommt, da hier weitere Bounded Contexts entdeckt oder bestehende Bounded Contexts verfeinert werden könnten. Um sicherzustellen,

dass die, durch die Anwendung von Structure 101, identifizierten Services geeignet für die Migration sind und keine weitere Verfeinerung der Bounded Contexts nötig ist, wurde Feedback von dem Softwarearchitekten, dem Head of Development und dem Product Manager eingeholt. Die Filterung der Service-Identifikationsansätze und Migrationsstrategien könnten in anderen Ansätzen und Methoden resultieren, wenn der Fokus der Filterung auf andere Filterungsaspekte gelegt wird. Dies könnte in anderen identifizierten Microservices und einem anders durchgeführten Migrationsprozess resultieren. Es wurde darauf geachtet, möglichst alle Filterungsaspekte bei der Filterung der Service-Identifikationsansätze und Migrationsstrategien zu berücksichtigen. Aufgrund der Verfügbarkeit der Stakeholder von L-mobile wurde sich für eine Architekturbewertungsmethode entschieden, welche sich innerhalb eines Nachmittags durchführen lässt. Um eine aussagekräftige Architekturbewertung durchzuführen, wurde darauf geachtet, eine Methode auszuwählen, in welcher mithilfe von Qualitätsattributen und Szenarien die Architektur bewertet wird und die sich an ATAM oder SAAM orientiert. Weiterhin ist zu beachten, dass es sich um einen konkreten Applikationstyp handelt, der in dieser Arbeit migriert wurde. Für andere Applikationen können die Resultate anders ausfallen. Jedoch kann das Resultat dieser Arbeit, dass das FMM für die Migration der Service/Sales-Applikation von L-mobile geeignet ist, auf gut strukturierte objektorientierte modulare Monolithen übertragbar sein, welche sich im Größenbereich und der Domäne der Service/Sales-Applikation von L-mobile befinden.

8. Fazit und Ausblick

Das Ziel dieser Arbeit war es, eine Teilmigration der Service/Sales-Applikation von L-mobile unter Verwendung des Framework für Microservices-Migrationen der Abteilung Empirical Software Engineering des Institute of Software Engineering durchzuführen. Zuerst wurden Grundlagen der Arbeit recherchiert. Dafür wurden Monolithen und Microservices charakterisiert. Es wurde definiert, was Software Migrationen im Allgemeinen sind und was eine Migration eines Monolithen in Microservices ist. Anschließend wurden allgemeine Herausforderungen, die bei einer Migration eines Monolithen in Microservices auftreten können, recherchiert. Dabei zeigte sich, dass die Migration der Datenbank eine der am häufigsten genannte Herausforderung bei einer Migration eines Monolithen in Microservices ist, genannt von [FFZ+17, PMA19]. Zum Abschluss des Migrationskapitels wurden Gründe, die für oder gegen eine Migration in Microservices sprechen, recherchiert. Die letzten Kapitel der Literaturrecherche stellen das FMM und die verwandten Arbeiten vor. Es wurden strukturierte Feldnotizen [Sea08] für die Evaluation der Arbeit mit dem FMM bei der Migration der Service/Sales-Applikation von L-mobile eingesetzt.
Kapitel 4 widmet sich der Durchführung des FMM auf die Service/Sales-Applikation von L-mobile. Phase 1, Kapitel 4.1, behandelt das Vorgehen während der Planung und Ausführung der Architekturbewertung. Es wurde begründet, warum sich gegen die Methoden ATAM [KKC00], SAAM [KBAW94] und die Methode von Auer et al. [ALFT21] entschieden wurde. Schlussendlich wurde sich für die Methode von Svahnberg und Mårtensson [SM07] entschieden, da sich diese an einem Nachmittag durchführen lässt und dennoch brauchbare Ergebnisse für die Teilmigration dieses System erwarten ließ. Obwohl bei den Stakeholdern als auch bei dem Autor keine Erfahrung mit Architekturbewertungen vorhanden war, ließ sich diese Methode gut durchführen. Während der Architekturbewertung wurden die relevanten Qualitätsattribute für die Service/Sales-Applikation von L-mobile anhand des ISO 25010[1] identifiziert. Als die drei wichtigsten Qualitätsattribute wurden *Security*, *Reliability* und *Functional Suitability* identifiziert. Für diese drei Qualitätsattribute wurden für jedes Subattribut jeweils zwei Szenarien erhoben und bewertet. Anschließend wurde die Architektur der Service/Sales-Applikation von dem Softwarearchitekten präsentiert und es wurde analysiert, ob die aktuelle Architektur die Szenarien erfüllt. Das Resultat der Architekturbewertung war, dass die aktuelle Architektur die Szenarien noch erfüllt, es aber von Vorteil ist, sich mit der Migration in Microservices zu befassen.

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

In Phase 2 wurden die vom FMM vorgeschlagenen Service-Identifikationsansätze und Migrationsstrategien auf die Anwendbarkeit auf die Service/Sales-Applikation hin analysiert und gefiltert. Aus den elf Methoden, die nach dieser Filterung übrig blieben, wurden die Methoden von Krause et al. [KZH+20] für die Service-Identifikation und die Methode von Li et al. [LML20] für die Migration ausgewählt und eingesetzt. Obwohl aus den drei vorgestellten Tools aus Krause et al. [KZH+20] nur Structure 101 ausgeführt wurde, wurden die mit diesem Tool identifizierten Services vom Head of Development, dem Product Manager und dem Softwarearchitekten akzeptiert. Die Migration eines dieser identifizierten Services wurde anhand des in Li et al. [LML20] vorgestellten Vorgehens für die Migration erfolgreich ausgeführt.
Das Kapitel zu Phase 3 beschreibt wie die Services mit der ausgewählten Methode identifiziert, Patterns und Best Practices gefiltert und eine Microservices-Architektur mit diesen Resultaten generiert wurde. Es wurden insgesamt 41 Microservices identifiziert. Für eine erste Microservices-Architektur wurden 14 Patterns und 14 Best Practices ausgewählt. Von dem Softwarearchitekten wurde ein Review für die generierte Microservices-Architektur durchgeführt. Die Microservices-Architektur wurde, mit einer Anpassung von 4 Microservices, akzeptiert. Außerdem wurde angemerkt das *Log Aggregator* Pattern ebenfalls in die Microservices-Architektur mit aufzunehmen. Auch der Head of Development und der Product Manager haben die Microservices-Architektur akzeptiert. Anschließend beschreibt Kapitel 4.3 wie die identifizierten Services priorisiert und ein Service implementiert wurde.
In Kapitel 5 wird beschrieben, wie Code-Reviews und automatisierte Tests nach der Implementierung des ersten Microservice durchgeführt wurden. Alle sieben durchgeführten automatisierten Tests lieferten für die monolithische als auch für die Microservices Implementierung dieselben Ergebnisse. Die Code-Reviews, durchgeführt von dem Softwarearchitekten und Information Security Engineer, lieferten 27 Befunde, von denen nach einer Aufwandsabschätzung 5 Befunde in dieser Arbeit noch umgesetzt wurden. Kapitel 5 schließt mit einem Review des Proof of Concept durch den Head of Development und dem Product Manager ab. Das Resultat dieses Reviews war, dass der Proof of Concept sein Ziel erreicht hat und die angewendeten Methoden sowie das FMM auch für die Migration der kompletten Service/Sales-Applikation geeignet sind.
Die Arbeit wird mit der Auswertung der strukturierten Feldnotizen abgeschlossen. Diese Auswertung zeigte, dass das FMM, obwohl wenig Erfahrung vom Autor mit Microservices vorhanden war und er sich oft unsicher gefühlt hat, gut für die Migration der Service/Sales-Applikation von L-mobile geeignet ist. Es wurden außerdem Verbesserungsvorschläge, wie die Bereitstellung von Beispielen und Grundlagen für die Implementierung von Microservices, wie z. B. das YouTube-Video[2], unterbreitet. Die Durchführung der drei Phasen des FMM erzielte für L-mobile gute Resultate. Obwohl es in Phase 1 und Phase 3 Unsicherheiten gab, hat sich herausgestellt, dass das FMM des ISTE ESE für die Migration der Service/Sales-Applikation von L-mobile in Microservices geeignet ist.

[2] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg

Ausblick

Der Ausblick ist in zwei Abschnitte gegliedert. Der erste Abschnitt bespricht die Implikationen und den Ausblick für L-mobile. Der zweite Abschnitt bespricht die Implikationen und den Ausblick für das FMM.

Im Hinblick auf L-mobile wurde durch diesen Proof of Concept bewiesen, dass es möglich ist, die Service/Sales-Applikation von L-mobile in Microservices zu migrieren. Bevor jedoch eine Migration der kompletten Service/Sales-Applikation in Angriff genommen werden kann, gibt es noch offene Fragen, die geklärt werden sollten.
Die weiteren Arbeiten, die für L-mobile durchgeführt werden können, wurden hauptsächlich in dem Review des Proof of Concept vom Head of Development genannt. Hier sollte eine Arbeit durchgeführt werden, mit der herausgefunden werden kann, wie sich die Entwicklung von Microservices in die Entwicklungsstrategie von L-mobile integrieren lässt. Außerdem sollte herausgefunden werden, wie diese Microservices idealerweise von L-mobile betrieben werden können, Container oder IIS, Messaging oder REST API, etc.
Der Product Manager hat angemerkt, dass in weiteren Proof of Concepts herausgefunden werden soll, ob es noch eine weitere Methode für die Service-Identifikation oder Migration gibt, die sich als geeigneter herausstellt. Außerdem sollte ein Leitfaden für die Extraktion idealer Microservices erstellt werden. Mithilfe dieses Leitfadens und der geeignetsten Service-Identifikationsansatz und Migrationsstrategie sollten dann fünf bis sechs verschiedene Microservices unterschiedlicher Komplexitätsklassen extrahiert werden, um eine Aufwandsabschätzung für die komplette Migration der Service/Sales-Applikation anzugehen.
Für diese Arbeiten bei L-mobile kann dieser Proof of Concept als Grundlage dienen.

Im Hinblick auf das FMM wurde in dieser Arbeit gezeigt, dass es sich für die Migration der Service/Sales-Applikation von L-mobile als geeignet erwiesen hat. Dennoch gibt es auch hier weitere Arbeiten, die durchgeführt werden sollten. Es können die Verbesserungsvorschläge aus dieser Arbeit umgesetzt werden. Weitere Arbeiten, die für das FMM durchgeführt werden können, wäre eine Migration der kompletten Service/Sales-Applikation von L-mobile. Weiterhin sollte das FMM auf andere Monolithen angewendet werden, um herauszufinden, ob es auch für diese Applikationen geeignete Service-Identifikationsansätze und Migrationsstrategien vorschlagen kann. Es können nach weiteren Migrationsstrategien in das FMM eingebunden und die Filterungsmechanismen für die Methoden sowie die Patterns und Best Practices verfeinert werden.

Literaturverzeichnis

[ACC+21] W. K. Assunção, T. E. Colanzi, L. Carvalho, J. A. Pereira, A. Garcia, M. J. de Lima, C. Lucena. „A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study". In: *2021 IEEE International conference on software analysis, evolution and reengineering (SANER)*. IEEE. 2021, S. 377–387 (zitiert auf S. 59, 61).

[ACC+22] W. K. Assunção, T. E. Colanzi, L. Carvalho, A. Garcia, J. A. Pereira, M. J. de Lima, C. Lucena. „Analysis of a many-objective optimization approach for identifying microservices from legacy systems". In: *Empirical Software Engineering* 27.2 (2022), S. 51 (zitiert auf S. 59, 61).

[AI16] M. Ahmadvand, A. Ibrahim. „Requirements reconciliation for scalable and secure microservice (de) composition". In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE. 2016, S. 68–73 (zitiert auf S. 59).

[AIE19] M. Abdullah, W. Iqbal, A. Erradi. „Unsupervised learning approach for web application auto-decomposition into microservices". In: *Journal of Systems and Software* 151 (2019), S. 243–257 (zitiert auf S. 59, 61).

[ALFT21] F. Auer, V. Lenarduzzi, M. Felderer, D. Taibi. „From monolithic systems to Microservices: An assessment framework". In: *Information and Software Technology* 137 (2021), S. 106600 (zitiert auf S. 33, 48, 52, 119).

[AM21] O. Al-Debagy, P. Martinek. „A microservice decomposition method through using distributed representation of source code". In: *Scalable Computing: Practice and Experience* 22.1 (2021), S. 39–52 (zitiert auf S. 59–62, 115).

[AM22] O. Al-Debagy, P. Martinek. „Dependencies-based microservices decomposition method". In: *International Journal of Computers and Applications* 44.9 (2022), S. 814–821 (zitiert auf S. 59, 61).

[AO22] K. Akkaya, T. Ovatman. „A Comparative Study of Meta-Data-Based Microservice Extraction Tools". In: *International Journal of Service Science, Management, Engineering, and Technology (IJSSMET)* 13.1 (2022), S. 1–26 (zitiert auf S. 63).

[ASM+21] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G. El Boussaidi, G. Hecht, J. Privat, Y.-G. Guéhéneuc. „A taxonomy of service identification approaches for legacy software systems modernization". In: *Journal of Systems and Software* 173 (2021), S. 110868 (zitiert auf S. 19, 56).

[AWS23a] AWS. *AWS Microservice Extractor for .NET*. https://docs.aws.amazon.com/microservice-extractor/. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 38, 114).

[AWS23b] AWS. *AWS Microservice Extractor for .NET*. https://docs.aws.amazon.com/microservice-extractor/latest/userguide/what-is-microservice-extractor.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 38, 114).

[BBGG20] D. Bajaj, U. Bharti, A. Goel, S. Gupta. „Partial migration for re-architecting a cloud native monolithic application into microservices and faas". In: *Information, Communication and Computing Technology: 5th International Conference, ICICCT 2020, New Delhi, India, May 9, 2020, Revised Selected Papers*. Springer. 2020, S. 111–124 (zitiert auf S. 59, 61).

[BBGG21] D. Bajaj, U. Bharti, A. Goel, S. C. Gupta. „A prescriptive model for migration to microservices based on SDLC artifacts". In: *Journal of Web Engineering* (2021), S. 817–852 (zitiert auf S. 29, 30).

[BBM18] M. Borges, E. Barros, P. H. Maia. „Cloud restriction solver: A refactoring-based approach to migrate applications to the cloud". In: *Information and Software Technology* 95 (2018), S. 346–365 (zitiert auf S. 59, 61).

[BCS21] M. Brito, J. Cunha, J. Saraiva. „Identification of microservices from monolithic applications through topic modelling". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021, S. 1409–1418 (zitiert auf S. 59, 60).

[BGD17] L. Baresi, M. Garriga, A. De Renzis. „Microservices identification through interface analysis". In: *Service-Oriented and Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings 6*. Springer. 2017, S. 19–33 (zitiert auf S. 60, 63).

[BM20] M. H. G. Barbosa, P. H. M. Maia. „Towards identifying microservice candidates from business rules implemented in stored procedures". In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2020, S. 41–48 (zitiert auf S. 59).

[BOP22] G. Blinowski, A. Ojdowska, A. Przybyłek. „Monolithic vs. microservice architecture: A performance and scalability evaluation". In: *IEEE Access* 10 (2022), S. 20357–20374 (zitiert auf S. 24, 26).

[BRW+08] A. Brooks, M. Roper, M. Wood, J. W. Daly, J. Miller. *Replication's Role in Software Engineering*. 2008 (zitiert auf S. 44).

[BSG20] A. Bucchiarone, K. Soysal, C. Guidi. „A model-driven approach towards automatic migration to microservices". In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6–8, 2019, Revised Selected Papers 2*. Springer. 2020, S. 15–36 (zitiert auf S. 60).

[CCRZ20] M. Camilli, C. Colarusso, B. Russo, E. Zimeo. „Domain metric driven decomposition of data-intensive applications“. In: *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2020, S. 189–196 (zitiert auf S. 59, 61).

[CGC+20a] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. Assunção, M. J. Lima, B. Fonseca, M. Ribeiro, C. Lucena. „Search-based many-criteria identification of microservices from legacy systems“. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 2020, S. 305–306 (zitiert auf S. 59, 61).

[CGC+20b] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. Assunção, J. A. Pereira, B. Fonseca, M. Ribeiro, M. J. de Lima, C. Lucena. „On the performance and adoption of search-based microservice identification with tomicroservices“. In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, S. 569–580 (zitiert auf S. 59–62, 115).

[CLL17] R. Chen, S. Li, Z. Li. „From monolith to microservices: A dataflow-driven approach“. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2017, S. 466–475 (zitiert auf S. 59).

[CM22] R. Capuano, H. Muccini. „A Systematic Literature Review on Migration to Microservices: a Quality Attributes perspective“. In: *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2022, S. 120–123 (zitiert auf S. 19, 29).

[CPF+19] B. Cartaxo, G. Pinto, B. Fonseca, M. Ribeiro, P. Pinheiro, M. T. Baldassarre, S. Soares. „Software engineering research community viewpoints on rapid reviews“. In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2019, S. 1–12 (zitiert auf S. 39).

[CPS18] B. Cartaxo, G. Pinto, S. Soares. „The role of rapid reviews in supporting decision-making in software engineering practice“. In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. 2018, S. 24–34 (zitiert auf S. 39).

[CPS20] B. Cartaxo, G. Pinto, S. Soares. „Rapid reviews in software engineering“. In: *Contemporary Empirical Methods in Software Engineering* (2020), S. 357–384 (zitiert auf S. 39, 40).

[DB22] H. Dinh-Tuan, F. Beierle. „MS2M: A message-based approach for live stateful microservices migration“. In: *2022 5th Conference on Cloud and Internet of Things (CIoT)*. IEEE. 2022, S. 100–107 (zitiert auf S. 59, 61).

[DBFP18] A. A. C. De Alwis, A. Barros, C. Fidge, A. Polyvyanyy. „Discovering microservices in enterprise systems using a business object containment heuristic“. In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part II*. Springer. 2018, S. 60–79 (zitiert auf S. 59–63, 115).

[DBFP21] A. A. C. De Alwis, A. Barros, C. Fidge, A. Polyvyanyy. „Microservice remodularisation of monolithic enterprise systems for embedding in industrial IoT networks“. In: *Advanced Information Systems Engineering: 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28–July 2, 2021, Proceedings*. Springer. 2021, S. 432–448 (zitiert auf S. 59–61, 63, 64).

[DBPF18] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, C. Fidge. „Function-splitting heuristics for discovery of microservices in enterprise systems“. In: *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*. Springer. 2018, S. 37–53 (zitiert auf S. 63).

[DBT21] U. Desai, S. Bandyopadhyay, S. Tamilselvam. „Graph neural network to dilute outliers for refactoring monolith application“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Bd. 35. 1. 2021, S. 72–80 (zitiert auf S. 59–63, 115).

[De 18] A. A. C. De Alwis. *Subtype*. https://github.com/AnuruddhaDeAlwis/Subtype. [Online; zugegriffen 13-August-2023]. 2018 (zitiert auf S. 63).

[De 19] L. De Lauretis. „From monolithic architecture to microservices architecture“. In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, S. 93–96 (zitiert auf S. 59).

[DEF+21] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, A. El Fazziki. „A multi-model based microservices identification approach“. In: *Journal of Systems Architecture* 118 (2021), S. 102200 (zitiert auf S. 59).

[DKTT22] M. Dehghani, S. Kolahdouz-Rahimi, M. Tisi, D. Tamzalit. „Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning“. In: *Software and Systems Modeling* 21.3 (2022), S. 1115–1133 (zitiert auf S. 59, 61).

[DLM18] P. Di Francesco, P. Lago, I. Malavolta. „Migrating towards microservice architectures: an industrial survey“. In: *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2018, S. 29–2909 (zitiert auf S. 19).

[DMF+20] M. Daoud, A. E. Mezouari, N. Faci, D. Benslimane, Z. Maamar, A. E. Fazziki. „Automatic microservices identification from a set of business processes“. In: *Smart Applications and Data Analysis: Third International Conference, SADASC 2020, Marrakesh, Morocco, June 25–26, 2020, Proceedings 3*. Springer. 2020, S. 299–315 (zitiert auf S. 59).

[DN02] L. Dobrica, E. Niemela. „A survey on software architecture analysis methods“. In: *IEEE Transactions on software Engineering* 28.7 (2002), S. 638–653 (zitiert auf S. 48).

[Don09] M. G. Donoff. „Field notes: assisting achievement and documenting competence“. In: *Canadian Family Physician* 55.12 (2009), S. 1260–1262 (zitiert auf S. 41, 43).

[EB18] S. Eski, F. Buzluca. „An automatic extraction approach: Transition to microservices architecture from monolithic application“. In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*. 2018, S. 1–6 (zitiert auf S. 59, 61).

[ECA+16] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, R. Casallas. „Towards the understanding and evolution of monolithic applications as microservices“. In: *2016 XLII Latin American computing conference (CLEI)*. IEEE. 2016, S. 1–11 (zitiert auf S. 59, 61).

[FBH+22] J. Fritzsch, J. Bogner, M. Haug, S. Wagner, A. Zimmermann. „Towards an architecture-centric methodology for migrating to microservices“. In: *arXiv preprint arXiv:2207.00507* (2022) (zitiert auf S. 19, 20, 30, 33–35, 41, 42, 47, 56, 66, 74, 82, 114).

[FBWZ19] J. Fritzsch, J. Bogner, S. Wagner, A. Zimmermann. „Microservices migration in industry: intentions, strategies, and challenges“. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2019, S. 481–490 (zitiert auf S. 19, 32, 110, 116).

[FBZW19] J. Fritzsch, J. Bogner, A. Zimmermann, S. Wagner. „From monolith to microservices: A classification of refactoring approaches“. In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1*. Springer. 2019, S. 128–141 (zitiert auf S. 19, 30, 32, 56, 57, 60).

[FFC21] F. Freitas, A. Ferreira, J. Cunha. „Refactoring java monoliths into executable microservice-based applications“. In: *Proceedings of the 25th Brazilian Symposium on Programming Languages*. 2021, S. 100–107 (zitiert auf S. 60).

[FFZ+17] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, A. Barros. „Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency“. In: *Ieee Software* 35.3 (2017), S. 63–72 (zitiert auf S. 31, 32, 116, 119).

[FGPS22] D. Faustino, N. Gonçalves, M. Portela, A. R. Silva. „Stepwise migration of a monolith to a microservices architecture: Performance and migration effort evaluation“. In: *arXiv preprint arXiv:2201.07226* (2022) (zitiert auf S. 25).

[FL14] M. Fowler, J. Lewis. *Microservices a definition of this new architectural term*. https://martinfowler.com/articles/microservices.html. [Online; zugegriffen 13-August-2023]. März 2014 (zitiert auf S. 19, 23, 24, 26–28, 31, 32, 40).

[FM17] C.-Y. Fan, S.-P. Ma. „Migrating monolithic mobile application to microservice architecture: An experiment report“. In: *2017 ieee international conference on ai & mobile services (aims)*. IEEE. 2017, S. 109–112 (zitiert auf S. 19).

[Fow04] M. Fowler. *StranglerFigApplication*. https://martinfowler.com/bliki/StranglerFigApplication.html. [Online; zugegriffen 13-August-2023]. Juni 2004 (zitiert auf S. 30, 31, 78, 80).

[Fow15] M. Fowler. *MicroservicePremium*. https://martinfowler.com/bliki/MicroservicePremium.html. [Online; zugegriffen 13-August-2023]. Mai 2015 (zitiert auf S. 24, 33, 40).

[FPR+17] F. A. Fontana, I. Pigazzini, R. Roveda, D. Tamburri, M. Zanoni, E. Di Nitto. „Arcan: A tool for architectural smells detection". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017, S. 282–285 (zitiert auf S. 36).

[FSC+21] A. F. A. Freire, A. F. Sampaio, L. H. L. Carvalho, O. Medeiros, N. C. Mendonça. „Migrating production monolithic systems to microservices using aspect oriented programming". In: *Software: Practice and Experience* 51.6 (2021), S. 1280–1307 (zitiert auf S. 19, 59, 61).

[GCD+17a] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, A. Di Salle. „Microart: A software architecture recovery tool for maintaining microservice-based systems". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017, S. 298–302 (zitiert auf S. 63).

[GCD+17b] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, A. Di Salle. „Towards recovering the software architecture of microservice-based systems". In: *2017 IEEE International conference on software architecture workshops (ICSAW)*. IEEE. 2017, S. 46–53 (zitiert auf S. 63).

[GCL+20] M. Gao, M. Chen, A. Liu, W. H. Ip, K. L. Yung. „Optimization of microservice composition based on artificial immune algorithm considering fuzziness and user preference". In: *IEEE access* 8 (2020), S. 26385–26404 (zitiert auf S. 59).

[GDD05] B. Graaf, H. van Dijk, A. van Deursen. „Evaluating an embedded software reference architecture-industrial experience report". In: *Ninth European Conference on Software Maintenance and Reengineering*. IEEE. 2005, S. 354–363 (zitiert auf S. 49).

[GFSP21] N. Gonçalves, D. Faustino, A. R. Silva, M. Portela. „Monolith modularization towards microservices: Refactoring and performance trade-offs". In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2021, S. 1–8 (zitiert auf S. 25).

[GKGZ16] M. Gysel, L. Kölbener, W. Giersche, O. Zimmermann. „Service cutter: A systematic approach to service decomposition". In: *Service-Oriented and Cloud Computing: 5th IFIP WG 2.14 European Conference, ESOCC 2016, Vienna, Austria, September 5-7, 2016, Proceedings 5*. Springer. 2016, S. 185–200 (zitiert auf S. 60, 63, 64).

[Gra17] G. Granchelli. *MicroART*. https://github.com/microart/microART-Tool. [Online; zugegriffen 13-August-2023]. 2017 (zitiert auf S. 63).

[GW05] R. Gimnich, A. Winter. „Workflows der Software-Migration". In: *Softwaretechnik-Trends* 25.2 (2005), S. 22–24 (zitiert auf S. 28, 29).

[GZ20] K. Gos, W. Zabierowski. „The comparison of microservice and monolithic architecture“. In: *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE. 2020, S. 150–153 (zitiert auf S. 23, 24, 26, 27, 32).

[HAB17] S. Hassan, N. Ali, R. Bahsoon. „Microservice ambients: An architectural metamodelling approach for microservice granularity“. In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, S. 1–10 (zitiert auf S. 19, 59, 61).

[Hal22a] T. Haller. „Design, implementation and evaluation of an application for guiding architectural refactoring to microservices“. Magisterarb. 2022 (zitiert auf S. 33).

[Hal22b] T. Haller. *GitHub repository of the Architecture Refactoring Helper*. https://github.com/T-Haller/architecture-refactoring-helper. [Online; zugegriffen 13-August-2023]. 2022 (zitiert auf S. 33).

[HLT18] S. Habibullah, X. Liu, Z. Tan. „An approach to evolving legacy enterprise system to microservice-based architecture through feature-driven evolution rules“. In: *International Journal of Computer Theory and Engineering* 10.5 (2018) (zitiert auf S. 59–62, 115).

[HRJ+04] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, S. Krieghoff. „The dublo architecture pattern for smooth migration of business information systems: An experience report“. In: *Proceedings. 26th International Conference on Software Engineering*. IEEE. 2004, S. 117–126 (zitiert auf S. 29).

[IBM20] IBM. *Mono2Micro*. https://www.ibm.com/cloud/blog/announcements/ibm-mono2micro. [Online; zugegriffen 13-August-2023]. 2020 (zitiert auf S. 38, 114).

[IHO02] M. T. Ionita, D. K. Hammer, H. Obbink. „Scenario-based software architecture evaluation methods: An overview“. In: *Workshop on methods and techniques for software architecture review and assessment at the international conference on software engineering*. Citeseer. 2002, S. 1–12 (zitiert auf S. 48).

[Ins23] U. S. Institute of Software Engineering Empirical Software Engineering. *Framework für Microservices-Migrationen*. http://193.196.55.99:9000/home. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 20, 33, 42, 53, 114, 159).

[IT21] N. Ivanov, A. Tasheva. „A Hot Decomposition Procedure: Operational Monolith System to Microservices“. In: *2021 International Conference Automatics and Informatics (ICAI)*. IEEE. 2021, S. 182–187 (zitiert auf S. 59, 61).

[Jic79] T. D. Jick. „Mixing qualitative and quantitative methods: Triangulation in action“. In: *Administrative science quarterly* 24.4 (1979), S. 602–611 (zitiert auf S. 43).

[JLC+19] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, Q. Zheng. „Service candidate identification from monolithic systems based on execution traces“. In: *IEEE Transactions on Software Engineering* 47.5 (2019), S. 987–1007 (zitiert auf S. 36, 59, 61).

[JSK91] C. M. Judd, E. R. Smith, L. Kidder. „Research Methods in Social Relations“. In: *Fort Worth, TX* (1991) (zitiert auf S. 43).

[KBAR21] L. J. Kirby, E. Boerstra, Z. J. Anderson, J. Rubin. „Weighing the evidence: On relationship types in microservice extraction“. In: *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE. 2021, S. 358–368 (zitiert auf S. 60, 63).

[KBAW94] R. Kazman, L. Bass, G. Abowd, M. Webb. „SAAM: A method for analyzing the properties of software architectures“. In: *Proceedings of 16th International Conference on Software Engineering*. IEEE. 1994, S. 81–90 (zitiert auf S. 33, 48, 119).

[KBB+09] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman. „Systematic literature reviews in software engineering–a systematic literature review“. In: *Information and software technology* 51.1 (2009), S. 7–15 (zitiert auf S. 39).

[KDJ04] B. A. Kitchenham, T. Dyba, M. Jorgensen. „Evidence-based software engineering“. In: *Proceedings. 26th International Conference on Software Engineering*. IEEE. 2004, S. 273–281 (zitiert auf S. 39).

[Kha09] S. H. Khandkar. „Open coding“. In: *University of Calgary* 23.2009 (2009) (zitiert auf S. 101).

[Kit04] B. Kitchenham. „Procedures for performing systematic reviews“. In: *Keele, UK, Keele University* 33.2004 (2004), S. 1–26 (zitiert auf S. 40).

[KKC+12] S. Khangura, K. Konnyu, R. Cushman, J. Grimshaw, D. Moher. „Evidence summaries: the evolution of a rapid review approach“. In: *Systematic reviews* 1.1 (2012), S. 1–9 (zitiert auf S. 39).

[KKC00] R. Kazman, M. Klein, P. Clements. *ATAM: Method for architecture evaluation*. Techn. Ber. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2000 (zitiert auf S. 33, 48, 49, 52, 53, 119, 159).

[Klo22] S. Klock. *MicADO*. https://github.com/AMUSEResearch/MicADO. [Online; zugegriffen 13-August-2023]. 2022 (zitiert auf S. 63).

[KMK22] J. Kazanavičius, D. Mažeika, D. Kalibatienė. „An Approach to Migrate a Monolith Database into Multi-Model Polyglot Persistence Based on Microservice Architecture: A Case Study for Mainframe Database“. In: *Applied Sciences* 12.12 (2022), S. 6189 (zitiert auf S. 59).

[KMM18] M. Kalske, N. Mäkitalo, T. Mikkonen. „Challenges when moving from monolith to microservice architecture“. In: *Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17*. Springer. 2018, S. 32–47 (zitiert auf S. 19, 23, 24, 26, 27, 32).

[Koc22] D. Koch. „Migrating monolithic architectures to microservices: a study on software quality attributes“. Magisterarb. 2022 (zitiert auf S. 33, 40, 53, 56, 57, 66, 71, 159).

[Kol23] M. Kolny. *Scaling up the Prime Video audio/video monitoring service and reducing costs by 90% The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.* https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 33).

[KVGJ17] S. Klock, J. M. E. Van Der Werf, J. P. Guelen, S. Jansen. „Workload-based clustering of coherent feature sets in microservice architectures“. In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, S. 11–20 (zitiert auf S. 59, 61, 63).

[KXK+21] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, D. Banerjee. „Mono2Micro: a practical and effective tool for decomposing monolithic Java applications to microservices“. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2021, S. 1214–1224 (zitiert auf S. 38, 60, 114).

[KZH+20] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, D. Kröger. „Microservice decomposition via static and dynamic analysis of the monolith“. In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2020, S. 9–16 (zitiert auf S. 60, 62–64, 75, 76, 78, 114, 115, 117, 120).

[LG85] Y. S. Lincoln, E. G. Guba. *Naturalistic inquiry*. sage, 1985 (zitiert auf S. 44).

[LML20] C.-Y. Li, S.-P. Ma, T.-W. Lu. „Microservice migration using strangler fig pattern: A case study on the green button system“. In: *2020 International Computer Symposium (ICS)*. IEEE. 2020, S. 519–524 (zitiert auf S. 30, 59, 60, 62–66, 75, 78, 80, 82, 83, 88, 114, 115, 120).

[LO20] J. Löhnertz, A.-M. Oprescu. „Steinmetz: Toward Automatic Decomposition of Monolithic Software Into Microservices.“ In: *SATToSE*. 2020 (zitiert auf S. 59, 61, 63, 64).

[LW22] R. Lichtenthäler, G. Wirtz. „Towards a Quality Model for Cloud-native Applications“. In: *Service-Oriented and Cloud Computing: 9th IFIP WG 6.12 European Conference, ESOCC 2022, Wittenberg, Germany, March 22–24, 2022, Proceedings*. Springer. 2022, S. 109–117 (zitiert auf S. 71).

[LXR+22] B. Liu, J. Xiong, Q. Ren, S. Tyszberowicz, Z. Yang. „Log2MS: a framework for automated refactoring monolith into microservices using execution logs“. In: *2022 IEEE International Conference on Web Services (ICWS)*. IEEE. 2022, S. 391–396 (zitiert auf S. 19, 30, 35, 36, 113).

[MB07] P. Montgomery, P. H. Bailey. „Field notes and theoretical memos in grounded theory“. In: *Western Journal of Nursing Research* 29.1 (2007), S. 65–79 (zitiert auf S. 41, 43).

[MBDT21] A. Mathai, S. Bandyopadhyay, U. Desai, S. Tamilselvam. „Monolith to microservices: Representing application software through heterogeneous gnn“. In: *arXiv preprint arXiv:2112.01317* (2021) (zitiert auf S. 60, 63, 64).

[MBMR21] N. C. Mendonça, C. Box, C. Manolache, L. Ryan. „The monolith strikes back: Why istio migrated from microservices to a monolithic architecture“. In: *IEEE Software* 38.05 (2021), S. 17–22 (zitiert auf S. 33).

[MCF+20] T. Matias, F. F. Correia, J. Fritzsch, J. Bogner, H. S. Ferreira, A. Restivo. „Determining microservice boundaries: a case study using static and dynamic software analysis“. In: *Software Architecture: 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings 14*. Springer. 2020, S. 315–332 (zitiert auf S. 60, 62).

[MCL17] G. Mazlami, J. Cito, P. Leitner. „Extraction of microservices from monolithic software architectures“. In: *2017 IEEE International Conference on Web Services (ICWS)*. IEEE. 2017, S. 524–531 (zitiert auf S. 19, 50, 60, 62).

[MH94] M. B. Miles, A. M. Huberman. *Qualitative data analysis: An expanded sourcebook*. sage, 1994 (zitiert auf S. 43, 101).

[New20] S. Newman. *Vom Monolithen zu Microservices: Patterns, um bestehende Systeme Schritt für Schritt umzugestalten*. O'Reilly, 2020 (zitiert auf S. 24, 25).

[NSR19] L. Nunes, N. Santos, A. Rito Silva. „From a monolith to a microservices architecture: An approach based on transactional contexts“. In: *Software Architecture: 13th European Conference, ECSA 2019, Paris, France, September 9–13, 2019, Proceedings 13*. Springer. 2019, S. 37–52 (zitiert auf S. 60).

[NUEH18] R. Nakazawa, T. Ueda, M. Enoki, H. Horii. „Visualization tool for designing microservices with the monolith-first approach“. In: *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE. 2018, S. 32–42 (zitiert auf S. 59–64, 115).

[PAM19] I. Pigazzini, F. Arcelli Fontana, A. Maggioni. „Tool support for the migration to microservice architecture: An industrial case study“. In: *Software Architecture: 13th European Conference, ECSA 2019, Paris, France, September 9–13, 2019, Proceedings 13*. Springer. 2019, S. 247–263 (zitiert auf S. 36, 37, 60, 113).

[Par72] D. L. Parnas. „On the criteria to be used in decomposing systems into modules“. In: *Communications of the ACM* 15.12 (1972), S. 1053–1058 (zitiert auf S. 50).

[PKY20] J. Park, D. Kim, K. Yeom. „An approach for reconstructing applications to develop container-based microservices“. In: *Mobile Information Systems* 2020 (2020), S. 1–23 (zitiert auf S. 59).

[PL18] J. Phillippi, J. Lauderdale. „A guide to field notes for qualitative research: Context and conversation“. In: *Qualitative health research* 28.3 (2018), S. 381–388 (zitiert auf S. 41, 43).

[PMA19] F. Ponce, G. Márquez, H. Astudillo. „Migrating from monolithic architecture to microservices: A Rapid Review“. In: *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE. 2019, S. 1–7 (zitiert auf S. 19, 23, 24, 26, 27, 29–32, 116, 119).

[PP18] K. Perera, I. Perera. „A rule-based system for automated generation of serverless-microservices architecture“. In: *2018 IEEE International Systems Engineering Symposium (ISSE)*. IEEE. 2018, S. 1–8 (zitiert auf S. 59).

[Pra22] M. Pratt. *Introduction to Spring Cloud Load Balancer*. https://www.baeldung.com/spring-cloud-load-balancer. [Online; zugegriffen 13-August-2023]. Juni 2022 (zitiert auf S. 80).

[PS22] S. Pulnil, T. Senivongse. „A Microservices Quality Model Based on Microservices Anti-patterns“. In: *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE. 2022, S. 1–6 (zitiert auf S. 71).

[Ric23a] C. Richardson. *Access token*. https://microservices.io/patterns/security/access-token.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80, 81).

[Ric23b] C. Richardson. *API Gateway*. https://microservices.io/patterns/apigateway.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80, 81).

[Ric23c] C. Richardson. *Circuit Breaker*. https://microservices.io/patterns/reliability/circuit-breaker.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80, 81).

[Ric23d] C. Richardson. *Externalized configuration*. https://microservices.io/patterns/externalized-configuration.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80).

[Ric23e] C. Richardson. *Health Check API*. https://microservices.io/patterns/observability/health-check-api.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80).

[Ric23f] C. Richardson. *Log Aggregation*. https://microservices.io/patterns/observability/application-logging.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80).

[Ric23g] C. Richardson. *Messaging*. https://microservices.io/patterns/communication-style/messaging.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 78, 80, 81).

[Ric23h] C. Richardson. *Microservice chassis*. https://microservices.io/patterns/microservice-chassis.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80, 81).

[Ric23i] C. Richardson. *Microservices.io*. https://microservices.io/. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 66).

[Ric23j] C. Richardson. *Multiple service instances per host*. https://microservices.io/patterns/deployment/multiple-services-per-host.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80).

[Ric23k] C. Richardson. *Rest API*. https://microservices.io/patterns/communication-style/rpi.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 78, 81).

[Ric23l] C. Richardson. *Server-side service discovery*. https://microservices.io/patterns/server-side-discovery.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 81).

[Ric23m] C. Richardson. *Service instance per container*. https://microservices.io/patterns/deployment/service-per-container.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 81).

[Ric23n] C. Richardson. *service Registry*. https://microservices.io/patterns/service-registry.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80, 81).

[Ric23o] C. Richardson. *Service Template*. https://microservices.io/patterns/service-template.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 80, 81).

[Ric23p] C. Richardson. *Shared Database Pattern*. https://microservices.io/patterns/data/shared-database.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 77, 78).

[Ric23q] C. Richardson. *Strangler Application*. https://microservices.io/patterns/refactoring/strangler-application.html. [Online; zugegriffen 13-August-2023]. 2023 (zitiert auf S. 78, 80).

[RR22] V. Raj, S. Ravichandra. „A service graph based extraction of microservices from monolith services of service-oriented architecture“. In: *Software: Practice and Experience* 52.7 (2022), S. 1661–1678 (zitiert auf S. 59).

[RWW+18] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, T. Huang. „Migrating web applications from monolithic structure to microservices architecture“. In: *Proceedings of the 10th Asia-Pacific Symposium on Internetware*. 2018, S. 1–10 (zitiert auf S. 59, 61).

[SB21] T. Schirgi, E. Brenner. „Quality assurance for microservice architectures“. In: *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE. 2021, S. 76–80 (zitiert auf S. 71).

[Sea08] C. B. Seaman. „Qualitative Methods“. In: *Guide to Advanced Empirical Software Engineering*. Hrsg. von F. Shull, J. Singer, D. I. K. Sjøberg. London: Springer London, 2008, S. 35–62. ISBN: 978-1-84800-044-5. DOI: 10.1007/978-1-84800-044-5_2. URL: https://doi.org/10.1007/978-1-84800-044-5_2 (zitiert auf S. 41, 43–45, 101, 119).

[Sea99] C. B. Seaman. „Qualitative methods in empirical studies of software engineering“. In: *IEEE Transactions on software engineering* 25.4 (1999), S. 557–572 (zitiert auf S. 101).

[Seb+17] S. Sebastian et al. „Transform monolith into microservices using docker“. In: *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. IEEE. 2017, S. 1–5 (zitiert auf S. 26, 29, 30, 32).

[Set81] R. Sethi. „A model of concurrent database transactions“. In: *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*. IEEE. 1981, S. 175–184 (zitiert auf S. 24).

[SK17] A. Shashwat, D. Kumar. „A service identification model for service oriented architecture“. In: *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*. IEEE. 2017, S. 1–5 (zitiert auf S. 59–63, 115).

[SM07] M. Svahnberg, F. Mårtensson. „Six years of evaluating software architectures in student projects“. In: *Journal of Systems and Software* 80.11 (2007), S. 1893–1901 (zitiert auf S. 42, 48, 49, 52, 53, 119).

[SP21] A. Santos, H. Paula. „Microservice decomposition and evaluation using dependency graph and silhouette coefficient“. In: *15th Brazilian Symposium on Software Components, Architectures, and Reuse*. 2021, S. 51–60 (zitiert auf S. 60).

[SSB+18] A. Selmadji, A.-D. Seriai, H. L. Bouziane, C. Dony, R. O. Mahamane. „Re-architecting oo software into microservices: A quality-centred approach“. In: *Service-Oriented and Cloud Computing: 7th IFIP WG 2.14 European Conference, ESOCC 2018, Como, Italy, September 12-14, 2018, Proceedings 7*. Springer. 2018, S. 65–73 (zitiert auf S. 30).

[SSB+20] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. O. Mahamane, P. Zaragoza, C. Dony. „From monolithic architecture style to microservice one based on a semi-automatic approach“. In: *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2020, S. 157–168 (zitiert auf S. 30).

[SSO22] K. Sellami, M. A. Saied, A. Ouni. „A hierarchical dbscan method for extracting microservices from monolithic applications“. In: *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*. 2022, S. 201–210 (zitiert auf S. 60).

[SSR19] N. Singhal, U. Sakthivel, P. Raj. „Efficient Hybrid Research for QoS-Aware Mcroservice Compostion“. In: *International Journal of Recent Technology and Engineering* 8.2 (2019), S. 5251–5255 (zitiert auf S. 59).

[Sta15] G. Starke. *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. Carl Hanser Verlag GmbH Co KG, 2015 (zitiert auf S. 50).

[TAA+22] I. Trabelsi, M. Abdellatif, A. Abubaker, N. Moha, S. Mosser, S. Ebrahimi-Kahou, Y.-G. Guéhéneuc. „From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis“. In: *Journal of Software: Evolution and Process* (2022), e2503 (zitiert auf S. 60, 63).

[TAZ+15] A. C. Tricco, J. Antony, W. Zarin, L. Strifler, M. Ghassemi, J. Ivory, L. Perrier, B. Hutton, D. Moher, S. E. Straus. „A scoping review of rapid review methods“. In: *BMC medicine* 13.1 (2015), S. 1–15 (zitiert auf S. 39).

[THLL18] S. Tyszberowicz, R. Heinrich, B. Liu, Z. Liu. „Identifying microservices using functional decomposition“. In: *Dependable Software Engineering. Theories, Tools, and Applications: 4th International Symposium, SETTA 2018, Beijing, China, September 4-6, 2018, Proceedings 4*. Springer. 2018, S. 50–65 (zitiert auf S. 60).

[Thö15] J. Thönes. „Microservices“. In: *IEEE software* 32.1 (2015), S. 116–116 (zitiert auf S. 26, 32).

[TLP17] D. Taibi, V. Lenarduzzi, C. Pahl. „Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation“. In: *IEEE Cloud Computing* 4.5 (2017), S. 22–32 (zitiert auf S. 84).

[TLP18] D. Taibi, V. Lenarduzzi, C. Pahl. „Architectural patterns for microservices: a systematic mapping study“. In: *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. SciTePress. 2018 (zitiert auf S. 66).

[TS19] D. Taibi, K. Systä. „From monolithic systems to microservices: A decomposition framework based on process mining“. In: (2019) (zitiert auf S. 19, 36, 113).

[VF21] V. Velepucha, P. Flores. „Monoliths to microservices-migration problems and challenges: A sms“. In: *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*. IEEE. 2021, S. 135–142 (zitiert auf S. 31, 116).

[VGO+17] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano et al. „Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures“. In: *Service Oriented Computing and Applications* 11 (2017), S. 233–247 (zitiert auf S. 50).

[VLC19] J. A. Valdivia, X. Limón, K. Cortes-Verdin. „Quality attributes in patterns related to microservice architecture: a Systematic Literature Review“. In: *2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE. 2019, S. 181–190 (zitiert auf S. 66, 80).

[VLL+20] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, J. O. Ocharán-Hernández. „Patterns related to microservice architecture: a multivocal literature review“. In: *Programming and Computer Software* 46 (2020), S. 594–608 (zitiert auf S. 66, 80).

[VMK22] E. Volynsky, M. Mehmed, S. Krusche. „Architect: A Framework for the Migration to Microservices“. In: *2022 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE. 2022, S. 71–76 (zitiert auf S. 37, 38, 113).

[VPAG21] F. H. Vera-Rivera, E. Puerto, H. Astudillo, C. M. Gaona. „Microservices Backlog–A Genetic Programming Technique for Identification and Evaluation of Microservices From User Stories“. In: *IEEE Access* 9 (2021), S. 117178–117203 (zitiert auf S. 59).

[WYPZ20] Y. Wei, Y. Yu, M. Pan, T. Zhang. „A Feature Table approach to decomposing monolithic applications into microservices“. In: *Proceedings of the 12th Asia-Pacific Symposium on Internetware*. 2020, S. 21–30 (zitiert auf S. 60, 62, 63).

[YM20] J. W. Yoder, P. Merson. „Strangler patterns“. In: *Proceedings of the 27th Conference on Pattern Languages of Programs*. 2020, S. 1–25 (zitiert auf S. 30, 31, 78, 80–82).

[ZLD+20] Y. Zhang, B. Liu, L. Dai, K. Chen, X. Cao. „Automated microservice identification in legacy systems with functional and non-functional metrics“. In: *2020 IEEE international conference on software architecture (ICSA)*. IEEE. 2020, S. 135–145 (zitiert auf S. 30).

[ZSS+21] P. Zaragoza, A.-D. Seriai, A. Seriai, H.-L. Bouziane, A. Shatnawi, M. Derras. „Refactoring Monolithic Object-Oriented Source Code to Materialize Microservice-oriented Architecture.“ In: *ICSOFT* 117 (2021), S. 126 (zitiert auf S. 30, 37, 59–64, 113, 115).

[ZZ21] J. Zhao, K. Zhao. „Applying Microservice Refactoring to Object-2riented Legacy System“. In: *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE. 2021, S. 467–473 (zitiert auf S. 60, 62–64, 115).

Alle URLs wurden zuletzt am 13. 08. 2023 geprüft.

A. Filterungs-Tabellen

Pattern	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
Aggregator	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Ambassador	Ausgefiltert (1)	Trade-off zwischen QAs, SQAs, SPs und erhöter Komlexität nicht akzeptabel.
Anti-Corruption Layer	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität und Trade-Offs inakzeptabel.
Asynchronous messaging	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
API gateway	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Asynchronous completion token	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Asynchronous query	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Auth-service	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Backend for frontend	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität und Trade-Offs inakzeptabel.
Bulkhead	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Change code dependency to	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.

Pattern	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
service call		
Circuit breaker	Angenommen	Da es die Reliability berücksichtigt und keine Trade-Offs besitzt.
Competing consumers	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Compute Resource Consolidation	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Consumer-driven contracts	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Container	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Correlation ID	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
CQRS	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
DB Cluster	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität. und Trade-Offs inakzeptabel.
DB is the service	Ausgefiltert (2)	Wegen Applikations-Kontext.
DB per Service	Ausgefiltert (2)	Wegen Applikations-Kontext.
Deploy cluster and orchestrate containers	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Edge server	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Enable continous integration	Angenommen	Da bereits vorhanden.
Event notification	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Event sourcing	Ausgefiltert (2)	Wegen Applikations-Kontext.
External configuration	Ausgefiltert (2)	Wegen Applikations-Kontext.

Pattern	Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen	Begründung
store		
External load balancer	Ausgefiltert (2)	Wegen Applikations-Kontext.
Externalized configuration	Angenommen	Da bereits implementiert und Security berücksichtigt.
Gatekeeper	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Gateway Aggregation	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Gateway Offloading	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Gateway Routing	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Health check	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Internal load balancer	Angenommen	Da es Reliability berücksichtigt und keine Trade-Offs besitzt.
Key-Value Store	Ausgefiltert (2)	Wegen Applikations-Kontext.
Leader Election	Ausgefiltert (1)	Erfüllt keine der gewollten QAs, SQAs oder SPs.
Load balancer/ load-balancing	Angenommen	Da es Reliability berücksichtigt und keine Trade-Offs besitzt.
Local database proxy	Ausgefiltert (2)	Wegen Applikations-Kontext.
Local sharing-based router	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Log aggregator	Angenommen	Durch Empfehlung des Software-Architekten.
Microservice DevOps	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Monitor	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu

Pattern	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
		späterem Zeitpunkt diskutiert.
Mulitple service per host	Zurückgestellt (4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Page cache	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Pipes and filters	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Priority queue	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Request-reaction	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
REST integration	Angenommen	Da Messaging zu einem späteren Zeitpunkt diskutiert wird, wird die Kommunikation für den Proof of Concept in dieser Arbeit über eine REST API realisiert.
Result cache	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Scalable Store	Ausgefiltert (2)	Wegen Applikations-Kontext.
Secure channel	Zurückgestellt (3)	Wird wegen Firmen internen Vorgaben zu späterem Zeitpunkt diskutiert.
Self-contained systems	Ausgefiltert (2)	Wegen Applikations-Kontext.
Self-containment of services	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Service discovery	Angenommen	Da es Security berücksichtigt. und keine Trade-Offs besitzt.
Service locator	Angenommen	Da es Security berücksichtigt. und keine Trade-Offs besitzt.
Service registry	Angenommen	Da es Reliability berücksichtigt. und die Vorteile die Trade-Offs übertreffen.
Service registry + *Message* bus hybrid	Ausgefiltert (1)	Erfüllt keine der gewollten QAs, SQAs oder SPs.

Pattern	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
Service registry *client*	Angenommen	Da es Reliability berücksichtigt. und keine Trade-Offs besitzt.
Shared DB server	Angenommen	Durch Firmen interne Absprache.
Sidecar	Ausgefiltert (1)	Trade-off zwischen QAs, SQAs, SPs und erhöter Komlexität nicht akzeptabel.
Single Service per host	Ausgefiltert (1)	Erfüllt keine der gewollten QAs, SQAs oder SPs.
Static Content Hosting	Ausgefiltert (1)	Trade-off zwischen QAs, SQAs, SPs und erhöter Komlexität nicht akzeptabel.
Strangler	Angenommen	Da nur ein paar Microservice extrahiert werden und der Monolith nebenher betrieben werden soll.
Tolerant reader	Ausgefiltert (1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.

Tabelle A.1.: Die vom FMM vorgeschlagenen Patterns, in welchem Filterungsschritt sie ausgefiltert wurden oder übernommen wurden und eine Begründung, warum die Patterns ausgefiltert wurden.

Best Practice	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
Access restriction	Angenommen	Da bereits implementiert.
Account Separation	Angenommen	Da bereits implementiert.
Acyclic Calls	Zurückgestellt(3)	Wegen Firmeninterne vorgaben zurückgestellt.
API-based communication	Angenommen	Da die Kommunikation im Rahmen dieser Arbeit durch eine API basierte Kommunikation realisiert wird.
Appropriate Service *Relationship*	Ausgefiltert(1)	Erfüllt keine der gewollten QAs, SQAs oder SPs.
Authentication delegation	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Automated infrastructure	Ausgefiltert(2)	Wegen Applikations-Kontext.
Automated monitoring	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Automated restarts	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Autonomous fault *handling*	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Built-in autoscaling	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Cloud vendor abstraction	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.

Best Practice	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
Coarse-Grained Microservices	Angenommen	Da die identifizierten Microservices bereits sehr grob sind.
Communication indirection	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Configuration management	Angenommen	Da es Reliability berücksichtigt und bereits implementiert ist.
Cost variability	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Data encryption in *transit*	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Dynamic scheduling	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Guarded ingress	Angenommen	Da es Reliability berücksichtigt und bereits implementiert ist.
Immutable artifacts	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Infrastructure abstraction	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Isolated state	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Loose Coupling	Angenommen	Vereinfachung des Codes und hohe Wiederverwendbarkeit.
Manageable Connections	Angenommen	Firmen interne Vorgabe.
Manageable Standards	Angenommen	Firmen interne Vorgabe.
Non-ESB Microservices	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Operation outsourcing	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Persistent Communication	Zurückgestellt(3)	Wegen Firmeninterne vorgaben zurückgestellt.
Replication	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs

Best Practice	**Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen**	**Begründung**
		mit niedriger Priorität.
Resolved Endpoints	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Right Cuts	Ausgefiltert(1)	Erfüllt keine der gewollten QAs, SQAs oder SPs.
Seamless upgrades	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Secrets management	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Separate Persistency	Ausgefiltert(1)	Erfüllt keine der gewollten QAs, SQAs oder SPs.
Separation by gateways	Zurückgestellt(4)	Wird wegen Umfang der Implementierung in dieser Arbeit zu späterem Zeitpunkt diskutiert.
Seperate Libraries	Angenommen	Vereinfachung des Codes und hohe Wiederverwendbarkeit.
Service Independence	Angenommen	Vereinfachung des Codes und hohe Wiederverwendbarkeit.
Service-orientation	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Sparsity	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Standardization	Angenommen	Firmen interne Vorgabe.
Standardized deployment unit	Angenommen	Da schon implementiert.
Use infrastructure as code	Ausgefiltert(1)	Erfüllt nur QAs, SQAs oder SPs mit niedriger Priorität.
Versioned APIs	Angenommen	Da der Monolith mit verschiedenen Versionen der Microservices kommunizieren soll.

Best Practice	Ausgefiltert(Schritt)/ Zurückgestellt(Schritt)/ Angenommen	Begründung

Tabelle A.2.: Die vom FMM vorgeschlagenen Best Practices, in welchem Filterungsschritt sie ausgefiltert wurden oder übernommen wurden und eine Begründung, warum die Best Practices ausgefiltert wurden.

B. Listings

```
44  public void ConfigureServices(IServiceCollection services)
45  {
46     serviceSettings = configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
47     communicationSettings = configuration.GetSection(nameof(CommunicationSettings))
            .Get<CommunicationSettings>();
48     services.AddSingleton<IPdfService, PdfService>();
49     services.AddControllers();
50     services.AddTransient<PdfService>();
51  }
```

Listing B.1: Die Intitailisierung der Konfigurationsdateien, des PdfServices und der Controller in der Startup Klasse des PdfGeneration-Microservice.

```
 6  namespace Crm.PdfGeneration.Controllers
 7  {
 8     [ApiController]
 9     [Route("api/Pdf")]
10     public class PdfController : ControllerBase
11     {
12        private readonly IPdfService pdfService;
13        public PdfController(IPdfService pdfService)
14        {
15              this.pdfService = pdfService;
16        }
17        [HttpGet("Html2Pdf/{dto}")]
18        public PdfDTO GetHtml2Pdf([FromBody] SendPdfDTO dto)
19        {
20           var pdf = this.pdfService.Html2Pdf(dto.html, dto.headerMargin, dto.footerMargin);
21              var responseDTO = new PdfDTO(Guid.NewGuid(), pdf);
22              return responseDTO;
23        }
24     }
25   }
```

Listing B.2: Die Get-Methode für den Aufruf der Html2Pdf Methode des PdfGeneration-Microservice.

```
35  namespace Crm.PdfGeneration.AutoFac
36  {
37     public class PdfModule : Module
38     {
39        protected override void Load(ContainerBuilder builder)
40        {
41           base.Load(builder);
42           builder.Register(c => c.Resolve<ISiteService>().CurrentSite).As<Site>();
43           builder.RegisterType<CefToPdfConverter>().As<ICefToPdfConverter>();
44           builder.Register(
45              c =>
46              {
47              var appSettingsProvider = c.Resolve<IAppSettingsProvider>();
48              var cefToPdfPath = appSettingsProvider.GetValue(ServiceSettings.CefToPdfPath);
49              var fileName = Path.Combine(cefToPdfPath, "CefToPdf.exe");
50                    return (IDeployment)new StaticDeployment(cefToPdfPath);
51              }).As<IDeployment>();
52        }
53     }
54  }
```

Listing B.3: Die *PdfModule.cs* Klasse des PdfGeneration-Microservice.

```
 1  namespace Crm.PdfGeneration.Services
 2  {
 3     public class PdfService : IPdfService
 4     {
 5        protected readonly ILog logger;
 6        protected readonly ICefToPdfConverter converter;
 7        private Site site;
 8        public PdfService(Site site, ILog logger, ICefToPdfConverter converter)
 9        {
10           this.site = site;
11           this.logger = logger;
12           this.converter = converter;
13        }
14        public virtual byte[] Html2Pdf(string html, double headerMargin = 0, double footerMargin = 0)
15        {
16           var result = converter.HtmlToPdf(html, new CefToPdfSettings
17           {
18                         MarginTop = headerMargin,
19                         MarginBottom = footerMargin
20           });
21           return result;
22        }
23     }
24  }
```

Listing B.4: Die Html2Pdf Methode des PdfService im PdfGenerierungs-Microservice.

```
12  namespace Crm.Library.Services
13  {
14     public class PdfServiceClient
15     {
16        private readonly HttpClient httpClient;
17        public PdfServiceClient(HttpClient httpClient)
18        {
19              this.httpClient = httpClient;
20        }
21        public async Task<PdfDTO> GetPdfAsync(string html, double headerMargin, double footerMargin)
22        {
23              var dto = new SendPdfDTO(html, headerMargin, footerMargin);
24           var json = JsonConvert.SerializeObject(dto);
25           var uri = new Uri(httpClient.BaseAddress.ToString() + $"api/Pdf/Html2Pdf/dto");
26              var request = new HttpRequestMessage
27              {
28              Method = HttpMethod.Get,
29              RequestUri = uri,
30              Content = new StringContent(json, Encoding.UTF8, "application/json")
31              };
32              var response = await httpClient.SendAsync(request);
33              var pdf = await response.Content.ReadFromJsonAsync<PdfDTO>();
34              return pdf;
35        }
36     }
37  }
```

Listing B.5: Der Aufruf der API für die Html2Pdf Methode des PdfGeneration-Microservice in der *PdfServiceClient.cs* Klasse des *Crm.Library* Projekts.

```
 1  namespace Crm.Library.Services
 2  {
 3     public class PdfService : IPdfService
 4     {
 5        protected readonly ILog logger;
 6        private readonly PdfServiceClient pdfServiceClient;
 7        public PdfService(ILog logger, PdfServiceClient pdfServiceClient)
 8        {
 9              this.logger = logger;
10              this.pdfServiceClient = pdfServiceClient;
11        }
12        public virtual byte[] Html2Pdf(string html, double headerMargin = 0, double footerMargin = 0)
13        {
14           var result = Task.Run(async () => await pdfServiceClient.GetPdfAsync(html, headerMargin,
                  footerMargin));
15           var res = result.Result.content;
16           return res;
17        }
18     }
19  }
```

Listing B.6: Anpassung der Methodenkörper der *PdfService.cs* Klasse in *Crm.Library*.

```
 99  services.AddHttpClient<PdfServiceClient>(client =>
100  {
101     client.BaseAddress = new Uri(configuration.GetValue<string>("PdfServiceAddress"));
102  })
103  .AddTransientHttpErrorPolicy(builder => builder.Or<TimeoutRejectedException>().WaitAndRetryAsync(
104     5,
105     retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt))
106                       + TimeSpan.FromMilliseconds(jitterer.Next(0, 1000)),
107     onRetry: (outcome, timeSpan, retryAttempt) =>
108     {
109        var serviceProvider = services.BuildServiceProvider();
110        serviceProvider.GetService<ILogger<PdfServiceClient>>()?
111        .LogWarning($"Delaying for {timeSpan.TotalSeconds} seconds, then making retry {retryAttempt}");
112     }
113  ))
114  .AddTransientHttpErrorPolicy(builder => builder.Or<TimeoutRejectedException>().CircuitBreakerAsync(
115     3,
116     TimeSpan.FromSeconds(15),
117     onBreak: (outcome, timeSpan) =>
118     {
119        var serviceProvider = services.BuildServiceProvider();
120        serviceProvider.GetService<ILogger<PdfServiceClient>>()?
121        .LogWarning($"Onening the circuit for {timeSpan.TotalSeconds} seconds.");
122     },
123     onReset: () =>
124     {
125        var serviceProvider = services.BuildServiceProvider();
126        serviceProvider.GetService<ILogger<PdfServiceClient>>()?
127        .LogWarning($"Closing the circuit.");
128     }
129  ))
130  .AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));
```

Listing B.7: Prototypische Cricuit Breaker für den PdfServiceClient implementiert in der Startup Klasse des Monolithen.

C. Feldnotizen

Kategorie	Inhalt
Strukturierte Feldnotiz Nr.:	1
Datum:	01.03.23 - 23.03.23
Uhrzeit:	-
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Betreuer von L-mobile, Betreuer von der Uni
Phase des FMM:	1
Schritt:	Planung der Architekturbewertung
Was wurde gemacht:	Es wurde eine Architekturbewertungsmethode ausgewählt, dafür wurden ATAM, SAAM, Auer und ATAM Light analysiert. Es wurde sich dafür entschieden die ATAM Light Methode anzuwenden, da sich diese innerhalb von 4 - 5 Stunden ausführen lässt. Mit dem Betreuer von L-mobile wurde das weitere Vorgehen, wie die wichtigsten Stakeholder kontaktiert werden sollen und die Architekturbewertung geplanen werden soll besprochen. Bei dem Product Manager und dem Head of Development wurde angefragt, ob die ausgewählten Stakeholder für eine Architekturbewertung genehmigt werden. Ein Termin für den Architekturbewertungs -Workshop wurde gesucht und gefunden. Die Stakeholder sind: Der Product Manager, der Head of Development, der Software Architekt und ein Software Engineer. Es wurde eine Präsentation erstellt, die durch den Architektur-bewertungs-Workshop führt und den Prozess dabei erklärt.
Welcher Teil der Software war Betroffen:	-
Was lief gut:	Terminfindung ging schnell. Product Manager und Head of Development waren schnell überzeugt. Eine geeignete Architekturbewertungsmethode zu finden ging gut, mit Hilfe von dem Betreuer von der Uni.

Kategorie	Inhalt
Was lief schlecht:	Die Suche nach Beispielen für eine Szenario basierte Bewertung der Architektur. Es ist mir immer noch ein bisschen unklar, wie mithilfe der Szenarien die Architektur bewertet werden soll.
Empfindungen:	Nervös, weil die Architekturbewertung bald ansteht und ich noch nie eine Architekturbewertung durchgeführt habe. Nervös und unsicher ob ich die Architekturbewertung genehmigt bekomme.
Kommentare:	Für jemanden der zum ersten Mal ein Architektur Review durchführt und noch keine Ahnung hat wie genau die Bewertung mit Szenarien aussieht gibt es wenig Hilfe, auch im Internet. Auch die Kollegen haben noch nie ein Architektur Review durchgeführt.
Sonstiges:	Auf YouTube wurde nach Videos von Beispielen für die Architekturbewertung gesucht. Auch im Internet wurde nach Beispielen gesucht. Vielleicht wäre es gut auf ein oder zwei Beispiele zu verweisen, um den Software Engineer/Architekt besser zu unterstützen, sollte er noch keine Erfahrung mit Architektur Reviews haben.

Tabelle C.1.: F1_P1_230301_ArchitekBewPlanung

Kategorie	Inhalt
Feldnotiz Nr.:	2
Datum:	11.03.23 - 16.03.23
Uhrzeit:	-
Ort:	Sulzbach an der Murr und Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	1
Schritt:	Projekt Beschreibung
Durchgeführte Aktivitäten:	Im Rahmen der Planung der Architekturbewertung wurde das Projekt beschrieben. Es wurde die Masterarbeit, die zu migrierende Software, das Institut, das FMM und die Dauer des Projekts thematisiert.
Betroffener Teil der Software:	-
Was lief gut:	Die Beschreibung war einfach zu formulieren, da die L-mobile Arbeiter sich mit dem System das Migriert werden soll sehr gut auskennen und über meine Masterarbeit Bescheid wissen.
Was lief schlecht:	-
Empfindungen:	Gut vorbereitet, ich kenne mich mit meinem Projekt gut aus. Unsicher, wie genau ich das Projekt in der Ausarbeitung beschreiben soll.
Kommentare:	Vielleicht wäre es hilfreich, wenn im ARH- Tool dieser Schritt ein bisschen genauer beschrieben ist und vielleicht ein Beispiel angegeben wird.
Sonstiges:	-

Tabelle C.2.: F2_P1_230311_ProjBeschr

Kategorie	**Inhalt**
Feldnotiz Nr.:	3
Datum:	30.03.23
Uhrzeit:	11 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Head of Development
Phase des FMM:	1
Schritt:	2
Durchgeführte Aktivitäten:	Die strategischen Ziele von L-mobile mit Bezug auf Microservices wurden von dem Head of Development beschrieben. Als strategische Ziele wurden Modularisierung weiter vorantreiben, damit Teilbereiche einfacher und unabhängiger bearbeitet werden können, bessere Wartbarkeit und Isolation, Kostenreduktion durch Verwendung günstiger Ressourcen, z.B. Docker, Kubernetes, Codebasis auf einen mögliche Microservice betrieb hin zu führen, damit die Migration ohne größere Schwierigkeiten durchgeführt werden kann, genannt.
Betroffener Teil der Software:	-
Was lief gut:	Die strategischen Ziele ließen sich einfach und schnell identifizieren.
Was lief schlecht:	-
Empfindungen:	-
Kommentare:	Es ist noch unsicher, ob alle strategischen Ziele genannt wurden oder ob noch welche nachgeliefert werden.
Sonstiges:	-

Tabelle C.3.: F3_P1_230330_StratGoals

Kategorie	Inhalt
Feldnotiz Nr.:	4
Datum:	13.03.23
Uhrzeit:	7:15 - 15:45
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	Migrationsstrategie, Service Identifikation, Best Practices und Patterns Analyse
Durchgeführte Aktivitäten:	Alle bis zu diesem Tag im ARH bereitgestellten Methoden wurden gelesen und schon eine erste grobe Priorisierung der Strategien vorgenommen.
Betroffener Teil der Software:	-
Was lief gut:	Die Strategien konnten gut analysiert werden.
Was lief schlecht:	Nicht alle Veröffentlichungen sind ohne Uni VPN verfügbar, bzw. kosten dann Geld.
Empfindungen:	Guten ersten Überblick über die vorhandenen Strategien. Erste Favoriten ausgewählt.
Kommentare:	Informationen zum Strangler Fig Pattern wurden vermisst, ansonsten gab es einen guten ersten Überblick über die vorhandenen Strategien. Informationen über Infrastruktur, Verwaltung auf dem Server, Monitoring, Continuous Deployment, Technologien?
Sonstiges:	Über die Patterns des Strangler Fig Patterns habe ich mich separat informiert, hier sollten noch Referenzen eingefügt werden, da diese Methode wahrscheinlich zum Einsatz kommt, da man die Applikation in Microservices migrieren kann während sie nebenher noch aktiv im Einsatz ist.

Tabelle C.4.: F4_P2_230313_MethodAnalysis

Kategorie	**Inhalt**
Feldnotiz Nr.:	5
Datum:	23.03.23
Uhrzeit:	13:10 Uhr - 13:40 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Head of Development, Software Architekt, Product Manager, Software Engineer
Phase des FMM:	1
Schritt:	3
Durchgeführte Aktivitäten:	Es wurden, anhand des ISO 25010, neun Kategorien für Qualitätsattribute und 2 Subattribute pro Kategorie, die für die Architektur wichtig sind, identifiziert und priorisiert. Die Ergebnisse wurden in einem Utility Tree dokumentiert.
Betroffener Teil der Software:	Architektur
Was lief gut:	Die Identifikation der Attribute ging schnell und produktiv voran, auch in der Priorisierung haben die Teilnehmer produktiv mitgearbeitet. Es gab oft Diskussionen zwischen den Teilnehmern was die Attribute der Kategorien und ihre Priorität angeht.
Was lief schlecht:	Die Teilnehmer hatten die Migration in Microservices bereits zu sehr im Kopf.
Empfindungen:	Ein bisschen nervös. Unsicher, ob die Teilnehmer die Aufgabe richtig verstanden haben und ob die Resultate hilfreich sein werden.
Kommentare:	Die Teilnehmer haben anfangs schon mehr die Qualitätsattribute identifiziert die für Microservices für sie wichtig waren, nach einer Erklärung des Moderators wurden dann aber Qualitätsattribute identifiziert welche allgemein, für jede Architektur, gelten sollten.
Sonstiges:	-

Tabelle C.5.: F5_P1_230323_IdentifyQualAttr

Kategorie	Inhalt
Feldnotiz Nr.:	6
Datum:	23.03.23
Uhrzeit:	13:40 Uhr - 14:32 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Head of Development, Software Architekt, Product Manager, Software Engineer
Phase des FMM:	1
Schritt:	3
Durchgeführte Aktivitäten:	Basierend auf den drei am höchsten priorisierten Qualitätsattribut Kategorien wurden zwei bewertbare/ messbare Szenarien erhoben und nach ihrer Priorität und technischen Schwierigkeit bewertet. Die Szenarien wurden auch wie in ATAM von Kazman et al. [KKC00] bewertet. Die Bewertung der Szenarien orientierte sich an dem vom FMM und von Daniel Koch vorgeschlagenen Bewertungsschema [Ins23, Koc22], und hatte folgende Form. Für die Wichtigkeit wurden die Buchstaben A, B und C vergeben. Dabei stehen die Buchstaben: A für sehr wichtig, B für Medium wichtig, C für weniger wichtig. Die technische Schwierigkeit wurde mit den Zahlen 1, 2 und 3 bewertet. Dabei steht: 1 für einfach/kein Risiko, 2 für normal, 3 für sehr schwierig/risikoreich. Die erhobenen Szenarien wurden in einer Diskussion mit den vorgestellten Attributen bewertet. Wobei der Redeanteil des Head of Development und des Product Manager bei der Wichtigkeit größer war und bei der technischen Schwierigkeit hatte der Software Architekt und der Software Engineer mehr Redeanteil.
Betroffener Teil der Software:	Architektur
Was lief gut:	Für jedes Attribut wurden zwei bewertbare/messbare Szenarien erhoben und bewertet. Es gab oft Diskussionen zwischen den Teilnehmern was die Szenarien und ihre Bewertung angeht.
Was lief schlecht:	Für manche Attribute fiel es den Teilnehmern schwer

Kategorie	Inhalt
	Szenarien zu definieren. Manchmal wurden messbare Kriterien der Szenarien vergessen und anschließend ergänzt. Für manche Szenarien wurden die messbaren Kriterien etwas schwammig und ungenau beschrieben. Es wurde eher im Stillen überlegt und der Moderator musste die Teilnehmer oft dazu angeregt ihre Gedanken zu teilen.
Empfindungen:	Manche Szenarien habe ich nicht auf Anhieb verstanden und die Teilnehmer waren oft zurückhaltend. Unsicherheit, ob die Szenarien Zielführend sind.
Kommentare:	Die Diskussion war ein bisschen zäh. Von einem Teilnehmer kam das Feedback, dass für das nächste Mal ein online Tool verwendet werden sollte, in dem die Teilnehmer Postits verwenden können um ihre Gedanken aufzuschreiben. Das nächste Mal wird das auf jeden Fall berücksichtigt, da sonst lange Pausen entstehen und der Moderator die Teilnehmer oft dazu anregen muss ihre Gedanken zu teilen. Das nächste Mal sollte eine Strategie zur Bewertung und Priorisierung der Qualitätsattribute und der Szenarien verwendet werden.
Sonstiges:	-

Tabelle C.6.: F6_P1_230323_ElicitScenarios

Kategorie	Inhalt
Feldnotiz Nr.:	7
Datum:	23.03.23
Uhrzeit:	14:47 Uhr - 15:05 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Head of Development, Software Architekt, Product Manager, Software Engineer
Phase des FMM:	1
Schritt:	4
Durchgeführte Aktivitäten:	Die Architektur des aktuellen Systems wurde vom Software Architekten präsentiert. Es wurde ein Überblick über die verwendeten Technologien gegeben und das Client-Server Setup sowie das High Availability Setup vorgestellt. Das CI Setup wurde vorgestellt und es wurde ein Komponentendiagramm gezeigt.
Betroffener Teil der Software:	Architektur
Was lief gut:	Die Präsentation war ausführlich und es gab keine offenen Fragen.
Was lief schlecht:	-
Empfindungen:	Gute und übersichtliche Präsentation der Architektur, zuversichtlich dass sich diese gut analysieren lässt, aber noch etwas unsicher wie man mit den erhobenen Szenarien diese Architektur analysiert.
Kommentare:	Die Übersicht und Präsentation der Architektur wird ins Onboarding der Firma aufgenommen.
Sonstiges:	-

Tabelle C.7.: F7_P1_230323_UnderstandLegacySys

Kategorie	Inhalt
Feldnotiz Nr.:	8
Datum:	23.03.23
Uhrzeit:	15:05 Uhr - 16:30 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Head of Development, Software Architekt, Product Manager, Software Engineer
Phase des FMM:	1
Schritt:	4
Durchgeführte Aktivitäten:	Die bestehende Architektur wurde hinsichtlich der erhobenen Szenarien bewertet und ein Vergleich zu einer Microservices Architektur wurde gemacht. Es wurde für zwei Szenarien entschieden, dass die aktuelle Architektur nicht genügt um diese zu erfüllen. Die restlichen Szenarien werden von der aktuellen Architektur erfüllt. Für jedes Szenario wurde auch besprochen wie dieses in einer Microservices Architektur erreicht werden kann.
Betroffener Teil der Software:	Architektur
Was lief gut:	Für jedes Szenario wurde die Architektur analysiert und entdeckt wie die Architektur dieses Szenario erfüllt, bzw. nicht erfüllt. Es gab gute Diskussionen.
Was lief schlecht:	Die Teilnehmer wussten am Anfang nicht ganz wie sie an diesen Schritt herangehen sollten. Es wurde wieder viel im Stillen überlegt und kaum geredet. Ein Teilnehmer hat in diesem Schritt sehr wenig Beitrag geleistet. Die Teilnehmer mussten oft dazu angeregt werden etwas beizutragen.
Empfindungen:	Unsicherheit wie ich durch diesen Schritt gut leiten kann und wie ich die Ergebnisse verwenden kann.
Kommentare:	Wenig Erfahrung von den Teilnehmern und dem Moderator was die Architekturbewertung angeht. Für die zweite Hälfte der Szenarien wurde schneller drüber gegangen als für die erste. Ein Teilnehmer musste wegen eines Terminkonfliktes die Architekturbewertung in der zweite Hälfte verlassen. Es wurde nach der Hälfte eine kurze Pause eingefügt.
Sonstiges:	-

Tabelle C.8.: F8_P1_230323_ArchitekturBewertung

Kategorie	Inhalt
Feldnotiz Nr.:	9
Datum:	23.03.23
Uhrzeit:	15:05 Uhr - 16:30 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Head of Development, Software Architekt, Product Manager, Software Engineer
Phase des FMM:	1
Schritt:	5
Durchgeführte Aktivitäten:	Die Architekturbewertung wurde zusammengefasst und ein Resultat über die aktuelle Architektur und die Migration in Microservices wurde gefällt. Man ist zu der Entscheidung gekommen, dass die aktuelle Architektur die aktuellen Anforderungen erfüllt und dass eine Migration in Microservices zu dem aktuellen Zeitpunkt noch nicht nötig aber ein „nice to have“ ist, welches sich auf jeden Fall lohnt sich darüber zu informieren und Wissen aufzubauen, da es wahrscheinlich ist dass eine Microservices Architektur in der Zukunft eine Rolle spielt.
Betroffener Teil der Software:	Architektur
Was lief gut:	Die Teilnehmer waren in diesem Schritt kommunikativer und waren alle einer Meinung. Die Meinung der Teilnehmer ist auf jeden Fall begründet.
Was lief schlecht:	-
Empfindungen:	Das Resultat spiegelt auf jeden Fall die Bewertung der Architektur wieder. Ein bisschen noch unsicher, ob man die Bewertung anders hätte durchführen können.
Kommentare:	Eine Teilmigration der Applikation in Microservices wird auf jeden Fall weiterhin in dieser Masterarbeit verfolgt, da sich die Erfahrungen daraus in der Zukunft lohnen können, und da es so bereits mit der Firma und dem Institut abgesprochen wurde. Es würde mich noch interessieren, wie die Architekturbewertung ausgefallen wäre, wenn der ATAM Prozess vollständig durchgeführt worden wäre und Szenarien für alle Qualitätsattribute analysiert worden wären.
Sonstiges:	-

Tabelle C.9.: F9_P1_230323_Conclusion

Kategorie	**Inhalt**
Feldnotiz Nr.:	10
Datum:	11.04.23
Uhrzeit:	16:50 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	1
Schritt:	3
Durchgeführte Aktivitäten:	Die Qualitätsattribute und Szenarien wurden in den ARH, unter Schritt 3 Qualitätsattribute, eingegeben.
Betroffener Teil der Software:	-
Was lief gut:	Die Eingabe der Szenarien und Qualitätsattribute war einfach und die UI dafür ist offensichtlich aufgebaut.
Was lief schlecht:	Neuen Szenarien wurden in der UI am Ende der Liste eingefügt, was viel Zeit zum scrollen verbraucht. Einige Qualitätsattribute (auch aus dem ISO 25010) fehlen in der Vorschlagsliste.
Empfindungen:	Etwas genervt, dass ein neues Szenario am Ende der Liste eingefügt wird anstatt am Anfang, da deswegen immer ans Ende der Liste gescrollt werden muss.
Kommentare:	Die Qualitätsattribute wurden noch nicht final eingegeben. Diese Durchführung war rein zum Testen des ARH gedacht, um zu sehen, wie die weitere Ausführung des ARH vorangeht.
Sonstiges:	Empfehlung die neuen Szenarien in der UI am Anfang der Liste einzufügen, da das die User Experience erhöht und viel gescrolle wegfällt. Mehr Qualitätsattribute hinzufügen, oder zumindest Schaltfläche, mit der neue Qualitätsattribute hinzufügt werden können implementieren.

Tabelle C.10.: F10_P1_230411_SystemComprehension

Kategorie	Inhalt
Feldnotiz Nr.:	11
Datum:	17.04.23
Uhrzeit:	10 Uhr - 17 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	SIA/MA Analyse
Durchgeführte Aktivitäten:	Es wurden viele der 90 vom FMM vorgeschlagenen SIAs/MAs analysiert und nach verschiedenen Kriterien gefiltert. Weitere Tabelle zum Vergleichen der Methoden aufgestellt.
Betroffener Teil der Software:	-
Was lief gut:	SIAs/MAs gut analysiert, Exceltabelle zum Vergleichen aufgestellt. Bereitgestellte Exceltabelle für Vergleich der Methoden hilfreich.
Was lief schlecht:	Manchmal unsicher, ob eine SIA/MA für die Migration der L-mobile Applikation geeignet ist.
Empfindungen:	Manchmal unsicher.
Kommentare:	SIA/MA Ausschlusskriterien: Neu, MDA, WDA, nur für Java, wenige der erhobenen QAs werden berücksichtigt, Workflowspecific. SIA/MA Einschlusskriterien: Refactor, einige der erhobenen QAs werden berücksichtigt, SCA, einige MDA, DMC, SCA Hybrid.
Sonstiges:	Vielleicht können noch mehr Kriterien in die bereits bestehende Excel (und später in das ARH) aufgenommen werden, die für die Auswahl einer bestimmten SIA/MA wichtig sind, z. B. Input der vorgeschlagenen Methode, da nicht immer alle Dokumente/Modelle etc. vorhanden sind.

Tabelle C.11.: F11_P2_230417_SIAReading

Kategorie	**Inhalt**
Feldnotiz Nr.:	12
Datum:	18.04.23
Uhrzeit:	8:30 Uhr - 17 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	SIA/MA analysieren
Durchgeführte Aktivitäten:	Die vorgeschlagenen SIAs/MAs wurden weiter analysiert.
Betroffener Teil der Software:	-
Was lief gut:	SIAs/MAs wurden gut analysiert, es konnten gute Entscheidungen getroffen werden, ob ein SIA/MA weiterhin betrachtet werden sollte. Es wurden weitere Punkte mit den bestehenden Veröffentlichungen entdeckt, die in der Ausarbeitung angepasst werden können.
Was lief schlecht:	Unkonzentriert
Empfindungen:	Analyse läuft besser als gestern, sichereres Gefühl, aber unkonzentrierter und mehr Stress/Druck.
Kommentare:	-
Sonstiges:	Es wird als gut empfunden, dass das FMM verschiedene SIAs, MAs, Best Practises und Patterns vorstellt und nicht nur einen spezifischen Ansatz wie die anderen Frameworks.

Tabelle C.12.: F12_P2_230418_SIAReading

Kategorie	Inhalt
Feldnotiz Nr.:	13
Datum:	19.04.23
Uhrzeit:	11:30 Uhr - 15:00 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	SIA/MA analysieren
Durchgeführte Aktivitäten:	Die vorgeschlagenen SIAs/MAs wurden weiter analysiert.
Betroffener Teil der Software:	-
Was lief gut:	SIAs/MAs wurden gut analysiert, es konnten gute Entscheidungen getroffen werden, ob ein SIA/MA weiterhin betrachtet werden sollte. Es wurden weitere Punkte mit den bestehenden Veröffentlichungen entdeckt, die in der Ausarbeitung angepasst werden können.
Was lief schlecht:	-
Empfindungen:	-
Kommentare:	-
Sonstiges:	-

Tabelle C.13.: F13_P2_230419_SIAReading

Kategorie	**Inhalt**
Feldnotiz Nr.:	14
Datum:	20.04.23
Uhrzeit:	7:30 Uhr - 15:00 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	SIA/MA mit Qualitätsattributen von L-mobile analysieren.
Durchgeführte Aktivitäten:	Die vorgeschlagenen SIAs/MAs wurden mit den erstellten Tabellen analysiert. Dafür wurden 3 Listen erstellt. In der ersten Liste stehen die SIAs/MAs, welche für die Qualitätsattribute von L-mobile und die erste Vorauswahl (keine MDAs, keine WDAs, keine Neuentwicklung, keine Methode nur Java) geeignet waren. In der zweiten Liste stehen SIAs/MAs, welche auf die Qualitätsattribute von L-mobile und eine weniger strengere Filterung nach der Vorauswahl (keine Neuentwicklung, keine Methode nur Java) geeignet waren. In der dritten Liste stehen SIAs/MAs, welche nach einer Einschätzung von mir am geeignetsten sind. (Gefilterte (ohne reine MDA) SIAs/MAs mit Qualitätsattributen ausgewählt: (4),5,6,10,11,12,(14),19,26,33,37,38,40,58,59,65,67,74,89,90) SIAs/MAs mit Qualitätsattributen ausgewählt: 4(3),5,6(2),9,10(2),11(3),12(2),13(2),14,15(3),16,17,19(2),24(2), 26(3),30,3 1,33(3),35,36(3),37(3),38,40(2),41,43(2),46(4),47(3), 48,54,57(2),58(2),59,60(3),65,66(2),67(2),73,74,75(2),78(2), 79(2),89(6),90 Nach zweiter Überprüfung: SIAs/MAs mit QAs ausgewählt: 4(4),5,6(2),9,10(2),11(3),12(2),13(2),14(2),15(3),16,17,19(2), 24(2),26(3),3 0,31,33(3),35,36(4),37(3),38,40(2),41,43(3),46(4), 47(3),48,54(2),57(2),58(2),59,60(3),65,66(2),67(2),73,74,75(2), 78(2),79(2),89(7),90 Gefilterte(ohne reine MDA) SIAs/MAs mit QAs ausgewählt: 4(4),9,10(2),11(3),12(2),13(2),15(3),16,17,19(2),26(3),30,33(3), 37(3), 38,4 6(4),48,57(2),58(2),59,60(3),65,67(2),74,75(2),79(2), 89(7),90 Erhöhung Filter: 9(+),11,12,13,14(+),15,16(+),17,20(+),22,24(+),27,29,31(+),33(+), 34,37,3 8,44,45(+),47,49,51(+),52(+),56(+),58(+),59,61(+),66,67, 68,71(+),72,73(+),74,76,79

Kategorie	Inhalt
	Verringerung Filter: 14(-),16(-),21(-),24(-),25(-),33(-),45(-),52(-),55(-),56(-),58(-),60(-), 61(-),7 1(-),73(-),87(-) Gefilterte(ohne reine MDA) SIAs/MAs mit QAs und SPs: 9,16,58(2),33(3),60(3) Von Filterung als geeignet eingeschätzte SIAs/MAs: 9,16,58(2),33(3),60(3),40,89 Von mir als geeignet eingeschätzte SIAs/MAs: 4,6,10,11,12,17,19,26,30,33,38,39,40,46,48,58,59,65,67,74,89,90
Betroffener Teil der Software:	-
Was lief gut:	SIAs/MAs wurden gut analysiert, es konnten gute Entscheidungen getroffen werden, ob ein SIA/MA weiterhin betrachtet werden sollte.
Was lief schlecht:	Bei der Sortierung der SPs (System Eigenschaften) ist mir am Anfang ein Fehler unterlaufen, da in der Tabelle nach + increase und - decrease aufgeteilt sind, ich habe die SPs jedoch nach positiv und negativ beeinflusst aufgeteilt. Hier muss man beachten bei welchem SP + bedeutet dass es positiv bzw. negativ beeinflusst wird, bzw. wo - bedeutet dass es positiv bzw. negativ beeinflusst wird.
Empfindungen:	-
Kommentare:	Die Liste mit den vorhandenen SIAs/MAs ist sehr übersichtlich, gut strukturiert und gut aufgeteilt. Eine Filterung mit dieser Liste ist sehr gut.
Sonstiges:	-

Tabelle C.14.: F14_P2_230420_SIAReading

Kategorie	**Inhalt**
Feldnotiz Nr.:	15
Datum:	27.04.23
Uhrzeit:	8 Uhr bis 16 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	Analyse der Tools für die Service-Identifikation.
Durchgeführte Aktivitäten:	Es wurden die Tools nach der Anwendbarkeit auf die L-mobile Applikation gefiltert. Alle Tools: 2,3,15,20,21,22,25,26,27,31,32,39,50,51,65,71,72,76, 81,91,92,93 Ohne rein für Java Tools: 2,15,21,22,25,26,27,32,39,65,71,72,76,81,91,92,93 Ohne nicht erfolgreich, vom Uni Betreuer, getestete Tools: 2,15,21,22,25,26,27,32,39,65,71,76,81,93 Nur Quellcode als Input: 22,25,26,27,32,39,65,76,93 Weitere Filterung Resultat: 26, 32, 39, 65, 93
Betroffener Teil der Software:	-
Was lief gut:	-
Was lief schlecht:	-
Empfindungen:	-
Kommentare:	Manchmal stand nicht in der Exceltabelle, ob ein Tool nur für Java Applikationen implementiert wurde.
Sonstiges:	-

Tabelle C.15.: F15_P2_230427_SIAToolReading

Kategorie	Inhalt
Feldnotiz Nr.:	16
Datum:	01.05.23
Uhrzeit:	11 Uhr - 16 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	2
Schritt:	SIAs/MAs filtern
Durchgeführte Aktivitäten:	Die von der bisherigen Filterung übrig gebliebenen SIAs/MAs wurden eingehender analysiert, um mit jeweils 2-3 Publikationen eine Service- Identifizierung und einen Migrationsansatz für die Arbeit zu definieren. Von mir als geeignet eingeschätzte SIAs/MAs: 9,16,58(2),33(3),60(3),40,89,65,26,39,27,92. Von mir ausgewählte SIAs/MAs Hybride: 39,58,16,92 Reine SIAs: - Reine MAs: 40
Betroffener Teil der Software:	-
Was lief gut:	-
Was lief schlecht:	-
Empfindungen:	Unsicherheit darüber wie genau ich diese Restlichen SIAs/MAs weiter filtern kann.
Kommentare:	L-mobil spezifisches Problem: viele Tools sind auf Java ausgelegt und es gibt hauptsächlich Beispiele mit Java.
Sonstiges:	Vielleicht noch weitere Kriterien und Beispiele anfügen, mit denen die SIAs/MAs effizienter gefiltert werden können und damit der Anwender nicht so unsicher ist wie genau er den Rest filtern soll.

Tabelle C.16.: F16_P2_230501_SIAFiltering

Kategorie	**Inhalt**
Feldnotiz Nr.:	17
Datum:	05.05.23
Uhrzeit:	7 Uhr bis 15:30 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Testen der Tools aus den ausgewählten Publikationen.
Durchgeführte Aktivitäten:	Es wurden die Tools, welche in den ausgewählten Publikationen vorgestellt werden ausgeführt und getestet.
Betroffener Teil der Software:	Architektur, kompletter Quellcode.
Was lief gut:	Manche Tools waren einfach in Betrieb zu nehmen.
Was lief schlecht:	Andere Tools waren ein bisschen schwerer zu testen, da oft nicht offensichtlich war wie man diese ausführt. Es ist noch unklar, wie mit diesen Tools eine MSA erstellt werden soll.
Empfindungen:	Unsicher
Kommentare:	Kritik bezieht sich nicht auf das FMM.
Sonstiges:	Vielleicht wäre es hilfreich Hilfestellungen für die vorgeschlagenen Tools anzubieten, wobei das wahrscheinlich ein sehr großer Aufwand wäre.

Tabelle C.17.: F17_P3_230505_TestedTools

Kategorie	Inhalt
Feldnotiz Nr.:	18
Datum:	11.05.23 - 15.05.23
Uhrzeit:	-
Ort:	Daheim und Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Suche nach Tools für die Service Identifizierung.
Durchgeführte Aktivitäten:	Es wurde in der zur Verfügung gestellten Exceltabelle und im Internet nach weiteren Serviceidentifikations-Tools gesucht, welche sich interessant anhörten und sich auf die .Net Applikation anwenden lassen.
Betroffener Teil der Software:	-
Was lief gut:	Viele weitere Veröffentlichungen haben einen interessanten Ansatz.
Was lief schlecht:	Die Tools aus den Veröffentlichungen sind entweder nur für Java Applikationen implementiert oder schlecht dokumentiert. Es wurden keine weiteren Tools für die Service- Identifikation gefunden.
Empfindungen:	-
Kommentare:	-
Sonstiges:	-

Tabelle C.18.: F18_P3_230511_ErweiterteToolAnalyse

Kategorie	**Inhalt**
Feldnotiz Nr.:	19
Datum:	16.05.23
Uhrzeit:	9 Uhr - 10 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel, Head of Development
Phase des FMM:	3
Schritt:	Präsentation der Identifizierten Services
Durchgeführte Aktivitäten:	Die Bounded Contexts und damit die identifizierten Services wurden präsentiert. Weitere architektonische Entscheidungen wie die Datenbankteilung, Technologien, Monorepository, Messaging und REST API Kommunikation wurden besprochen.
Betroffener Teil der Software:	Architektur
Was lief gut:	Die Services sind sehr gut identifiziert, die Kontexte müssen nicht mehr angepasst werden.
Was lief schlecht:	-
Empfindungen:	Der Head of Development scheint überzeugt von den Fortschritten der Arbeit zu sein.
Kommentare:	-
Sonstiges:	-

Tabelle C.19.: F19_P3_230516_IdentifizierteServices

Kategorie	Inhalt
Feldnotiz Nr.:	20
Datum:	16.05.23
Uhrzeit:	10 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Erstellung einer Architektur mithilfe der vom FMM vorgeschlagenen Methode (Methode 39).
Durchgeführte Aktivitäten:	Es wurde eine MSA mithilfe der vorgeschlagenen Mitteln der Methode 39 (Structure 101 und Analyse der Bounded Contexts) erstellt.
Betroffener Teil der Software:	Architektur, MSA
Was lief gut:	Die monolithische Architektur der Applikation ist bereits sehr gut in ihre Bounded Contexts aufgeteilt, so dass diese Analyse recht einfach war.
Was lief schlecht:	Exploreviz konnte nicht zur Ausführung gebracht werden, da die angebotene Version immer einen Fehler ausgibt wenn ein neues Projekt hinzugefügt werden soll, eine online Version ist nur für Java einsatzbereit.
Empfindungen:	Unsicher, ob die Bounded Contexts Analyse mit Structure 101 ausreichend für eine akzeptable MSA ist.
Kommentare:	Viele der Tools sind nur für Java Applikationen implementiert, oder benötigen eine bestimmte Form an Daten oder sie sind so schlecht dokumentiert, so dass sie nicht ausgeführt werden können.
Sonstiges:	-

Tabelle C.20.: F20_P3_230516_MSAGenerierung

Kategorie	**Inhalt**
Feldnotiz Nr.:	21
Datum:	17.05.23
Uhrzeit:	10 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel, Head of Development
Phase des FMM:	3
Schritt:	Besprechung der generierten Architektur.
Durchgeführte Aktivitäten:	Die von Structure 101 und den Bounded Contexts einer Domain Analysis erstellte Architektur wurde auf ihre Akzeptanz hin untersucht.
Betroffener Teil der Software:	Architektur, MSA
Was lief gut:	Die monolithische Architektur der Applikation ist bereits sehr gut in ihre Bounded Contexts aufgeteilt, so dass diese Analyse recht einfach war. Die MSA wird mit wenigen Anpassungen akzeptiert.
Was lief schlecht:	-
Empfindungen:	Ich denke, dass der Head of Development die vorgeschlagene Architektur gut findet, Services können in späteren Iterationen weiter verfeinert werden falls nötig.
Kommentare:	-
Sonstiges:	-

Tabelle C.21.: F21_P3_230517_MSAEvaluation

Kategorie	Inhalt
Feldnotiz Nr.:	22
Datum:	19.05.23
Uhrzeit:	9:30 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel, Product Manager
Phase des FMM:	3
Schritt:	Besprechung der generierten Architektur.
Durchgeführte Aktivitäten:	Die von Structure 101 und den Bounded Contexts einer Domain Analysis erstellte Architektur wurde auf ihre Akzeptanz hin untersucht. Es wurden Vorschläge zum weiteren vorgehen gemacht, wie z. B. die Aufsplittung der DB für jeden Service einzeln zu entscheiden. Erstmal einfache Services zu implementieren und dann später evtl. verfeinern. Dass Technologien für den Proof of Concept mir überlassen werden und ich z. B. mit Docker und Kubernetes arbeiten soll. Es wurden weitere Möglichkeiten für Microservices vorgeschlagen. Es wurden Vorschläge für die Implementierungsreihenfolge gemacht.
Betroffener Teil der Software:	Architektur, MSA
Was lief gut:	Die monolithische Architektur der Applikation ist bereits sehr gut in ihre Bounded Contexts aufgeteilt, so dass diese Analyse recht einfach war. Die MSA wurde akzeptiert.
Was lief schlecht:	-
Empfindungen:	Ich denke, dass der Product Manager die vorgeschlagene Architektur gut findet, Services können in späteren Iterationen weiter verfeinert werden falls nötig.
Kommentare:	-
Sonstiges:	-

Tabelle C.22.: F22_P3_230519_MSAEvaluation2

Kategorie	**Inhalt**
Feldnotiz Nr.:	23
Datum:	23.05.23 - 27.05.23
Uhrzeit:	-
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Architektur Erzeugung
Durchgeführte Aktivitäten:	Es wurde eine Architektur erstellt, welche die Implementierung der MSA beschreibt, welche Patterns und sonstige Architekturentscheidungen getroffen wurden und welche noch zur Diskussion ausstehen. Es wurden Diagramme für die Übersicht generiert und die identifizierten Services präsentiert.
Betroffener Teil der Software:	MSA
Was lief gut:	Die Architektur lies sich einfach und schnell generieren, da viele Dinge wie die Services und Technologien bereits mit weiteren Stakeholdern diskutiert und abgenommen wurde.
Was lief schlecht:	-
Empfindungen:	Gutes Gefühl die Architektur endlich erstellt zu haben. Unsicher, ob die generierte Architektur ausreicht, da ich noch keine Erfahrung mit der Erstellung einer Architektur habe.
Kommentare:	-
Sonstiges:	Durch die existierenden Artefakte bei L-mobile sowie durch die generierte Service- Identifikation und die Empfehlungen des FMM habe ich das Gefühl eine solide Grundlage für die Architektur zu haben. Trotzdem fühle ich mich etwas unsicher, was die Architektur angeht. Werden die beschriebenen Patterns akzeptiert und reicht die MSA aus?

Tabelle C.23.: F23_P3_230523_ArchitekturErstellen

Kategorie	Inhalt
Feldnotiz Nr.:	24
Datum:	24.05.23
Uhrzeit:	14:15 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Pattern Filterung
Durchgeführte Aktivitäten:	Mithilfe der vom FMM zur Verfügung gestellten Excel-tabelle wurden die Patterns untersucht. Es wurden Patterns herausgefiltert, welche die von Lmobile identifizierten QAs, SQAs und SPs nicht ausreichend erfüllten. Bei erster Filterung rausgeflogen: Anti-Corruption Layer, Backend for frontend, Change code dependency to service call, Competing consumers, Compute Resource Consolidation, Consumer-driven contracts, Correlation ID, DB Cluster, Gateway Aggregation, Health check, Leader Election, Local sharing-based router, Log aggregator, Microservice DevOps, Monitor, Mulitple service per host, Page cache, Priority queue, Requestreaction, REST integration, Result cache, Self-containment of services, Service registry + Message bus hybrid, Shared DB server, Single Service per host, Static Content Hosting, Strangler, Tolerant reader, Aggregator, Ambassador, Enable continous integration, Sidecar. Bei erster Filterung verblieben: Asynchronous messaging, API gateway, Asynchronous completion token, Asynchronous query, Auth-service, Bulkhead, Circuit breaker, Container, CQRS, DB is the service, DB per Service, Deploy cluster and orchestrate containers, Edge server, Event notification, Event sourcing, External configuration store, External load balancer, Externalized configuration, Gatekeeper, Gateway Offloading, Gateway Routing, Internal load balancer, Key-Value Store, Load balancer/load-balancing, Local database proxy, Pipes and filters, Scalable Store, Secure channel, Self-contained systems, Service discovery, Service locator, Service registry, Service registry client. Die in der ersten Filterung ausgefilterten Patterns erfüllten entweder keine der gewollten QAs, SQAs und SPs, oder

Kategorie	Inhalt
	erfüllten nur solche QAs, SQAs und SPs die eine niedrige Priorität haben, oder der Trade-off zwischen QAs, SQAs und SPs und erhöhter Komlexität war nicht akzeptabel. In der zweiten Filterung wurden alle Patterns im Kontext der Service/Sales-Applikation von L-mobile betrachtet, dabei kam es vor, dass bereits herausfilterte Patterns erneut betrachtet wurden. Bei zweiter Filterung rausgeflogen: DB is the service, DB per Service, Event sourcing, External configuration store, External load balancer, Key-Value Store, Local database proxy, Scalable Store, Self-contained systems. Bei zweiter Filterung verblieben: Asynchronous messaging, API gateway, Asynchronous completion token, Asynchronous query, Auth-service, Bulkhead, Circuit breaker, Container, CQRS, Deploy cluster and orchestrate containers, Edge server, Event notification, Externalized configuration, Gatekeeper, Gateway Offloading, Gateway Routing, Internal load balancer, Load balancer/load-balancing, Pipes and filters, Secure channel, Service discovery, Service locator, Service registry, Service registry client. Bei zweiter Filterung wieder betrachtet: Correlation ID, Health check, Monitor, Mulitple service per host, REST integration, Shared DB server, Strangler, Aggregator, Enable continous integration. Da Firmeninterne vorgaben bestehen, welche besagen, dass Messaging und Containerization zu einem späteren Zeitpunkt diskutiert werden sollen gab es eine dritte Filterung, welche diese Patterns herausfiltert, damit sie zu einem späteren Zeitpunkt betrachtet werden können. Bei dritter Filterung für später zurückgestellt: Asynchronous messaging, Asynchronous completion token, Asynchronous query, Container, Deploy cluster and orchestrate containers, Event notification, Pipes and filters, Secure channel. Bei dritter Filterung verblieben: API gateway, Auth-service, Bulkhead, Circuit breaker, CQRS, Edge server, Externalized configuration, Gatekeeper, Gateway Offloading, Gateway Routing, Internal load

Kategorie	Inhalt
	balancer, Load balancer/load-balancing, Service discovery, Service locator, Service registry, Service registry client, Correlation ID, Health check, Monitor, Mulitple service per host, REST integration, Shared DB server, Strangler, Aggregator, Enable continous integration. Da das System, welches in dieser Arbeit implementiert wird, noch keine hohe Komplexität hat, können weitere Patterns für die spätere Implementierung zurückgestellt werden. Bei vierten Filterung für später zurückgestellt: API gateway, Auth-service, Bulkhead, CQRS, Edge server, Gatekeeper, Gateway Offloading, Gateway Routing, Correlation ID, Health check, Monitor, Mulitple service per host, Aggregator Bei vierten Filterung verblieben: Circuit breaker, Externalized configuration, Internal load balancer, Load balancer/load-balancing, Service discovery, Service locator, Service registry, Service registry client/server, REST integration, Shared DB server, Strangler, Enable continous integration. Nach Review des Architekten kam raus, dass der Log Aggregator auch verwendet werden soll.
Betroffener Teil der Software:	-
Was lief gut:	Patterns konnten gut gefiltert werden.
Was lief schlecht:	Beschreibung der Patterns war oft unklar und musste recherchiert werden. Manchmal war die Überlegung zwischen QAs, SQAs, SPs und Trade-offs schwer.
Empfindungen:	Unsicherheiten über die Filterung und ob alle Patterns die verblieben sind wirklich relevant sind und ob relevante Patterns ausgefiltert wurden.
Kommentare:	Es kommt mir so vor, dass bei der reinen Betrachtung der QAs, SQAs und SPs zu viele Patterns übrig bleiben. Viele der bei der ersten Filterung verbliebenen Patterns scheinen sinnvoll und hilfreich zu sein. Wenige Patterns die bei der ersten Filterung rausgeflogen sind scheinen doch noch wichtig zu werden, z. B. Correlation ID, Health check, Monitor, Mulitple service per host, REST integration, Shared DB server, Strangler, Aggregator, Enable continous integration.

Kategorie	Inhalt
	Ein paar der verblieben Patterns scheinen unnötig zu sein, deswegen gab es eine zweite Filterung. Es gibt weitere Patterns, wie z. B. Microservice Chassis und Microservice Template etc. (Microservice.io) welche weiter betrachtet werden. Bei der Filterung wurde sich nur auf die drei am höchsten Priorisierten QAs von L-mobile konzentriert (Security, Functional completeness, Reliability), da sonst jedes Pattern zutreffen würde. Es ist nicht ausgeschlossen, dass in diesem Schritt ausgefilterte Patterns zu einem späteren Zeitpunkt wichtig werden und implementiert wird.
Sonstiges:	Ein klarerer Leitfaden für die Filterung und eine bessere Beschreibung der Patterns wäre von Vorteil. Es gibt noch weitere Patterns, welche vom FMM bisher noch nicht berücksichtigt werden.

Tabelle C.24.: F24_P3_230524_PatternsAnalyse

Kategorie	Inhalt
Feldnotiz Nr.:	25
Datum:	27.05.23
Uhrzeit:	11:30 Uhr
Ort:	Daheim
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Filterung der Best Practices
Durchgeführte Aktivitäten:	Mithilfe der vom FMM zur Verfügung gestellten Excel-tabelle wurden die Best Practices untersucht. Es wurden Best Practices herausgefiltert, welche die von L-mobile identifizierten QAs, SQAs und SPs nicht ausreichend erfüllten. Bei erster Filterung rausgeflogen: API-based communication, Appropriate Service Relationship, Built-in autoscaling, Cloud vendor abstraction, Coarse-Grained Microservices, Communication indirection, Cost variability, Immutable artifacts, Infrastructure abstraction, Isolated state, Loose Coupling, Manageable Connections, Manageable Standards, Non-ESB MSs, Operation outsourcing, Replication, Resolved Endpoints, Right Cuts, Separate Persistency, Seperate Libraries, Service Independence, Service-orientation, Sparsity, Standardization, Standardized deployment unit, Use infrastructure as code, Versioned APIs. Bei erster Filterung verblieben: Access restriction, Account Separation, Acyclic Calls, Authentication delegation, Automated infrastructure, Automated monitoring, Automated restarts, Autonomous fault handling, Configuration management, Data encryption in transit, Dynamic scheduling, Guarded ingress, Persistent Communication, Seamless upgrades, Secrets management, Separation by gateways. Die in der ersten Filterung ausgefilterten Best Practices erfüllten entweder keine der gewollten QAs, SQAs und SPs, oder erfüllten nur solche QAs, SQAs und SPs die eine niedrige Priorität haben. In der zweiten Filterung wurden alle Best Practices im Kontext der Service/Sales-Applikation von L-mobile betrachtet, dabei kam es vor, dass bereits herausfilterte Best Practices erneut betrachtet wurden. Bei zweiter Filterung rausgeflogen:

Kategorie	Inhalt
	Automated infrastructure. Bei zweiter Filterung verblieben: Access restriction, Account Separation, Acyclic Calls, Authentication delegation, Automated monitoring, Automated restarts, Autonomous fault handling, Configuration management, Data encryption in transit, Dynamic scheduling, Guarded ingress, Persistent Communication, Seamless upgrades, Secrets management, Separation by gateways. Bei zweiter Filterung wieder betrachtet: API-based communication, Builtin autoscaling, Cloud vendor abstraction, Coarse-Grained Microservices, Loose Coupling, Manageable Connections, Manageable Standards, Seperate Libraries, Service Independence, Standardization, Standardized deployment unit, Versioned APIs. Da Firmeninterne vorgaben bestehen, welche besagen, dass Messaging und Containerization zu einem späteren Zeitpunkt diskutiert werden sollen gab es eine dritte Filterung, welche diese Best Practices herausfiltert, damit sie zu einem späteren Zeitpunkt betrachtet werden können. Bei dritter Filterung für später zurückgestellt: Acyclic Calls, Persistent Communication. Bei dritter Filterung verblieben: Access restriction, Account Separation, Authentication delegation, Automated monitoring, Automated restarts, Autonomous fault handling, Configuration management, Data encryption in transit, Dynamic scheduling, Guarded ingress, Seamless upgrades, Secrets management, Separation by gateways, API-based communication, Built-in autoscaling, Cloud vendor abstraction, Coarse-Grained Microservices, Loose Coupling, Manageable Connections, Manageable Standards, Seperate Libraries, Service Independence, Standardization, Standardized deployment unit, Versioned APIs. Da das System, welches in dieser Arbeit implementiert wird, noch keine hohe Komplexität besitzt, wurden weitere Best Practices für die spätere Implementierung zurückgestellt. Bei vierten Filterung für später zurückgestellt: Authentication delegation, Automated monitoring, Automated restarts, Autonomous fault handling, Data

Kategorie	Inhalt
	encryption in transit, Dynamic scheduling, Seamless upgrades, Secrets management, Separation by gateways, Built-in autoscaling, Cloud vendor abstraction. Bei vierten Filterung verblieben: Access restriction, Account Separation, Configuration management, Guarded ingress, API-based communication, Coarse-Grained Microservices, Loose Coupling, Manageable Connections, Manageable Standards, Seperate Libraries, Service Independence, Standardization, Standardized deployment unit, Versioned APIs
Betroffener Teil der Software:	-
Was lief gut:	Best Practices wurden gut über die QAs, SQAs, SPs gefiltert.
Was lief schlecht:	Die Bedeutung mancher Best Practices war unklar, vielleicht könnte man eine kurze Beschreibung bereitstellen.
Empfindungen:	-
Kommentare:	Es kommt mir so vor, dass bei der reinen Betrachtung der QAs, SQAs und SPs zu viele Best Practices übrig bleiben. Viele der bei der ersten Filterung verbliebenen Best Practices scheinen sinnvoll und hilfreich zu sein. Wenige Best Practices die bei der ersten Filterung rausgeflogen sind scheinen doch noch wichtig zu werden, z. B. API- based communication, Built-in autoscaling, Cloud vendor abstraction, Coarse-Grained Microservices, Loose Coupling, Manageable Connections, Manageable Standards, Seperate Libraries, Service Independence, Standardization, Standardized deployment unit, Versioned APIs. Ein paar der verblieben Best Practices scheinen unnötig zu sein, deswegen gab es eine zweite Filterung. Bei der Filterung wurde sich nur auf die drei am höchsten Priorisierten QAs von L-mobile konzentriert(Security, Functional completeness, Reliability), da sonst jedes Best Practice zutreffen würde. Es ist nicht ausgeschlossen, dass in diesem Schritt ausgefilterte Best Practices zu einem späteren Zeitpunkt wichtig werden und implementiert werden.
Sonstiges:	-

Tabelle C.25.: F25_P3_230527_BestPracticesAnalyse

Kategorie	Inhalt
Feldnotiz Nr.:	26
Datum:	30.05.23
Uhrzeit:	9:00 Uhr - 9:45 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Software Architekt
Phase des FMM:	3
Schritt:	Architektur Erzeugung
Durchgeführte Aktivitäten:	Die generierte Architektur wurde gereviewed. Viele der Kommentare bezogen sich auf die Gestaltung der Microservices, was jedoch nichts mit der Identifizierung zu tun hat, sondern mit internem Wissen. (Services Diagramm: was ist ZeitIntervalle?) - Services Diagram: „Main“, „Service“, „Library“ sind keine wirklichen Services. - Crm.PdfGeneration: die PDF-Generierung ist doch bereits ausgelagert in CefToPdf mit separatem Repo. - Crm.ProjectOrders ist vielleicht nicht die beste Wahl für einen Microservice, da das kein Service ist sondern über die Plugin) Der Punkt dass jeder Service seine Lizenzierung selbst verwalten sollte wurde angesprochen, dass dies nicht so sinnvoll sei, da zu aufwendig, daraufhin wurde dieser Punkt geändert.
Betroffener Teil der Software:	MSA
Was lief gut:	Konstruktives Feedback über die Architektur erhalten. Ein Großteil der MSA wurde ohne Änderungen akzeptiert.
Was lief schlecht:	-
Empfindungen:	-
Kommentare:	Es gab in diesem Schritt keine Empfindungen, da der Software-Architekt, welcher die Architektur gereviewed hat dieses Review lieber alleine durchgeführt hat. Die Patterns und Best Practices wurden alle angenommen, und es wurde das Pattern Log Aggregation gewünscht.
Sonstiges:	-

Tabelle C.26.: F26_P3_230530_ArchitekturReview

Kategorie	Inhalt
Feldnotiz Nr.:	27
Datum:	30.05.23 - 04.06.23
Uhrzeit:	-
Ort:	Daheim und Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Implementierung
Durchgeführte Aktivitäten:	Der Microservice PdfGenerierung wurde implementiert. 30.05.23 Es wurden Grundlagen über die Erstellung eines neuen .Net Projektes für Microservices angeeignet. Es wurde Wissen über die Implementierung von Microservices angeeignet, hauptsächlich über das YouTube Video[1]. Es wurden die Klassen PdfService.cs und IPdfService.cs von Crm.Library in das neue Microservice Projekt kopiert. 31.05.23 Weitere Klassen, von denen PdfService.cs und IPdfService.cs Abhängen und von denen die Abhängigkeiten abhängen etc. wurden in das Microservice Projekt kopiert. 01.06.23 Startup.cs und Program.cs wurden generiert. Erste DTOs wurden erzeugt. PdfController.cs für die REST API wurde generiert. 02.06.23 Eine Konfiguration mit den ersten Werten für den Microservice wurde erzeugt. PdfModule.cs für die Dependency Injection mit AutoFac wurde implementiert. 04.06.23 Weitere Abhängigkeiten wurden vom Monolithen in das Microservice Projekt kopiert. Eine erste prototypische Kommunikation zwischen Monolith und Microservice wurde implementiert(fake it till you make it).
Betroffener Teil der Software:	Crm.Library und neuer Microservice (PdfGenerierung)
Was lief gut:	Der Microservice lies sich gut extrahieren und ausführen. Grundwissen ließ sich schnell und einfach aneignen.
Was lief schlecht:	Ab und zu gab es ein paar Schwierigkeiten, die jedoch gelöst werden konnten.
Empfindungen:	Gut, erleichtert, freudig, in Situationen in denen es etwas schwerfälliger voran ging aber auch ängstlich und unsicher.

[1] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg

Kategorie	Inhalt
	Dankbar dass viel Material zur Aneignung von Grundlagen/ Wissen für die Generierung von Microservices frei verfügbar ist.
Kommentare:	Das YouTube Video[2] gibt eine sehr gute Einführung über Microservices mit .Net.
Sonstiges:	Es wurden auch noch weitere Videos zu Microservices angeschaut und zur Grundlagen von ASP .Net Projekten. Es wurde in GitHub nach Open Source .Net Microservice Projekten gesucht, für den Fall dass die YouTube Videos nicht genügend Information lieferten, dieses Material wurde jedoch nicht benötigt. Nicht nur am 30.05. wurde das YouTube Video[3] zur Rate gezogen, auch an den anderen Tagen wurde es verwendet, wenn Informationen nochmal nachgeschaut werden mussten.

Tabelle C.27.: F27_P3_230604_ImplementierungMicroservice

[2] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg
[3] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg

Kategorie	Inhalt
Feldnotiz Nr.:	28
Datum:	05.06.23 - 11.06.23
Uhrzeit:	-
Ort:	Daheim und Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Implementierung
Durchgeführte Aktivitäten:	Der Microservice PdfGenerierung wurde implementiert. 06.06.23 AppSettings wurden in der Konfigurationsdatei hinzugefügt. Eine Konfiguration für das Logging wurde angelegt. Abhängigkeiten die für die Implementierung eines Loggers notwendig waren wurden vom Monolithen in den Microservice kopiert. Der Logger wurde in AutoFac registriert. Das Repository wurde von unbenutzten Klassen gereinigt, dieser Commit wurde später rückgängig gemacht. Probleme mit dem Logger treten auf und es wurde versucht diese zu fixen. 07.06.23 Es wurde versucht das Problem mit dem Logger zu lösen. Es wurde viel über das Thema recherchiert, jedoch ohne nennenswertes Ergebnis. 08.06.23 Es wurde weiterhin an dem Logger Problem gearbeitet, jedoch ohne Ergebnis. 09.06.23 Rückgang auf die Version vom 06.06.23 Vormittags welche noch funktioniert hat. Nicht benutzte Klassen und Code wurden gelöscht, Repository und Code wurde gereinigt, dieses mal trat der Fehler nicht mehr auf. Die Dummy Kommunikation wurde zu einer richtigen Kommunikation umgebaut, zumindest für die erste Methode Html2Pdf. Die Startup Klasse wurde angepasst.
Betroffener Teil der Software:	Crm.Library und neuer Microservice (PdfGenerierung)
Was lief gut:	Der Microservice lies sich gut extrahieren und ausführen.
Was lief schlecht:	Ab und zu gab es ein paar Schwierigkeiten, die jedoch gelöst werden konnten. In dieser Woche kam es am 09.06.23 zu einem Problem, der Logger ließ sich nicht korrekt instanziieren, weshalb auf den Stand vom 06.06.23 zurück gegangen wurde. Sobald dieses Problem gelöst war (am 12.06.23) wurden die Versionen gemerged, aber keine der Änderungen vom 06.06.23 Nachmittags bis 08.06.23 wurden

Kategorie	Inhalt
	übernommen. Dies hat viel Zeit und Nerven gekostet.
Empfindungen:	Gut, erleichtert, freudig, in Situationen in denen es etwas schwerfälliger voran ging und zu manchen Problemen kam aber auch ängstlich, frustriert und unsicher. Sobald die Probleme jedoch behoben waren wieder euphorisch.
Kommentare:	Das YouTube Video[4] gibt eine sehr gute Einführung über Microservices mit .Net.
Sonstiges:	-

Tabelle C.28.: F28_P3_230611_ImplementierungMicroservice

[4] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg

Kategorie	Inhalt
Feldnotiz Nr.:	29
Datum:	12.05.23 - 23.06.23
Uhrzeit:	-
Ort:	Daheim und Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel, Software Architekt
Phase des FMM:	3
Schritt:	Implementierung
Durchgeführte Aktivitäten:	Der Microservice PdfGenerierung wurde implementiert. 12.05.23 Ein Problem Mit der SiteService.cs Klasse tritt auf, es wird zur Laufzeit keine Instanz der DomainExtension generiert. Da durch den Rückgang am 09.06.23 auf eine frühere Version ein neuer Head erzeugt wurde, werden beide heads gemerged, es werden jedoch keine Änderungen vom 06.06.23 Nachmittags bis 08.06.23 übernommen. Da sich das Problem mit der DomainExtension nicht so schnell lösen ließ, wurde eine Frage im Intranet von L-mobile gestellt und der Aufruf dieser Extension im Code durch hardgecodede Adressen ersetzt, fake it till you make it. Die PdfServiceClient.cs Klasse wurde in eine eigene Datei ausgelagert, davor war sie eine innere Klasse in PdfService.cs, und es wurden Methoden für die Kommunikation mit weiteren API Knoten des Microservice generiert. Die API im Microservice wird erweitert. Anstatt Parameter über die URL zu empfangen werden jetzt DTOs über den Aufruf Body entgegengenommen, die Kommunikation im Monolith wurde dahingehend ebenfalls angepasst. Weitere DTOs werden im Microservice und im Monolith implementiert. 13.05.23 Ein erster prototypischer Circuit Breaker wurde mittels der Library Polly implementiert. Nicht verwendete Klassen und Code wurden entfernt, Repository wurde gereinigt. Die URL des Microservice wurde in eine Konfigurationsdatei des Monolithen ausgelagert und wird für die Kommunikation des Monolithen mit dem Microservice vom Monolithen aus der Konfigurationsdatei ausgelesen. CefToPdfPath wird vom Microservice nun von einer Konfigurationsdatei eingelesen. Es gab eine erste Antwort auf die gestern gestellte Frage warum die DomainExtension nicht gesetzt wird. Daraufhin gab es eine

Kategorie	Inhalt
	weitere Recherche wie das Problem gelöst werden kann. 14.05.23 Code wurde bereinigt. Das Problem mit der nicht gesetzten DomainExtension wurde gelöst. Die Klassen ExtensibleObjectRegistrationSource.cs, DomainExtensionMapping.cs, DefaultDynamicExtensionRegistry.cs haben gefehlt und wurde in das Microservice Projekt kopiert. Im Module wird diese Extension nun richtig registriert und die hardgecodeden Adressen wurden wieder durch den Methodenaufruf in der DomainExtension ersetzt. log4net.config wird der Wert für das Loggen von CefToPdf ergänzt. Die Kommunikation im Microservice wird nun ohne eine Instanz der Klasse Site realisiert. Es kommt zum Kommunikationsproblem zwischen Microservice und Monolith im wöchentlichen Meeting. Das Problem wurde jedoch schnell identifiziert (Monolith wartet nicht auf Antwort von Microservice sondern läuft nach Aufruf direkt weiter(async Problem)). Die Kommunikation im Monolith wird async resistent implementiert und damit das Kommunikationsproblem behoben, Monolith wartet jetzt auf die Antwort vom Microservice. 15.05.23 Ein neues Problem tritt auf, der Aufruf von CefToPdf braucht zu lange um eine PDF zu generieren und liefert eine TimeoutException zurück. Daraufhin wird das Timeout des CefToPdfConverter auf Infinite gesetzt, was das Problem jedoch nicht löst, jetzt liefert CefToPdf-Converter keine PDF und keine Exception zurück selbst nach Stundenlangem betrieb. Die verbliebenen Aufrufe von CefToPdf wurden aus dem Monolithen entfernt, da diese jetzt rein über den Microservice geschehen. Die Kommunikation wird auch für die anderen API aufrufen und Methoden verbessert(Daten über DTO im Request Body). Unnötiger Code wurde entfernt. 16.05.23 Nachdem der Software Architekt sich das Problem mit dem CefToPdfConverter angeschaut hatte, stellte er fest dass die URL, welche zur Generierung des HTML Strings, aus dem die PDF generiert wird, aus der Datenbank gelesen wird eine falsche URL besitzt. Daraufhin wurde die URL in die localhost URL geändert, der CefToPdfConverter

Kategorie	Inhalt
	funktioniert jetzt. Der generierte HTML String wird jetzt wieder an den Microservice gesendet. 21.05.23 Weitere Klassen, welche das Problem lösen, dass das Logging nicht richtig funktioniert wurden in das Microservice Projekt kopiert und im PdfModule registriert. Das Logging funktioniert jetzt. Unnötige Klassen und Code wurden gelöscht, Repository und Code wurde gereinigt. Die Kommunikation im Monolith wurde weiter auf die Antwort vom Microservice angepasst, für die weiteren Methoden außer Html2Pdf. 22.05.23 Die Kommunikation für die zweite Html2Pdf Methode wurde angepasst. Unbenutzter Code wurde gelöscht, Code gereinigt.
Betroffener Teil der Software:	Crm.Library und neuer Microservice (PdfGenerierung)
Was lief gut:	Der Microservice lies sich gut extrahieren und ausführen.
Was lief schlecht:	Ab und zu gab es ein paar Schwierigkeiten, die jedoch gelöst werden konnten. Bei der Demonstration im wöchentlichen Meeting zwischen Jonas, Alex und mir kam es zu einem Problem bei der Rückkommunikation vom Microservice zum Monolithen, der Wert der zurückgesendet wurden kam beim Monolithen nicht an. Ich kam aber schnell darauf, dass es sich um ein async Problem handelt und konnte dies schnell beheben. Ein Problem Mit der SiteService.cs Klasse tritt auf, es wird zur Laufzeit keine Instanz der DomainExtension generiert. Dieses Problem konnte erst nach einigen Tagen und nach einem Hinweis von einem Software Engineer gelöst werden. Ein Problem mit dem CefToPdfConverter konnte ebenfalls erst mit Hilfe des Software Architekten nach einem Tag gelöst werden. Das Logging hat nicht von Anfang an funktioniert, konnte aber am Ende behoben werden.
Empfindungen:	Gut, erleichtert, freudig, in Situationen in denen es etwas schwerfälliger voran ging und zu manchen Problemen kam aber auch ängstlich, frustriert und unsicher. Sobald die Probleme jedoch behoben waren wieder euphorisch.

Kategorie	Inhalt
Kommentare:	Das YouTube Video[5] gibt eine sehr gute Einführung über Microservices mit .Net. Es gab Schwierigkeiten bei der Rückkommunikation (Microservice -> Monolith), was sich als async Problem herausstellte und leicht beheben lies. Eine externe Abhängigkeit lieferte anfangs keine Ergebnisse, nachdem ein Entwickler zur Rate gezogen wurde, wurde die Ursache gefunden und behoben.
Sonstiges:	-

Tabelle C.29.: F29_P3_230623_ImplementierungMicroservice

[5] https://youtu.be/CqCDOosvZIk?si=AFP8hIsqvYh5cVPg

Kategorie	Inhalt
Feldnotiz Nr.:	30
Datum:	16.06.23 - 23.06.23
Uhrzeit:	-
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Marvin Knodel
Phase des FMM:	3
Schritt:	Testen des Microservice
Durchgeführte Aktivitäten:	Es wurden bestehende Tests für den Anwendungsfall des PdfGenerierungs- Microservice gesucht und durchgeführt. Bestehende Tests wurden gesucht, gefunden und ausgeführt. Anfangs gab es das Problem, dass nicht alle Tests durchliefen, nach ein paar Stunden Recherche ist aufgefallen, dass die watcher.bat nicht lief während die Instanz des Monolith lief. Die Inbetriebnahme der watcher.bat löste das Problem. Weitere Tests wurden gefunden, ausgeführt und erfolgreich abgeschlossen.
Betroffener Teil der Software:	Code, Instanz des Microservice und Monolith, Tests.
Was lief gut:	Zum Schluss ließen sich die Tests alle erfolgreich ausführen. Es mussten keine Änderungen im Monolithen oder im MS vorgenommen werden, damit die Tests erfolgreich sind.
Was lief schlecht:	Zwischenzeitlich sind ein paar Tests fehlgeschlagen und es war eine lange Recherche nötig warum dies der Fall war, es stellte sich jedoch raus, dass die watcher.bat während der Testausführung nicht lief und die Tests deshalb fehlschlugen. Nachdem die watcher.bat gestartet wurde, liefen alle Tests erfolgreich durch.
Empfindungen:	Kurz panisch und angst als die Tests fehlgeschlagen sind. Aber erleichtert und froh nachdem sie doch durchgelaufen sind.
Kommentare:	-
Sonstiges:	-

Tabelle C.30.: F30_P3_230623_TestenMicroservice

Kategorie	**Inhalt**
Feldnotiz Nr.:	31
Datum:	04.07.23
Uhrzeit:	Vor 9:15 Uhr
Ort:	Sulzbach an der Murr
Beteiligte Personen:	Software Architekt, Marvin Knodel(bei der Umsetzung)
Phase des FMM:	3
Schritt:	Code Review
Durchgeführte Aktivitäten:	Der Code wurde von dem Software Architekten gereviewed. Ein paar der Punkte aus dem Code Review wurden umgesetzt. Anmerkungen des Software Architekten: - Die Anwendung baut (im CI) nicht. - Der Microservice akzeptiert jetzt Get-Requests mit einem json im Request Body, das ist nicht richtig. Ein Get Request sollte keinen Request Body haben. Sobald da irgend eine Art Cache zwischen Monolith und Microservice ist gibt es Probleme. - PdfDTOs und PdfServiceClient gehören ins PdfGeneration Projekt, bzw. ist jetzt sogar doppelt vorhanden. - In PdfServiceClient.cs ist viel duplizierter Code. - PdfController ohne Autorisierung, ist das gewollt? Im html zu PDF Fall wahrscheinlich egal, aber wie sähe das bei MSs aus bei denen eine Autorisierung benötigt wird? - ReplaceRelativeWithAbsolutePaths => diese Logik gehört eher nicht in den Microservice. - ReplaceVariables => das muss nicht von außen als Serviceaufruf erreichbar sein. - Wofür wird CommunicationToMonolith/Host CommunicationToMonolith/Port in den AppSettings genutzt? Der Monolith sollte dem Microservice doch nicht bekannt sein, der Microservice kann ja von verschiedenen Anwendungen heraus aufgerufen werden. - Viel kopierter Code von Crm.Web/Crm.Library/Main in den Microservice statt eine Referenz auf die benötigten Projekte. Bei dem zweiten Microservice würde das dann eher auffallen dass das sehr viel Boilerplate Code ist. Der gehört in ein separates Projekt was man dann referenziert. - Da scheint überhaupt mehr drin zu sein als nötig. Der Microservice kennt jetzt die Plugin Infrastruktur, Multitenancy, initialisiert NHibernate mit einem eigenen

Kategorie	Inhalt
	Connectionstring, etc. Dinge die um html in pdf umzuwandeln ja eigentlich nicht benötigt werden.
Betroffener Teil der Software:	Code
Was lief gut:	Die ins gesamte Struktur des Microservice scheint gut zu sein (Rückschluss von mir). Manche der umsetzungsfähigen Punkte ließen sich gut umsetzen.
Was lief schlecht:	-
Empfindungen:	Ein paar der Punkte sind verständlich (1, 4, 5, 6, 7, 9 und 10). Gutes Gefühl bei der Umsetzung.
Kommentare:	Kein wirklicher Schritt des FMM. Durch eine Aufwandsabschätzung wurde entschieden, dass die Punkte 4, 6 und 7 noch in dieser Arbeit umgesetzt werden können. Der Rest muss nach dieser Arbeit umgesetzt werden.
Sonstiges:	Nach der Umsetzung von Punkt 6 kam es zu Fehlern in der Testausführung, weshalb dieser Schritt wieder rückgängig gemacht wurde.

Tabelle C.31.: F31_P3_230705_CodeReviewSoftwareArchitect

Kategorie	**Inhalt**
Feldnotiz Nr.:	32
Datum:	12.07.23
Uhrzeit:	Vor 12:15 Uhr
Ort:	Home Office(Security Engineer), Sulzbach an der Murr (Marvin)
Beteiligte Personen:	Security Engineer, Marvin Knodel (bei der Umsetzung)
Phase des FMM:	3
Schritt:	Code Review
Durchgeführte Aktivitäten:	Der Code wurde von dem Security Engineer hinsichtlich Security Lücken gereviewed. Ein paar der Punkte aus dem Code Review wurden umgesetzt. Anmerkungen des Security Engineers: - Web API allgemein: ∼ Authentifizierung: Nutzung des API ist derzeit ohne Authentifizierung möglich. Bei jedem Aufruf eines API Endpunkts muss die Identität des jeweiligen Nutzers sichergestellt werden. Hierfür sollte der vorhandene Token basierte Autentifizierungsmechanismus genutzt werden. ∼ Autorisierung: Das API ist ohne Überprüfung der Autorisierung des jeweiligen Nutzers verwendbar. Die CRM und Service Anwendung verfügt über ein granulares Berechtigungsmanagementsystem, das auf den Microservice ausgeweitet werden sollte. ∼ HTTP Verbindungen werden akzeptiert, in produktiven Umgebungen sollte diese Möglichkeit zur unverschlüsselten Übertragung von Informationen unterbunden werden. - DTO-Klasse ∼ Positiv zu bewerten: grundsätzlicher Einsatz von DTO. Datenkapselung ist wichtig für die Sicherheit von Microservice Architekturen, bietet hier Typsicherheit. ∼ Negativ: DTO hat in diesem Fall keine Möglichkeit zur Inhaltsvalidierung, DTOs werden direkt mit clientseitig generierten Inhalten befüllt, hier besteht prinzipiell Risiko diverser Injections oder DoS Angriffe. ∼ byte[] ohne Größenbegrenzung kann potenziell für DoS missbraucht werden bei Überlastung. - Controller

Kategorie	Inhalt
	∼ Übersichtliche Trennung einzelner Funktionen in verschiedenen Endpunkten, gut wartbar, einfach testbar. ∼ Kein Auth vorgelagert (siehe oben). ∼ Kritisch: Eingabedaten werden nicht validiert, es erfolgt lediglich eine Transformation in DTO. ∼ Eingaben landen direkt in pdfService.Html2Pdf, führt keine Validierung oder Sanitizion aus. ∼ Risiko von Code Injections in generierte PDFs, besonders kritisch da PDFs von dem Nutzer vertrauter Quelle (CRM) stammen, folglich wird den generierten PDFs implizit vertraut. ∼ Einsatz von Log4Net sehr gut, Logging ist elementar zur Nachverfolgung und Behebung potentieller Sicherheitslücken. ∼ Prüfen ob Logging richtig funktioniert, ggf. weiter ausbauen. ∼ Prüfen ob beim Loggen sensible Daten abfließen können, ggf. ausschließen. ∼ Prüfen ob TimeoutExceptions korrekt arbeiten und ggf. anpassen, kann DoS durch lange Verzögerungen oder Mutual Concurrency verhindern. ∼ ReplaceRelativeWithAbsolutePaths kümmert sich zwar um relative Pfadangaben, stellt allerdings keine Absicherung gegen Path Vulnerabilities dar, prüfen und ggf. Schutz umsetzen. ∼ Stamp und MergeFiles verwenden Daten ungefragt ohne Inhaltsprüfung, ggf. auf bestimmte Datentypen begrenzen.
Betroffener Teil der Software:	Code
Was lief gut:	Die Umsetzung mancher Punkte war herausfordernd aber gut machbar.
Was lief schlecht:	-
Empfindungen:	Die Punkte sind verständlich. Die Anmerkungen sind schwierig umzusetzen.
Kommentare:	Kein wirklicher Schritt des FMM. Durch eine Aufwandsabschätzung wurde entschieden, dass die Punkte 1.1, 3.2 und 3.4 noch in dieser Arbeit umgesetzt werden

Kategorie	Inhalt
	können. Der Rest muss nach dieser Arbeit umgesetzt.
Sonstiges:	-

Tabelle C.32.: F32_P3_230712_CodeReviewSecurityEngineer

Codierung	Bedeutung
Angst	Bezieht sich auf ein ängstliches Gefühl bei der Durchführung der, von der Feldnotiz behandelten, Aufgabe.
ARH	Bezieht sich auf das ARH-Tool.
fitymi	Bezieht sich auf das fake it till you make it Prinzip.
Gut	Bezieht sich auf Dinge die in der, von der Feldnotiz behandelten, Aufgabe gut liefen.
Gutes Gefühl	Bezieht sich auf positive Empfindungen die in der, von der Feldnotiz behandelten, Aufgabe aufgetreten sind.
Hilfe	Bezieht sich auf Hilfe die von anderen Software Entwicklern angefordert und gewährt wurde.
Internes Problem	Bezieht sich auf ein Problem welches für die L-mobile Applikation aufgetreten ist, nicht verursacht durch das FMM, z. B. dass viele Tools für die Service-Identifizierung nur für Java Applikationen implementiert sind.
Schlecht	Bezieht sich auf Dinge die in der, von der Feldnotiz behandelten, Aufgabe schlecht liefen.
Schlechtes Gefühl	Bezieht sich auf negative Empfindungen die in der, von der Feldnotiz behandelten, Aufgabe aufgetreten sind.
Tools	Feldnotiz bezieht sich auf die Filterung und Anwendung von Tools zur Service-Identifizierung.
Unsicher	Bezieht sich auf ein unsicheres Gefühl bei der Durchführung der, von der Feldnotiz behandelten, Aufgabe.
Verbesserungsvorschlag	Bezieht sich auf einen in der Feldnotiz beschriebenen Verbesserungsvorschlag für das FMM.
Wdh.	Wiederholung in der Formulierung von vorherigen Feldnotizen.
Zuversichtlich	Bezieht sich auf zuversichtliches Gefühl die in der, von der Feldnotiz behandelten, Aufgabe aufgetreten sind.

Tabelle C.33.: Die Codierung für die Auswertung der Feldnotizen.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift