

Road to Microservices

ANA ISABEL GOMES MOREIRA BASTOS FONSECA

Outubro de 2022

A Road to Microservices

Ana Isabel Fonseca

**A dissertation submitted in partial fulfillment of
the requirements for the degree of Master of Informatics
Engineering, Specialisation Area of Software Engineering**

**Supervisor: Isabel Azevedo
External Supervisor: Vasco Oliveira**

Porto, October 13, 2022

Dedicatory

"(...)E foi perguntar-me a mim mesmo se realmente tinha alguma coisa para dizer que valesse a pena; para sentar-me a escrever.

Esse foi o grande momento da minha vida, em que, numa situação complicada, bastante difícil, eu tomo uma decisão arriscada.

Efetivamente...se há um momento na minha vida que é um momento-chave, é esse, o momento de decisão. Não perguntar a ninguém "E agora o que é que eu faço?", que foi coisa que nunca perguntei a ninguém.

"E agora o que é que eu faço? Dê-me um conselho." Nunca pedi conselhos a ninguém. Também não os dou. A não ser que me peçam. E ainda assim, com muitas reservas.

Pois...É agora ou nunca que vou saber, finalmente, se eu sou escritor ou não sou escritor...e tinha 60 anos, meu caro"

- José Saramago, do documentário *José e Pilar*

Abstract

This document intends to elucidate the complexity of microservices decomposition and this its inherent process of implementation.

Developments in technology and design, achieving higher performance, reliability, or lowering costs are valid reasons to consider controlling the product's quality by guaranteeing its conformance with established characteristics and standards. Control is made possible by adding quality control, inspection routines, and trend analysis to a manufacturing process. These techniques are established in the Quality field academically and business-wise.

Repeat Part Management (RPM) is software that allows users to apply these techniques efficiently and has brought value to the company. However, RPM has been growing, and issues have emerged due to customer needs - accumulated technical debt. These ever-growing modifications are common through different business areas, and architectures' research evolution has accompanied them. This demand for highly modifiable, rapid development, and independent systems has resulted in the adoption of microservices.

There is a concern for existing systems for decomposition due to the characteristics of microservices, which encourages approach/technique research. This architecture promotes legacy system analysis to map current functionalities and dependencies between components.

Furthermore, critical concepts in a microservices architecture are researched and implemented in a new system that encompasses Repeat Part Management's functionalities.

This thesis explores the microservices' architecture evolution with an already defined mature domain in quality assessment.

Keywords: Microservices, Design, Architecture, Pattern, Antipatterns, Migration, Decomposition

Resumo

Este documento pretende especificar a complexidade do processo de decomposição em microsserviços e do seu processo de implementação.

Avanços na tecnologia e design, alcançar melhor performance, ou a confiabilidade do produto, ou diminuir custos são justificações válidas para considerar controlar a qualidade do produto e, inerentemente, garantir a sua conformidade com as características previamente definidas e com padrões da indústria. É possível garantir controlo sob os produtos ao acrescentar, ao processo produtivo, métodos de controlo de qualidade, rotinas de inspeção e análise de tendências (de desvio). Estas técnicas estão bem estabelecidas academicamente e, de um ponto de vista do mercado, na área da Qualidade e garantia da qualidade.

O Repeat Part Management (RPM) é um software que permite aos seus utilizadores a utilização eficiente destas técnicas de qualidade, o que resulta numa adição de valor para a empresa. Porém, devido às crescentes necessidades dos clientes, alguns problemas têm sido identificados - relacionados com o conceito de acumulação de *technical debt*. Esta crescente necessidade de alteração é comum em diversas áreas de negócio, e a investigação de soluções arquiteturais tem acompanhado esta pesquisa. Esta solução arquitetura pode ser caracterizada pela facilidade de sistemas facilmente modificáveis, pelo rápido desenvolvimento e implementação, e pela independência dos serviços decompostos.

Aquando de uma migração para microsserviços num sistema maturo, existe uma maior preocupação na decomposição da aplicação e definição dos serviços dada a característica dos microsserviços, o que incentiva a uma pesquisa detalhada sobre as técnicas de decomposição. Pela mesma razão, esta arquitetura incentiva ao mapeamento e documentação das dependências entre serviços e componentes e o estudo da aplicação *legacy*.

Para além disto, estes conceitos, e a sua implementação devem ser planeados, justificados e documentados, o que explica a complexidade do processo de migração e implementação, que deve ter em consideração as funcionalidades existentes no Repeat Part Management.

Dessa forma, esta tese explora a implementação desta arquitetura numa aplicação matura na área de Garantia de Qualidade.

Acknowledgement

À minha mãe e ao Panos por serem pacientes.

À Cooperativa por estar presente por todas as fases do desespero.

À Maria por partilhar as minhas dores.

Ao Renato pelo ombro e as necessárias pausas.

Ao Xavier e ao Chamusca por me animarem.

Tornaram este processo significativamente mais fácil.

Contents

List of Figures	xv
List of Tables	xvii
List of Acronyms	xix
1 Introduction	1
1.1 Context	1
1.2 Problem	2
1.3 Objectives	4
1.4 Methodology	5
1.5 Document Structure	5
2 Legacy Context	7
2.1 Manufacturing Quality	7
2.1.1 Quality in Manufacture	7
2.1.2 Quality Control	8
2.2 Business Domain	8
2.2.1 CAM2	8
2.2.2 Repeat Part Management (RPM)	9
2.2.3 Relation between CAM2 and RPM	12
2.3 Stakeholders Overview	15
3 State of the Art	17
3.1 Architectural Evolution	17
3.2 Microservices	18
3.3 Architectural Patterns Research	21
3.3.1 Research Results	23
3.3.2 Design Patterns	25
3.4 Decomposition Research	26
3.4.1 Research Protocol Definition	26
3.4.2 Conducting the Search for Primary Studies	28
3.4.3 Screening	29
3.4.4 Classification System	30
3.4.5 Data Extraction and Aggregation	30
3.5 Analysis and Report	30
3.5.1 Research Question 1 (RQ1)	31
3.5.2 Research Question 2 (RQ2)	32
4 Value Analysis	35
4.1 Innovation Process	35

4.2	Front End of Innovation (FEI)	36
4.2.1	Opportunity Identification	36
4.2.2	Opportunity Analysis	37
4.2.3	Idea Genesis	37
4.2.4	Idea Selection	38
4.2.5	Concept Development	42
4.3	Value Analysis	42
4.3.1	Value for the Customer	42
4.3.2	Perceived Value	42
4.3.3	Value Proposition	43
4.4	Business Model Canvas	44
5	Design	47
5.1	Design	47
5.1.1	Attribute Driven Design (ADD)	48
5.1.2	Architectural Drivers	48
5.1.3	Functional Requirements	50
5.2	System Decomposition Alternatives	51
5.2.1	Decomposition by Business Capability Alternative	51
5.3	Applied Decomposition	52
5.3.1	Responsibility Separation	52
5.3.2	Document Service Domain Model	53
5.3.3	Report Service Domain Model	54
5.3.4	Session Service Domain Model	57
6	Implementation	59
6.1	Logical View	60
6.2	Process View	61
6.3	Development View	62
6.4	Physical View	66
6.5	Patterns Implementation	66
6.5.1	Domain-Driven Design	66
6.5.2	Database Per Service	68
6.5.3	OpenAPI	69
6.5.4	API Gateway	69
6.5.5	Containerization	71
7	Experimentation and Evaluation	73
7.1	Hypothesis Identification and Evaluation Indicators	73
7.2	Validation by Stakeholders	73
7.3	Hypothesis Evaluation	74
7.4	Questionnaire Answers Analysis	75
7.4.1	Identified Issues Section	76
7.4.2	Microservices Implementation Adequacy Section	78
7.5	Hypothesis Calculation	80
8	Conclusion	81
8.1	Goal Achievement	81
8.2	Threats to Validity	82
8.3	Future Work	82

A	Task Goals	89
A.1	Tasks	89
B	Auxiliary AHP Data	91
B.1	Random Index (RI) Table	91
B.2	Criteria Normalized Matrices	91
C	Architectural Patterns Research Results	93
C.1	Monitoring Patterns	93
D	RPM's Third Party Software	95
D.1	Third Party Software	95
E	Conducting the Search for Primary Studies	97
E.1	Primary Studies	97
F	Evaluation	99
F.1	Stakeholder Questionnaire	99

List of Figures

1.1	Relation between CAM2 and RPM	3
1.2	Problem Context Diagram	3
1.3	Thesis Tasks Overview	4
2.1	Control Server Domain Model	11
2.2	Screw [Adapted from [27]]	12
2.3	RPM Plugin in CAM2	12
2.4	Feature and Business Flow (BPMN)	13
2.5	Application's Capabilities regarding the domain (BPMN)	14
3.1	Microservices and its effect on development [Retrieved from [57]]	19
3.2	Domain Driven Design Overview [Retrieved from [44]]	21
3.3	Systematic Mapping Study Process [Adapted from [74]]	26
3.4	Classification System	30
4.1	Value Concept	35
4.2	New Concept Development Model	36
4.3	Decision Hierarchy Diagram	39
4.4	Business Model Canvas	45
5.1	Functional Requirements	50
5.2	Legacy Use Cases	51
5.3	Decomposition by Business Capability - Logical View Alternative	52
5.4	Context Map	53
5.5	Document Microservice Domain Model	54
5.6	Report Content Example	55
5.7	Report Microservice Domain Model	56
5.8	Session Microservice Domain Model	57
6.1	4+1 model [Adapted from [50]]	59
6.2	Implemented Logical View Diagram	60
6.3	Logical View Alternative (with Message Broker)	60
6.4	CleanArchitecture Overview [Adapted from [78]]	61
6.5	Process View Diagram	62
6.6	Report Solution Development View Diagram	63
6.7	Document Solution Development View Diagram	63
6.8	Create Report Sequence Diagram	65
6.9	Implemented Deployment View	66
6.10	Entity Framework in Report (Business Logic)	67
6.11	Context Database Set	67
6.12	Creating Report Model (Database Context)	68
6.13	OpenAPI Config Code	69

6.14	OpenAPI Report Controller Registration	69
6.15	Ocelot Basic Implementation [Adapted from [64]]	70
6.16	Ocelot Basic Routing Example	71
7.1	Job Title Division	75
7.2	Project Division	75
7.3	Microservices Adequacy Answers (2.1)	76
7.4	Microservices Responsibility Answers (2.2)	77
7.5	Functionality Implementation Answers (2.3)	77
7.6	Microservices General Experience Answers (2.4)	78
7.7	Microservices Pattern Adequacy Answers (3.1)	78
7.8	System Modularity Answers (3.2)	79
7.9	System Modifiability Answers (3.3)	79
7.10	System Maintainability Answers (3.4)	80
F.1	Questionnaire Introductory Text	99
F.2	About You Introductory Text	99
F.3	About You Questions	100
F.4	Expectations Introductory Text	100
F.5	Expectations Questions - Part 1	101
F.6	Expectations Questions - Part 2	101
F.7	Implementation Introductory Text	102
F.8	Implementations Questions - Part 1	102
F.9	Implementations Questions - Part 2	102

List of Tables

2.1	CAM2 features list	9
2.2	RPM's Legacy Responsibilities	10
2.3	Stakeholders, their interest, and priority [Adapted from [77]]	15
3.1	SOA and Microservices Incompatibility [Adapted from [16]]	18
3.2	Pattern/Solution Research Aggregation	23
3.3	Research Question 1 (RQ1)	27
3.4	Research Question 2 (RQ2)	27
3.5	Inclusion and Exclusion Criteria	28
3.6	Screened Studies	29
3.7	Sources classified according to pre-defined system	30
4.1	Fundamental Scale by Saaty [Retrieved from [73]]	39
4.2	Identified Criteria Comparison Matrix	40
4.3	Identified Criteria Comparison Matrix Normalized	40
4.4	Identified Criteria Priority Vector	40
4.5	Relevance Comparison Matrix	41
4.6	Scalability Comparison Matrix	41
4.7	Usability Comparison Matrix	41
4.8	Reliability Comparison Matrix	41
4.9	Benefits and Trade-Offs regarding the longitudinal perspective of value for the customer	43
5.1	Ubiquitous Language Definition	48
5.2	ADD Concepts [85]	48
5.3	Project Concerns	49
5.4	Project Constraints	49
5.5	Project Quality Attributes	50
5.6	Services and Responsibilities	52
7.1	Approval scale	74
7.2	About You Data Aggregation	76
7.3	Per Question and Overall Average V alues	80
8.1	Goal Achievement	81
A.1	Architectural drivers and non-functional requirements relation	89
A.2	Architectural drivers and functional requirements relation	89
B.1	Random Index (RI) Table [Adapted from [62]]	91
B.2	Relevance Comparison Normalized Matrix	91
B.3	Scalability Comparison Normalized Matrix	91
B.4	Usability Comparison Normalized Matrix	91

B.5	Reliability Comparison Normalized Matrix	91
D.1	Third Party Software in the Legacy System	96
E.1	Primary Studies	97
E.2	Primary Studies (continued)	98

List of Acronyms

API	Application Programming Interface.
FEI	Front End of Innovation.
ISO	International Organization for Standardization.
NCD	New Concept Model.
NPD	New Product Development.
QA	Quality Assurance.
QC	Quality Control.
RPM	Repeat Part Management.
SPC	Statistical Process Control.
VA	Value Analysis.

Chapter 1

Introduction

This document documents and describes the decision-making process inherently present in software design. Analyzing current academic and grey literature consensus was essential to identify good practices and patterns related to microservices and fulfill the identified goals.

This chapter introduces the thesis context, the problem, accompanied by an overview of the essential concepts, goals definition, the research methodology inherent to theoretical investigation, and the utilized approach to the solution development.

1.1 Context

Maintaining or developing new functionalities can get challenging when an application grows [81]).

Growth, most likely, is accompanied by increasing technical debt that limits scaling and software improvement, which is related to past limited (by time or resources) architectural decisions [60]. Netflix and Amazon's softwares were initially built on a monolithic-based architecture, which is excellent for quick launches, proof of concepts, and simple applications similar to RPM's initial development. However, monolithic architectures accumulated technical debt as these applications grew (number of users and, inherent, number of transactions and data) [71].

Each application identified its problems:

- Netflix pinpointed difficulty in maintaining data reliability as the cause of its service unavailability - their goal was to improve availability, scalability, and response time;
- Amazon's application development cycle was suffering from increasing bottlenecks which affected functionality evolution;

Netflix solved these issues by migrating to microservices, which simplified the development process. Microservices translated into multiple API services that communicated efficiently so that, in the case of failure, it would be easier to debug and, inherently, fix these issues. Creating and testing new services or functionalities would be a relatively low-impact process [66].

In Amazon's specific context, when migrating its two-tier monolithic application to microservices, their focus was on organization structure, Amazon used two fundamental rules: developer teams must be small and entirely responsible for their service. These rules explained further in chapter 3 (section 3.2) and the implementation of continuous delivery allowed Amazon to return value to customers faster and more flexibly [71].

RPM's developers identified difficulty in functionality expansion and difficulty debugging the application in case of unavailability. These are the same issues encountered in the explained case studies, which were solved by implementing microservices and considering migration patterns.

This thesis focuses on the implementation process of microservices on a greenfield application [84], specifically its inherent design decisions and trade-offs. As a result, a possible solution evolved by analyzing these existing services and splitting them, by responsibility, into independently deployed, cooperative, more minor services (Microservices) [81].

This architecture style allows for faster system build, horizontal scaling, and readability. However, its implementation is not superficial, given that it entails a detailed lifecycle plan, which includes responsibility concerns separation, decomposition, monitorization, continuous integration and deployment configuration, and service communication. These concerns make up the complexity of this architecture and explain the increased maintenance effort. Chapter 2 complements this contextualization.

1.2 Problem

Quality Assurance (QA) helps establish trust with consumers by ensuring specified requirements in the manufacturing process and, inherently, guarantying product reliability [35]. QA formalized practices have been encouraged by the International Organization for Standardization (ISO), which develops industry standards based on consensus [43]. On the other hand, as a segment of the quality measurement process, Quality Control (QC) relates to inspection, which is a more "operational" requirement [18].

As part of QC, inspections allow corporates to detect anomalies during manufacturing by identifying deviations from previously defined industry criteria [72]. The inspection concept represents the measurement and analysis process of a product's characteristics [72].

FARO is an imaging company that provides 3D digital measurement tools and software, including the CAM2 software, which suits repeat inspection plans and Statistical Process Control (SPC) analysis [30]. CAM2's list of features includes Repeat Part Management (RPM), which comprises RPM Control Station, RPM Control Server, and RPM Control Center. This system allows inspection programs to be published, run, and analyzed.

- Control Server: saves published documents (programs), related files, and results.
- Control Station: runs and shows a list of published documents (programs).
- Control Center: deals with result analysis; it shows every document present in the server, inspection notifications, and process warnings.

RPM, as CAM2, has been growing since 2018, when it was released. Four years later, RPM's three components have become difficult to maintain and modify. Migrating to a Microservices architecture presented an opportunity to achieve faster scaling and increase RPM's responsibility.

Decomposition to Microservices is a complex process, hindered by limited knowledge and lack of scientific consensus on decomposition strategies, and its evaluation, which is discussed further in section 3.5 [38] [51][52][17]. Therefore, it requires documentation exploration and research-paper analysis on design patterns, antipatterns, and decomposition techniques.

Furthermore, requirements must be analyzed and documented to identify clear objectives for the implementation process.

In figure 1.1 the relationship between CAM2 and RPM can be seen directly. As said previously, the control server acts as a database, the control center presents data in a visually intuitive way using Statistical Process Control (SPC), and the control station visually presents the inspection process. Furthermore, in section 2.2.3 there is complementary information about the CAM2 and RPM connection.

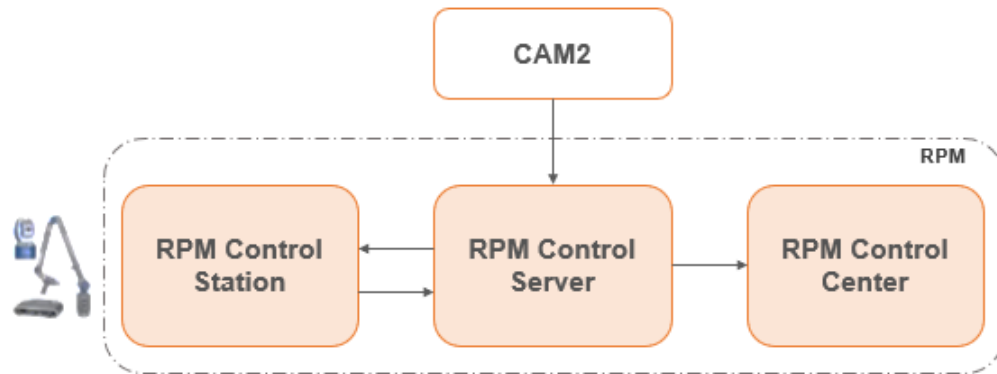


Figure 1.1: Relation between CAM2 and RPM

Figure 1.2 emphasizes this thesis target purpose, Repeat Part Management, also known as the legacy system, encompasses N functionalities, which must be identified and documented. Subsequently, by functionality analysis, a decomposition strategy and design is proposed for the system, which must be developed in accordance to the Microservices Architecture, and must be able to fulfill these defined functionalities. Figure 1.2 represents, at the top the legacy system which encompasses a N number of functionalities that must be considered, to be reproduced, when designing and implementing this decomposition.

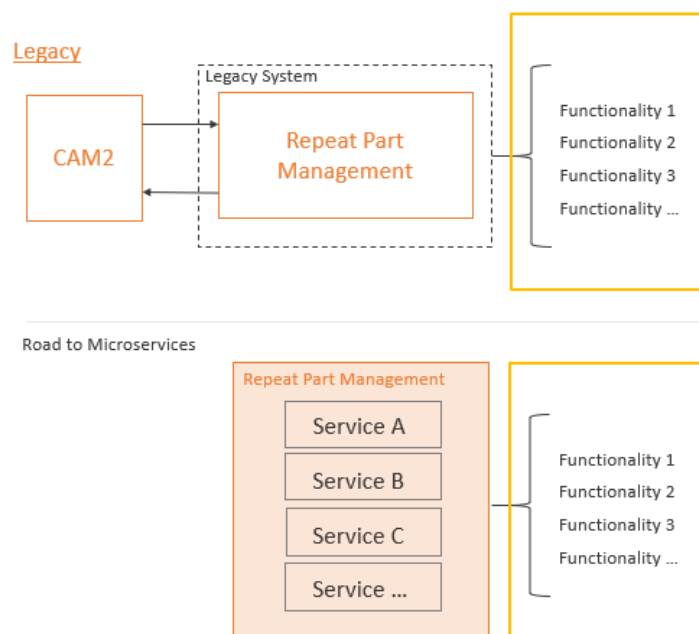


Figure 1.2: Problem Context Diagram

1.3 Objectives

RPM has grown over these last years and has gradually accumulated technical debt, hindering modification processes and continuous delivery. Also related to technical debt, increased tightness in coupling between CAM2 and RPM's components has decelerated development, which can explain an increase of breaking changes. Breaking changes translate code modifications or alterations that negatively affect other services or functionalities. These are this thesis' objectives:

- Tackle Microservices evolution: propose a decomposition strategy and design using the Microservices Architecture by utilizing practices and design patterns approved by peer-reviewed documentation.
 - While considering the legacy RPM application and its already available and maintained three components.
- After decomposing, validate design proposition, while considering the architecture's ecosystem and lifecycle: design (decomposition and requirement engineering), build, deployment, maintenance, and management.

Figure 1.3 presents further description on the thesis and project tasks. These can be correlated to architectural drivers. Requirements are defined by TN, where T is Task, TF if is functional task and N is an identifiable number,

These are the identified tasks, and their identification.

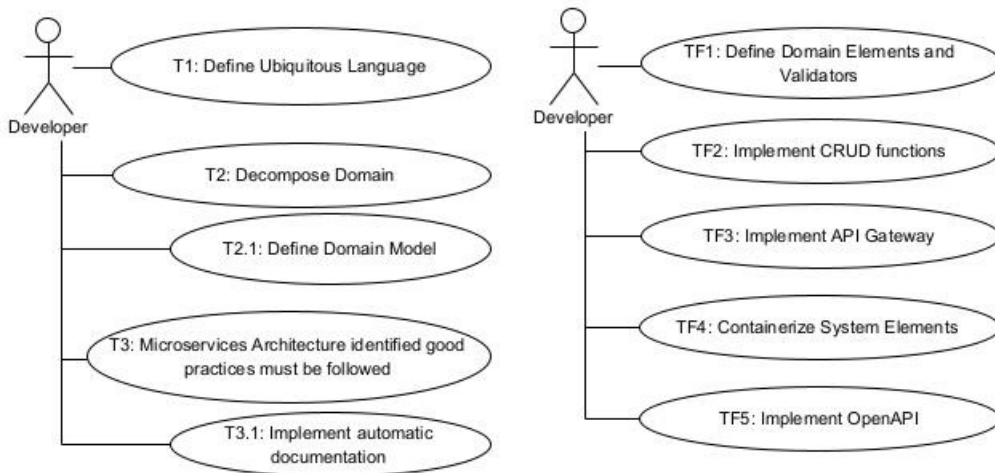


Figure 1.3: Thesis Tasks Overview

Implementation encompasses these requirements, which are equally important as to achieve success. Considering that this thesis focus heavily on decomposition strategies, it is crucial that the decomposition process is documented and related to the defined drivers, which will be specified in F.

1.4 Methodology

Given the initial stage of this project and its dependency on a legacy domain, it is essential to recognize application stakeholders, understand their motivations for this change, analyze the current solution's structure, and aggregate domain knowledge. At this phase, the problem must be identified and described.

Additionally, it is crucial to identify and gather techniques and good practices regarding Microservices' lifecycle, so it is possible to assert and avoid excessive implementation, given the project's initial state. Furthermore, developing decision-making documentation will help identify potential trade-offs inherent in alternative solution consideration.

A methodology was employed to ensure the solution's validity when filtering and analyzing field papers, ensuring that the several sources used to produce this document are reliable and adequate. In general, criteria were applied to used resources, such as having a clear preference for peer-reviewed sources (scientific articles and books) in lieu; and analyzing studies' abstracts, keywords, and introduction throughout the document's research process. The approach splits into these stages:

1. Initially, a breakdown of the surrounding industrial and quality assurance legacy context was performed and correlated with FARO and their hardware and software products.
2. A Narrative Literature Review (NLR) was carried out in section 3.3 regarding microservices' migration architectural patterns, which aimed to identify current practices, preventable mistakes, and the corresponding solutions regarding the total development cycle. It presents a taxonomy for these practices that allowed for further research planning and identifies research lacking in a crucial stage of development.
3. A Systematic Mapping Study was performed related to the design stage of the development cycle, specifically, a methodological literature review proposed by Alberto Sampaio that intends to gather and identify all documentation/resources concerning the search goal [74]. The Review's goal was to identify decomposition practices in microservices migration which is related to the design stage.

In each section related to research, there is an explanation of objectives and an introduction to methodology. After each review, a presentation of the followed approach is explained.

1.5 Document Structure

This document is split into 7 chapters:

This chapter provides project contextualization and thesis field of study. Based on the problem's definition, it presents the established goals, work methodology and approaches.

The Context chapter displays further domain and business model contextualization by specifying theoretical concepts necessary to understanding the software's business applications regarding quality assurance and repeated inspections. Furthermore, it presents the legacy system's current functionality, basic workflow, defining component responsibilities, and domain model. Chapter 2 also clarifies the relationship between CAM2 and RPM (and its' components).

Following, the State of the Art aggregates essential theoretical concepts regarding the microservices architecture, specifying a taxonomy of patterns and antipatterns and related

approaches to design. Furthermore, relevant information about the microservices architecture and its characteristics and used approaches. Specifically, in chapter 3 the Narrative Literature Review and the Systematic Mapping Study are documented.

Chapter 4 refers to value analysis, a theoretical description of the new concept development model, and, inherently, identified and defined idea and opportunity, and concept development. Additionally, it presents the Business Model Canvas, which relates to defining the customer segments, value proposition, channels, customer relationships, revenue streams, key resources, activities, partners, and cost structure.

Chapter 5 encompasses the Design phase of this thesis by describing how the design was performed, the considered alternatives and trade-off discussion.

Chapter 6 presents the Implementation phase, which specifies the considered alternatives. The architecture is documented by using the 4+1 model.

Furthermore, chapter 7 encompasses this thesis' evaluation strategy, which is validation by stakeholders, which entails the analysis of a developed questionnaire filled by FARO's, and specifically RPM's stakeholders, which are defined in section 2.3.

Lastly, chapter 8 includes general conclusions, specifying project goals achievement, threats to validity, and future work.

Chapter 2

Legacy Context

This chapter describes Repeat Part Management's existing domain and contextualizes the legacy application's main field of action, namely quality control. Furthermore, an overview of the legacy applications' structure is presented along with its correlation with CAM2.

2.1 Manufacturing Quality

This section presents critical concepts regarding the manufacturing quality process and its importance in facilitating domain understanding - specifically, terms such as Quality Assurance, Quality Control, Inspection, and statistical process control (SPC). This is crucial in understanding RPM's purpose and business model.

FARO offers a wide range of products, hardware, and software solutions. The company developed several measuring devices regarding the hardware offers, namely 3D-Coordinate Measuring Machines (CMM) to measure shapes and parts with high precision. It produces lasers, which provide non-contact measurement accuracy important in quality control processes.

This thesis focuses on CAM2 and, particularly, Repeat Part Management regarding the software offer, which is the target of the microservices migration.

FARO operates in different domains [32], such as: production, assembly, product design, modeling, construction, preservation, and forensic analysis; however, this thesis concentrates on quality control and inspection, given that it is FARO's main application field.

2.1.1 Quality in Manufacture

There are several definitions of product quality in manufacturing, given the different characteristics of product requirements throughout the industry. ASQ, an organization that focuses on providing expertise, tools, and solutions to its members in their industry fields [4], defines quality as a subjective concept and specifies two technical meanings [5]:

1. "The characteristics of a product or service that bear on its ability to satisfy stated or implied needs"
2. "A product or service free of deficiencies"

Both meanings reference customer satisfaction and product functionality by guarantying its conformance with provided and previously defined requirements. Nevertheless, ASQ adds that Joseph Juran, a renowned business consultant in quality management, defines quality as "fitness for use" [5].

Joseph Juran additionally outlined five factors to assure quality in manufacturing [47]: Who uses the product; How they use it; Possibility/Probability of danger; Consumers' and producers' resources; Consumer quality perception; From there, three components: planning, control, and improvement, relating to the industry's quality processes [22].

Quality Assurance (QA) corresponds with the planning component; it focuses on defect identification and prevention and relates to the International Standard Organization (ISO) 9000 standards, which defines this stage as confidence in requirement fulfillment [5]. This concept emphasizes the manufacturing process and its tools' quality. Additionally, the act of giving confidence translates to assurance.

Quality Control (QC), according to Joseph Juran [5], emerged to widen quality approaches. QC allowed for a decisive distinction between detection and prevention (or pro-active control) in quality processes, given their compliance with ISO standards. "Control" inherently implies corrective responses, which indicates a technical approach to quality - *inspections*.

2.1.2 Quality Control

Although assurance (QA) and control (QC) are inherently related, QC focuses on fulfilling the requirements during operations [46]. Monitoring the manufacturing process allows corporations to prevent additional costs by helping detect errors in the manufacturing process and, inherently, preventing further deviations. Additionally, it guarantees product quality before reaching the end consumer.

Quality Control must include monitoring performance, comparing goals with actual results, and having a reactive posture towards them, making it the top priority in the Quality Process components [46]. Inspections are made to compare these manufactured results to the previously defined theoretical values by choosing a sample, measuring, analyzing, and testing it [5].

Statistical Process Control (SPC) is a QC technique that uses statistical tools to check the product's "fitness for use." It requires randomness or variability in the choosing process of inspected samples [68]. By being consistently monitored and analyzed, these inspections allow businesses' to detect and react to conformance deviation [46].

2.2 Business Domain

In this section CAM2 and RPM are identified and their correlation is described explicitly, which complements the information in section 1.2.

2.2.1 CAM2

CAM2 Software platform is the FARO's solution suited for quality assurance and inspection routines, fitted to manage and analyze repeated inspections. As mentioned in the Quality Control section, enabling businesses to capture measurements and document them facilitates error detection and cost prevention. Furthermore, it encourages enterprises' informed decision-making by automating complete documentation and automating inspection routines and analysis with SPC.

Besides part inspection routines and quality control, CAM2 has other application fields, such as machines, tools, parts verification/calibration, and reverse engineering for prototyping, which are not crucial to the domain of RPM given its own context.

CAM2's listed features are [31]:

- Repeat Part Management: ensures inspection consistency by allowing defined inspection routines to run.
- Control Center: inspection trend analysis based on SPC.
- Simultaneous Measurement: allows for multiple device measuring.
- Metrology-grade Meshing: displays visually appealing metrology information.
- Scanning Profiles: fast selection of the suitable device and inspection routine to run.
- Geometric Dimensioning and Tolerancing (GD&T): GD&T-based results display.
- Universal CAD file Import.
- Alignment Wizard: Aligning the theoretical part with the measured one allows for comparison.
- Line Deviation Highlights: meaningful use of colors to highlight tolerance analysis.

FARO distributes two versions of the Software (Full and Probing). Table 2.1 provides a version comparison of features [29].

Features
Operate with contact devices
Operate with non-contact devices
Device relocation
CAD import
Constructions and dimensions
Coordinate systems
Alignments
GD&T
Point cloud editing
Point cloud analysis
Report Generation
Repeat Part Management (RPM)

Table 2.1: CAM2 features list

2.2.2 Repeat Part Management (RPM)

Repeat Part Management (RPM) is an application that enables pre-programmed repeat part inspections to run, then storing its results and analysis, which later facilitates trend analysis. RPM is available in InTouch, which manages application distribution, prerequisites, downloading, installation, updating, and execution according to the users' product key.

As stated previously, three applications that communicate to complete the quality process make part of this stand-alone application. Inherently, they have different responsibilities, which Table 2.2 specifies.

Component	Responsibilities
Control Server	Repository for programs (inspection routines) and report data. To interface this component CAM2 has a RPM tab.
Control Station	Loads and runs inspection programs that were published to the Control Server. Report generation, later stored in Control Server.
Control Center	Browser-based; it displays historical inspection data in meaningful graphs; it allows access to component reports and trend graphs.

Table 2.2: RPM's Legacy Responsibilities

Figure 2.1 presents the Repeat Part Management's domain model, representing the object's relationships - retrieved from the control server's analysis.

The document is the main object that encapsulates the theoretical representation of objects (*CadFile*), the list of created programs (*Inspection*) retrieved from CAM2, representing one or more inspection activities, such as alignments or feature measurements, and the document report template (*ReportTemplateFile*).

Traditionally, as explained previously in section 2.1.2, an inspection includes measurement activities, which is this object's goal by connecting it to the object's features (*Feature*); furthermore, these inspections are, consensually, repeated through the manufacturing process, which is represented by the object *Session* that represents each of the program's repetitions.

An object's *feature* represents a specific, measurable part of it, which will, then, be compared to the defined theoretical measurements. These measurements eventually define the actual part as in or out of tolerance to be further analyzed by on-sight technicians - the tolerance values (defined in CAM2) are kept in the *Statistic* object/objects for each feature.

The *Session* object, representing one repetition of the defined program, encompasses each session's *Report* according to the session's results. These results are calculated in runtime and kept in *Session Feature* and *Session Statistic* and used by *Report* and, eventually, by *Control Center*, which encompasses several analyzable displays of these data.

In appendix D you can find the legacy's solution third party softwares.

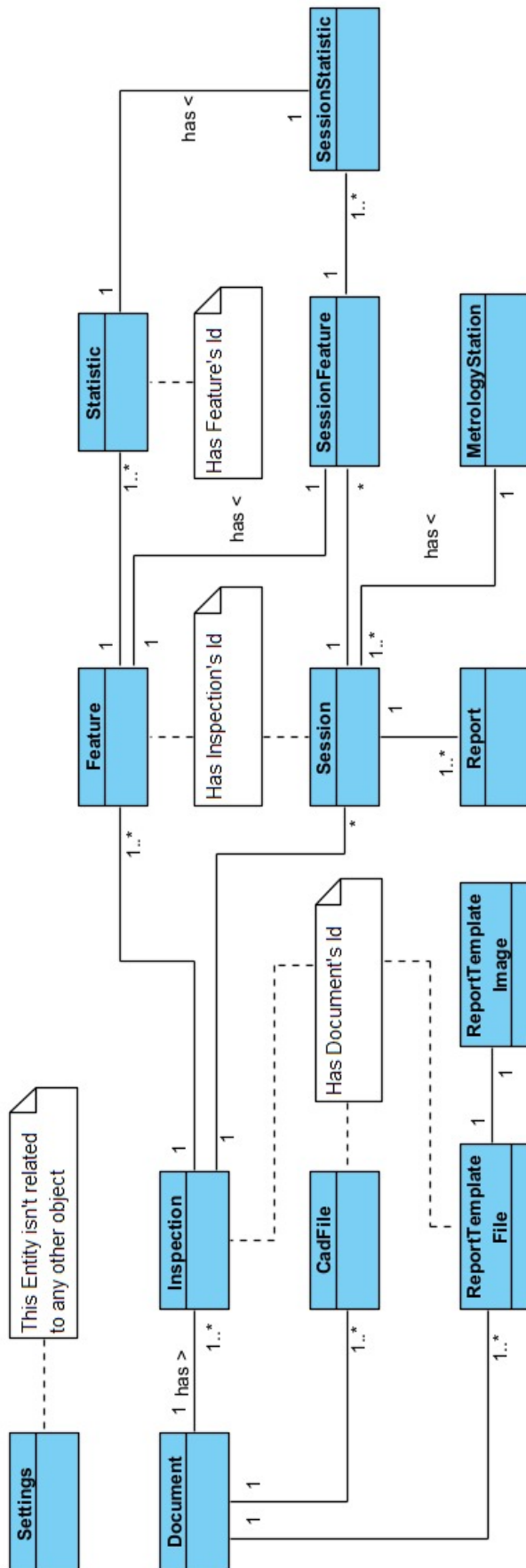


Figure 2.1: Control Server Domain Model

2.2.3 Relation between CAM2 and RPM

This section presents business-related concepts which help clarify the business and architectural relation between CAM2 and RPM.

It is adequate to consider CAM2 as a primary system, which allows the client's engineers to open *CAD files* or create further documents. By opening this *CAD File*, the engineer is automatically creating an *FCD File*, similar to a zip-like structure.

A crucial feature of CAM2, which allows the connection to RPM, is creating programs. A program is CAM2's concept term that corresponds with the *Inspection* concept (for further explanation, the used term will be *Inspection*).

Creating an inspection: When a user intends to automatize a repeated quality inspection they create an inspection on CAM2 by adding the desired tasks (one or more). A specific example would be:

In a given manufacturing company that intends to automatize the process of inspecting a part of a given product - e.g., in a company that produces screws that wants to automatize the process of inspecting a screw (related to the company's quality control process), it is required to open the *CAD File* of the specific product (CAM2) and creates an inspection for that screw, which can contain tasks like measuring the diameter of the screw's head (d), the length of the shank (c), the curve of the screw's thread (a), or others such as the alignment task. Figure 2.2 presents values d, c, and a.

Note: The *alignment* task overlaps the theoretical measurements present in the *CAD File* with the measured and actual object (in this case, the screw).

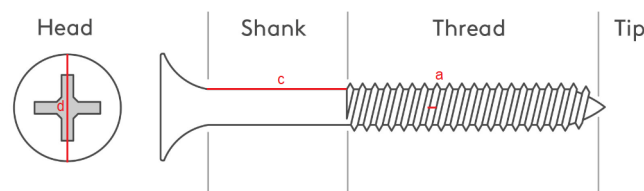


Figure 2.2: Screw [Adapted from [27]]

Besides having the specific *CAD File*, the *FCD file* can contain crucial complementary information and files, such as images, tolerance values, report information, and feature specification defined within CAM2. As shown in Figure 2.4, the client has the possibility of publishing this document (that must have at least one inspection). The client's feature: **publish a document** - saves the relevant characteristics of the document to RPM, such as *features, statistics, tolerances, report information, and inspections* (represented in figure 2.5 in CAM2's process). This feature is available through RPM Plugin in CAM2, as shown in figure 2.3.



Figure 2.3: RPM Plugin in CAM2

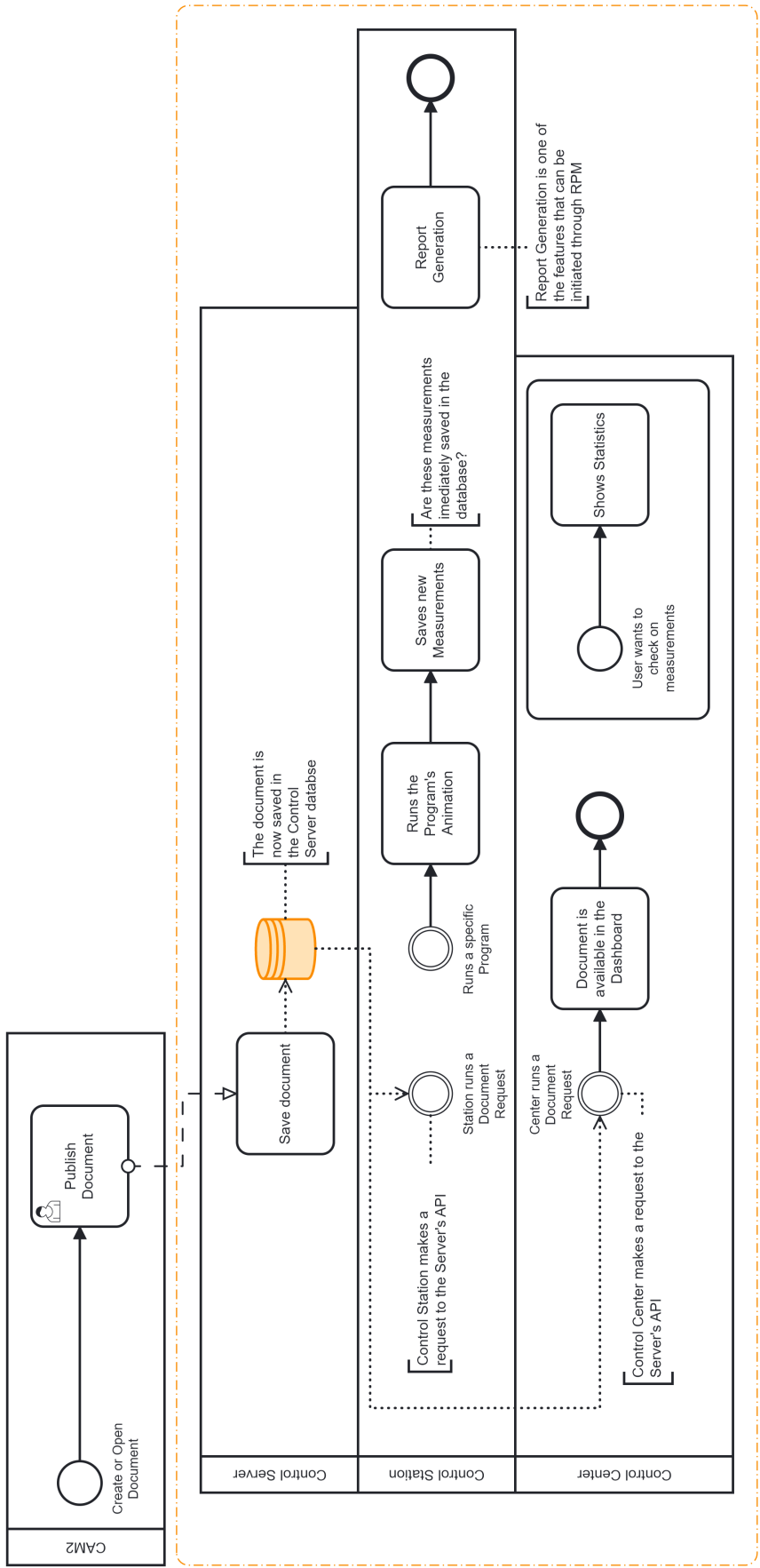


Figure 2.4: Feature and Business Flow (BPMN)

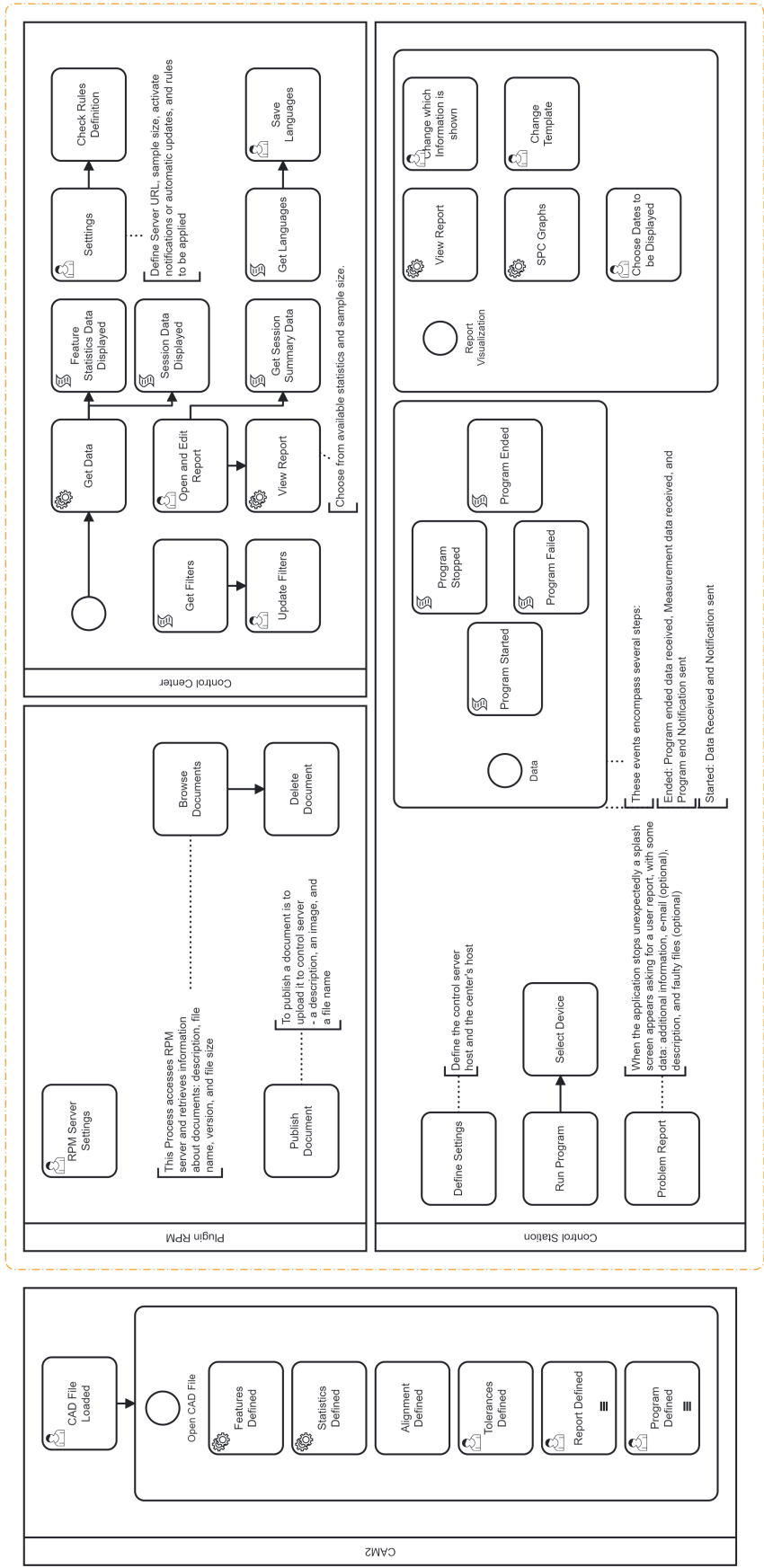


Figure 2.5: Application's Capabilities regarding the domain (BPMN)

2.3 Stakeholders Overview

Knowing the RPM software was developed for a technological fair and that its primary goal was to showcase new concepts in technology, it is possible to infer that it provoked a fast-paced development of a PoC that showcased the main features. This application evolved and grew through the years, which accumulated technical debt and created a Service-Oriented Architecture, which is deployed as a monolith.

This section intends to show the process of identifying problems in the legacy architecture. A sequence of techniques was applied to determine a list of doable requirements for project implementation.

An important part of identifying functional and non-functional requirements is recognizing relevant people who hold interest and influence over the project - in this case, Repeat Part Management. Larry Smith's approach to identifying stakeholders and pondering on their internal impact was adopted and is seen in table 2.3 [77]. The identified stakeholders are already filtered according to availability.

Name	Interest	Priority
Project Manager (RPM)	Reduce Overall Costs; Improve overall productivity;	1
Developers (including QAs)	Improve project's maintainability; Reduce technical debt; Project breaking changes related to deployment;	2
Project Manager (CAM2)	Reduce technical debt; Lessen the breaking changes charge to RPM's team;	3

Table 2.3: Stakeholders, their interest, and priority [Adapted from [77]]

Stakeholders identification allowed for general viewpoints identification (within the CAM2 and RPM team) and expectation recognition.

These are the current expectations about the new solution regarding management's identified issues:

- Develop a solution that enables scalability;
- The solution must foster the development of new features;
- Significant training needed for new employees (project readability);

These are the current solution problems identified by the development team:

- Greater difficulty in developing new functionalities;
- Greater difficulty debugging the system and inherently fixing issues;
- Considerable efforts are required to maintain code and resolve breaking changes;
- Significant training needed for new employees;

CAM2's interested parties have these expectations from migration:

- A total fusion of the two projects;
- Remove the RPM Plugin and insert its features onto CAM2;

Chapter 3

State of the Art

To design an adequate solution for the problem presented previously (Chapters 1 and 2), it is crucial to analyze the current knowledge on microservices migration and the approach's inherent patterns. This chapter intends to understand what is already known about these concepts and present them. Furthermore, within the Architectural Evolution section, the Service-Oriented Architecture (SOA) is presented, given the relation between the concept of service-orientation and microservices to clarify the distinction between the two architectural styles.

3.1 Architectural Evolution

Microservices derive from the concept of modularizing large monolithic applications into services, which is not unique to this architectural style. This section intends to define service-orientation and explain its evolution.

The definition of service, according to Thomas Erl, is a task (function) that may be performed relatively isolated from others; "it is a container of related capabilities.". Services comprise a body of logic, which can be invoked, if available. As a pattern, service-orientation states that a substantial problem would benefit from decomposition (into several minor concerns). Thomas introduces the concept of service composition, which corresponds to a set of services that provide the same functionality, coinciding with a business task or process [26].

These definitions relate to the Service-Oriented Architecture (SOA). SOA, mainly, differs from microservices in its implementation, but resembles it by principle [16]. Table 3.1 encompasses several application concerns, which diverge from microservices to SOA, which was screened from the Cerny et al. 2017 study to only encapsulate relevant content [16].

Deployment, as listed in microservices, is individual (per service), which explains the industry relation with containers and automated deployment [16]. Architectural decisions must abide by its scope - while SOA focuses on the enterprise, and developing reusable services - Microservices focuses on guaranteeing that its services are successful at the one purpose they intend to fulfill, explaining why SOA's services' are larger [16]. By distributing the system's responsibilities into services, microservices differ from the centralized management of SOA. Related to the individual deployment of services, data storage, in microservices, is created per service, SOA, by centralizing responsibilities (business rules), usually detains one large database. Furthermore, microservices are usually easier to understand concerning design [41].

Concern	Microservices	SOA
Deployment	Individual Deployment	Monolithic Deployment
Architecture Scope	Per Project	Enterprise
Service Management	Distributed	Centralized
Data Storage	Per Unit	Shared Persistence
Versioning	Integrated with architecture	Maintains different versions of services
Business Rules	Located in each individual service	Integrated in the Enterprise
Service Size	Small to medium size	Large infrastructure

Table 3.1: SOA and Microservices Incompatibility [Adapted from [16]]

3.2 Microservices

While Conway planned to list the flaws in distributed teams in his Law, Conway's Law also offered a theoretical basis for Microservices' implementation. It introduced awareness to communication efficiency, particularly communication channels and work division, iterative development processes, independent developer teams, and team size [67].

Four laws result in Conway's Law, which are [67]:

1. **System design replicate's the organization's structure:** Concerning the complexity of communication between developer teams, Conway implied that organizations' structure should reflect the system to prevent constant redressing and facilitate information flow.
2. **There is never enough time to do something right, but there is always enough time to do it over:** Information Flow affects early design decisions, which reiterates that a company's structure must be a facilitator to prevent further technical debt.
3. **Create independent subsystems to reduce communication costs:** By defining clear communication boundaries between services, its architectural decisions will be more suitable to its responsibilities and functionalities.
4. **Large systems tend to disintegrate during development, qualitatively more so than small systems:** According to standard Agile practices, architecture evolution should be iterative to guarantee error tolerance.

Conway, then, paved the way for microservices main characteristics by relating them to the organization's structure. Microservices are defined by their functionality aspect (services), communication mechanisms (REST/HTTP), modularization, and independence [37] [11].

Figure 3.1 graphically represents the effects of this architecture in the further development process by emphasizing Conway's Law in the organization's structure and the implementation of DevOps. Additionally, It represents these characteristics as essential assets in this architecture. Fundamentally, this figure advocates that to achieve successful software development within the microservices architecture, the business' organization must be aligned with the necessary services, and the responsibility for its Continuous Delivery process must lay with the teams.

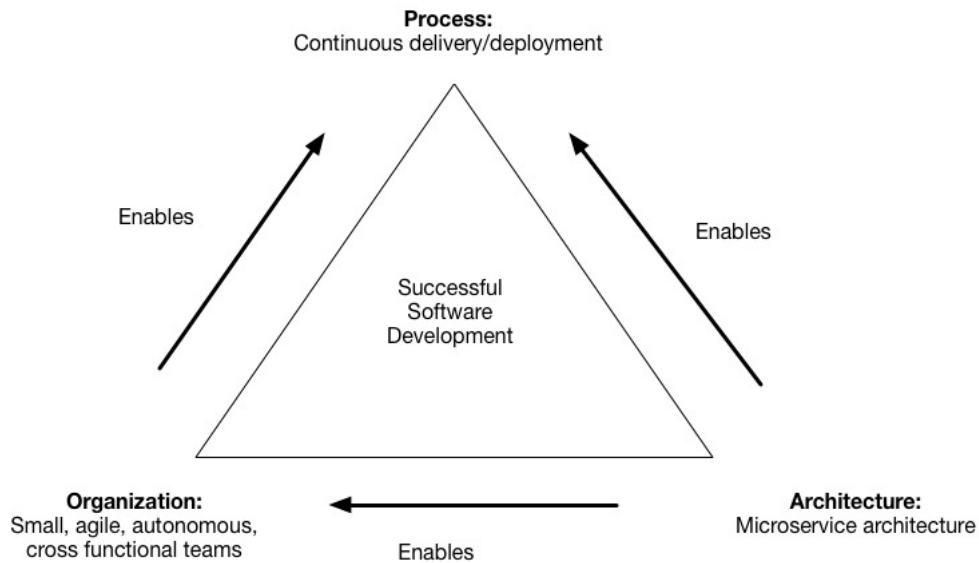


Figure 3.1: Microservices and its effect on development [Retrieved from [57]]

Application developers must build bridges with abstract business ideas to achieve a concrete system - or the design process deals with inherent software complexity [9]. The design of a given system relates to actual domain elements and their relations by defining properties and a given set of structures [9], which enables quality attributes requirements' satisfaction, emphasizing that the design process must consider theoretical quality attributes and trade-offs to achieve a successful application.

When migrating to a microservices architecture, a critical concern is the decomposition of an extensive application into a set of functional services [9]. Given that, it structures the system's build (unit of deployable modules) as a group of components representing these services. Each service corresponds to different business capabilities.

Services are standalone Application Programming Interface (API) that fit useful functionalities and are independently deployable. Additionally, microservices within the same architecture must be loosely coupled by guaranteeing that all interactions happen through the API. Limiting interactions allows developers to change these services without impacting others [9].

This section's intended purpose is to present crucial concepts and clarify the unique characteristics of the microservice architecture presented in the previous section 3.1. These are the relevant characteristics regarding microservices, which may not be unique to this architecture:

Modularity

Microservices focus on splitting an application into smaller components, which provides means for abstraction. Modularity, as component granularity, allows system management to become more efficient by splitting responsibilities. These modules are then independently deployed, which allows for the system to be developed with multiple technologies.

To properly achieve modularity, the legacy system must have defined services boundaries that must be fully independent, which means that each service must be completely responsible for their defined boundary. As described previously, the organization's structure must be aligned

to these defined contexts in order to enable cohesion. Modularity enables modifiability by making services replaceable. Robert C. Martin defined two principles that can be adapted and applied to the modularity concept [9] and [54]:

- **Single Responsibility Principle:** this principle emphasizes that component's responsibilities must be well defined since it promotes small services of increased stability [9];
- **Common Closure Principle:** by keeping domain concepts that are updated or required for the same reason in the same microservice, it is possible to avoid the nano services' antipattern;

Domain-Driven Design

Planning is essential when designing good software [44], particularly in Microservices. However, several questions are raised, specifically about services size. While some articles suggest that the number of lines of code is an appropriate measure for microservices partitioning, this idea does not consider that microservices allow for different technology use, which affects this number's variation [44].

Domain-Driven Design allows a better understanding of the high-level concepts that compose a system's architecture, complementing Business Capabilities by introducing several patterns.

The defined concept in section capabilities, business capabilities might not be enough to guarantee that these services can not be considered nano or mega according to the defined antipatterns [44]. These antipatterns may result in further orchestration and communication issues. DDD offers standards that allow developers to identify microservices and their sizes.

Figure 3.2 represents these patterns and how they relate, specifically:

- **Domain Model:** contains the different entities and processes that are involved parties in defined business functionalities.
 - **Bounded Contexts:** As stated previously, DDD defines business domain concepts as objects in a given system. Bounded contexts determine boundaries within a specific business capability/functionality. These contexts do not need to be isolated; however, to maintain their integrity, communication between dependent microservices should be well-defined (through APIs) [65].
 - **Aggregate:** defines logical boundaries in a specific context. Aggregates affect task coordination when in the same transaction. It implies that to change an object within an aggregate, it must go through the defined aggregate root - which must be a domain entity [44].
- **Ubiquitous Language:** As previously mentioned, the structure of an organization must be related to the software's architecture, which helps limit the necessary communication between teams since they shouldn't share domain responsibilities. However, this can still happen, corroborating the need for a ubiquitous language—furthermore, the need to communicate to other stakeholders also encourages this pattern.

Essentially, a set of domain experts creates a domain model, which must be encompassed in the domain knowledge. The domain model must use ubiquitous language and be applicable into every defined subdomain and to each related bounded context.

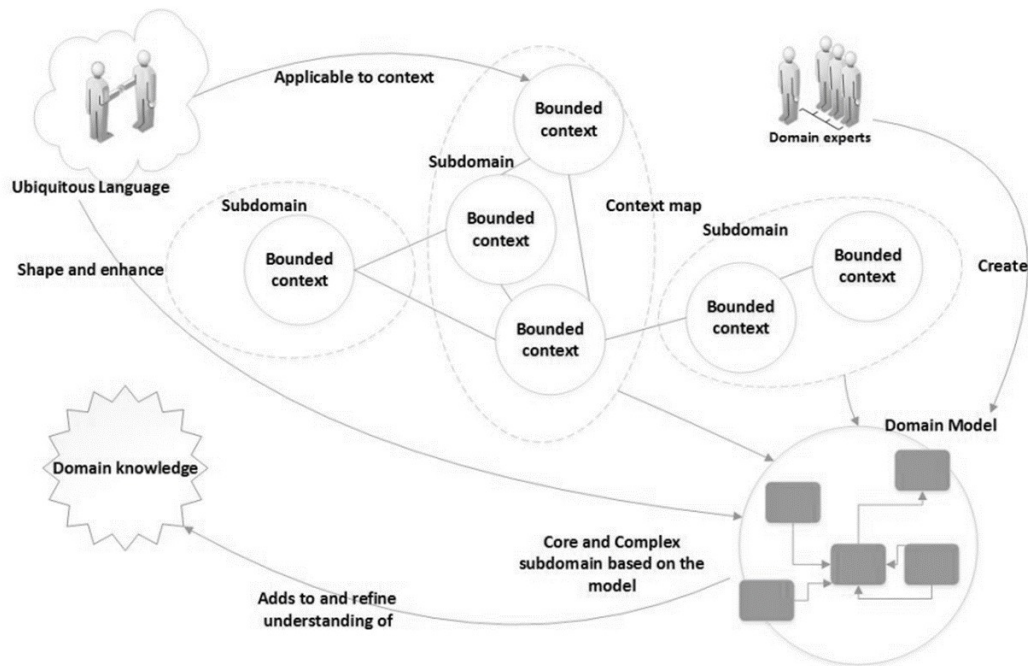


Figure 3.2: Domain Driven Design Overview [Retrieved from [44]]

Maintainability

Independent services come with a set of benefits, however maintenance becomes more complex. To maintain a system like microservices, testing and monitoring are essential to find potential sources of failure - which, in a microservices environment, can be the service, container, or network. This is leveled by the increased readability within the microservices scope, which makes code more reusable and easier to change.

Regarding data consistency, in an environment which stores data in different databases there are more risks, such as data redundancy.

Microservices' maintainability must be intentionally considered while developing the system.

3.3 Architectural Patterns Research

This section aims to identify current practices and patterns used in Microservice Architecture migration. It investigates five studies about applied patterns and recommended practices in microservices, which propose different taxonomies for them. Academic and grey literature were considered when researching this subject.

This review intends to gather knowledge on predictable microservice development problems to determine possible solutions and implementable patterns.

This analysis intends to provide a taxonomy of the recognized patterns. The following studies were identified during a research performed in January 2022, while only considering documents published until January 2018:

"On the Study of Microservices Antipatterns: a Catalog Proposal" - (Tighilt et al. 2020)

This document encompasses a systematic literature review of microservice-related documentation and an analysis of open-source projects (67) with the intent of cataloging the identified antipatterns. These antipatterns are presented according to a taxonomy based on the development cycle of a system (design, implementation, deployment, and monitoring). Furthermore, this study presents possible solutions, benefits of application (refactoring), and trade-offs for each antipattern. It also relies on an impact scale by assigning the architectural antipatterns a high or low effect on the system's quality.

"Migrating towards Microservices: Migration and Architecture" - (Carrasco et al. 2018)

Carrasco, Bladel, and Demeyer extrapolated architectural smells (antipatterns) by analyzing academia and grey literature publications - a total of 58 documents. These architectural smells are supported by a possible solution, which provides actionable information on Microservices migration. Furthermore, this 2018 study proposes a more straightforward taxonomy between architectural and migration smells/antipatterns.

"Design principles, architectural smells and refactorings for microservices: A multivo- cal review" - (Brogi et al. 2020)

This 2020 study proposes that architectural bad smells threaten microservices' principles. By performing a systematic literature review of grey and academic documents they propose a pattern taxonomy based on the process viewpoint splitting them into independent deployability, horizontal scaling, isolation of failures, and decentralization.

"Actual Use of Architectural Patterns in Microservices-Based Open Source Projects" - (Márquez et al. 2018)

Márquez e Astudillo's study extends previous documentation on microservices' patterns by analyzing known open-source projects (design and code review). Furthermore, it performs a comparison between Service-Oriented Architecture and Microservices. This study identifies 12 categories for pattern description: Internet of Things (IoT), DevOps, Front-End, Back-End, Orchestration, Migration, Communication, Behavior, Design, Mitigation, and Deployment.

"Development Frameworks for Microservice-based Applications: Evaluation and Comparison" - (Dinh-Tuan et al. 2020)

Using their own experience, Hai, Maria, Mora, and Felix evaluated Microservices' frameworks according to their features, design patterns, and performance. This study specified Integration patterns, Database, Observability, equivalent to other taxonomy's monitoring patterns, Cross-cutting concerns, which are comparable to the previously identified Deployment patterns, and Communication pattern, which only distinguishes between synchronous, asynchronous, and remote procedure invocation Fault Tolerance.

Table 3.2 gathers the identified microservices' patterns and which sources reference them. [82] managed to encapsulate the patterns in fewer categorizations and in accordance with the development cycle; therefore, patterns are presented using this approach. The Design patterns individual categories relate to architectural specifications and the event of the microservice's development cycle to which they qualify, specified in section 3.4.

	Brogi et al. [61]	Dinh-Tuan et al. [23]	Márquez et al. [53]	Carrasco et al. [13]	Tighilt et al. [82]	Total
API Gateway	Yes	Yes	Yes	No	Yes	4
API Versioning	No	No	No	No	Yes	1
Backend for Front-End	No	No	Yes	No	No	1
Circuit Breaker	Yes	Yes	Yes	No	Yes	4
Container	Yes	Yes	Yes	Yes	Yes	5
CQRS	Yes	Yes	No	Yes	No	3
CI/CD	No	Yes	Yes	Yes	Yes	4
Database per Service	Yes	Yes	Yes	Yes	Yes	5
Documentation	No	No	No	Yes	Yes	2
Logging	No	Yes	Yes	No	Yes	3
Monitoring	Yes	Yes	Yes	Yes	Yes	5
Service Discovery	Yes	Yes	Yes	No	No	3
Service Registry	Yes	Yes	Yes	No	Yes	4

Table 3.2: Pattern/Solution Research Aggregation

3.3.1 Research Results

These five studies aim to identify bad practices in microservices migration and development. Furthermore, they provide possible implementation strategies, patterns, or/and solutions to the identified architectural smells. After analyzing the documents, it is possible to conclude:

- Lack of research on microservices obstacles/architectural smells during development, given that most studies analyze published documents and maintained projects.
- All studies refer to systematic literature reviews (SLR) as a valid approach to software engineering research.
- Lack of patterns or techniques related to the decomposition stage of the development cycle.

This section specifies the identified patterns per the employed taxonomy (according to the development cycle). The patterns related to the design stage are specified in section 3.3.2. Furthermore, patterns related to the Monitoring stage are specified in appendix C since this thesis' goal is to focus on the design of the microservices.

These are the Implementation Patterns that were identified during the research:

1. **Documentation:** This practice is referenced in most reviewed sources, but it is not considered a pattern; in [23], the practice of documenting is still an advocated practice. Documenting is generally a good practice in any architecture; however, given service decoupling and independent deployment, microservice's dependencies and application overview can become lost, emphasizing this architecture's need for this pattern. Furthermore, considering Conway's law and team structure proposition, having no documentation might hinder communication between members. The recommended approach to documentation is an API in-code framework; otherwise, team members must be responsible for service documentation.

2. **Service Discovery:** Endpoint-based service interactions occur when microservices invoke others by using source code hardcoded locations. In this case, invokers would not locate these resources when scaling out by replicating services [61] and [82]. Service Discovery can dynamically update microservices' endpoints by tracking them with Service Registry. However, implementing a service registry may affect service performance [82] because it requires maintenance and might delay development.

These are the Deployment Patterns that were identified during the architectural:

1. **API Gateway:** Allowing communication between consumer applications and microservices, also related to user access, is essential when implementing microservices. Similar to previously defined *Endpoint-based service interactions*, not implementing an API gateway leads to direct consumer-microservice communication. In this situation, consumer services must maintain and manage endpoints by being aware of the system's structure [82]. Similar to Service Discovery, implementing this pattern requires implementing a new microservice that needs further maintenance and might slow down response times [61].
2. **API Versioning:** In cases where multiple versions of an API are necessary, such as major modifications, implementing API versioning guarantees that consumer changes impact is lessened. This pattern allows this by communicating the version in the URI or the header of a request. API Versioning also requires maintenance and documentation, which may slow down response time.
3. **Backend for Front-end:** In cases where there is a need for customizable user interfaces, such as Android and Desktop compatibility, the backend might not be in accordance with different performances. Backend for Front-end creates a backend for each environment, allowing for behavior and performance adaptation. Implementing this pattern requires considering if it brings value, given the time needed for implementation [6].
4. **Circuit Breaker:** *Wobbly service interactions* occur when a microservice A, which depends on one or more functionalities provided by Microservice B, does not prepare for potential failures of B. Not preparing for losses on potentially core functionalities may cause service failure and cascade failure if not contained. The circuit breaker is the most common solution by wrapping service invocations which are then forwarded and monitored. When the number of failed requests goes over previously defined values, it will instantly send an error message to the invokers [61].
5. **CQRS:** Command and Query Responsibility Segregation (CQRS) separates read requests from write requests by defining queries and commands in scenarios where performance is required in data readings or when service integration is needed. Commands change the state of a system, while queries only return results. This pattern emphasizes that methods can't return messages and change them. However, before implementing CQRS, it is essential to consider necessary team efforts [3].
6. **DevOps and CI/CD:** Development and delivery automated processes increase system agility by affecting software quality and reducing increasing delivery efficiency. Slower testing and deployment processes, error-proneness, and lack of coordination between microservice are consequences of no version-control, unit, functional or integration tests, automated delivery tools, or staging environments [13]. Pipelines must be configured and integrated independently in each microservice, facilitating system

tracking and debugging. Furthermore, testing is faster given the use of automated approaches. However, implementation requires significant effort and resources [82].

7. **Database per Service:** Shared persistence is discouraged by every referenced source. This antipattern occurs when microservices access and modifies the same database. [61] proposed three solutions that apply similar principles that limit microservice data access to only essential information.
 - Splitting the database requires less intervention; however, this approach isn't always possible to apply and may lead to increased efforts in maintaining data consistency.
 - Implementing a data manager microservice that becomes the only responsible service to manage the database.
 - The [82] can be recognized when microservices share too much information or cyclic dependencies are identified. Sharing persistence can signal this smell, which creates an opportunity for service merging [61]. However, this service may not always be applicable given that microservices should only fulfill one business capability [82], further explained in section 3.4.

3.3.2 Design Patterns

In this section the techniques/approaches to microservices' decomposition, associated with the Architectural Pattern Research in section 3.3, are identified and described:

1. Decomposing by Subdomain

- A subdomain is a Domain-Driven Design (DDD) concept related to a specific business' domain element [57]. The goal is to separate the domain model for each subdomain. Each subdomain is related to a bounded context, a DDD concept explained in section 3.4. Evans categorizes subdomains into three classifications: subdomains are considered as Core if they are differentiators and deemed the most valuable to the business, Supporting if they relate to core domains but are not differentiators; finally, there are Generic subdomains, which are usually not directly associated with the business domain and usually implemented independently [28].

2. Decomposition by Business Capability

- Business Capabilities capture stable functionalities that generate value for a company by disregarding how they are executed [70]. Chapter 2 provides context on FARO's field, which becomes essential when identifying RPM's business capabilities. The result of the identification process is usually not sharp. Addressing capabilities systematically guarantees that nano and mega services antipatterns are not a potential threat to the architecture's success [8]. Microservices should serve the identified business capabilities individually, encouraging system maintainability and modifiability. These characteristics result from independent deployments, a more straightforward debugging process, and minor-impact improvements regarding other services [70]. These business capabilities must later correspond to developer teams' responsibilities regarding Conway's Law to ensure the identified benefits [70]. The organization must undergo purpose, business processes, and structure analysis to identify these capabilities, beginning by identifying a high-level domain model [57].

3.4 Decomposition Research

This section intends to complement the architectural pattern research by identifying and aggregating existing approaches and techniques to decomposition by performing a Systematic Mapping Review (SMR), which is a common practice in software engineering research, by applying Alberto Sampaio's method [74]. Figure 3.3 represents the proposed process, which is split into six stages. While the order of these should be respected, it is possible to return to any stage and revisit a decision. These are the steps description:

1. **Protocol and definition of the research questions:** this stage's outcome is a rigorous protocol, which allows other researchers to replicate it;
2. **Conducting the search for primary studies:** this stage aggregates all papers and must encompass all relevant information;
3. **Screening:** screening focuses on selecting adequate papers by applying the previously defined criteria;
4. **Classification system:** the purpose of this stage is to organize the selected papers following the defined research questions;
5. **Data extraction and aggregation:** after organizing the studies in the previous step, this step focuses on extracting and recording data;
6. **Analysis and Report:** analyze the map, draw necessary conclusions, and describe possible threats to validity;

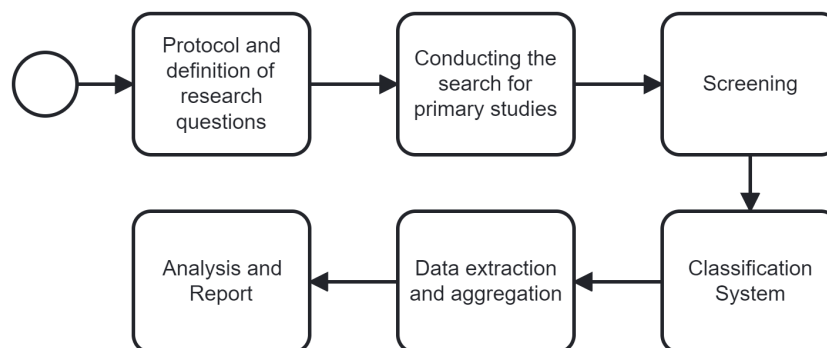


Figure 3.3: Systematic Mapping Study Process [Adapted from [74]]

3.4.1 Research Protocol Definition

As described previously, this section establishes the mandatory protocol and research questions regarding this mapping study. By clearly establishing a protocol this study becomes reproducible [74].

Boundaries

Having already performed a Narrative Literature Review (NLR) regarding architectural patterns and smells allowed for a scoping of existing literature regarding microservices. Furthermore, the crucial resources are reports on migration processes, resources focused on design patterns and smells from decomposition strategies, and similar Systematic Literature Reviews.

Given the scope of the thesis study, only digital libraries will be used, specifically B-On, IEEE Xplore, and ACM Digital Library, given the advanced search feature applicable to their repositories. This review must only consider articles published after 2018 to certify that only the latest techniques and approaches are regarded.

Research Questions

As reported previously in section 3.3.1 after scoping microservices it was possible to conclude that there was a lack of research regarding methodological decomposition patterns and issues regarding the migration process. However, NLR can be inadequate in gathering knowledge and understanding a more specific area, such as decomposing strategies in a microservices architecture migration process.

Sampaio suggests a commonly used framework for building evidence-based practical questions, which should be searchable: PICOCS (Population, Interventions, Comparison, Outcomes, Context) [74]. By using the PICOC framework, the outcomes were tables 3.3 and 3.4.

The research question in table 3.3 focuses on the main research conclusion from the NLR by concentrating on identifying common techniques in decomposition processes. While the research question present in table 3.4 is focuses on avoiding issues related to decomposition.

Research Question	What are the most common techniques for decomposition in microservices migration processes?
Population	Articles about actual migration processes, resources focused on decomposition and related patterns or smells.
Intervention	Identify techniques, methods or approaches related to the decomposition of an established system.
Comparison	The existing and common approaches are Decomposition by Business Capability or by Subdomain.
Outcome	Mentioned and applicable approaches or techniques and the details of applicability to projects.
Context	Articles regarding microservices migration processes and design.

Table 3.3: Research Question 1 (RQ1)

Research Question	What are the preventable architectural and implementation smells of decomposition processes in microservices?
Population	Articles/Reports on actual migration processes, and resources regarding architectural smells and other related issues.
Intervention	Determine common issues and smells within the decomposition process in the migration to a microservices architecture.
Comparison	Already analyzed smells and preventable mistakes are mostly related to implementation.
Outcome	Identifying avoidable problems and their specific solutions.
Context	Articles regarding microservices migration processes and design.

Table 3.4: Research Question 2 (RQ2)

Inclusion and Exclusion Criteria

During this stage, it is crucial to define criteria per the specified research questions. Sampaio suggests coding every criterion and giving it a specific identification to more directly guide the screening process [74]. Table 3.5 encompasses the defined Inclusion (I) and Exclusion (E) Criteria to be applied to the found studies.

Criterion	Description
I1	Studies that describe the migration process to microservices specifically.
I2	Resources that provide techniques or approaches regarding decomposition of legacy systems.
I3	Resources that identify architectural smells and patterns regarding the microservices architecture.
E1	Resources published before January 2018.
E2	Resources that are not written in English or Portuguese.
E3	Studies that do not mention microservices migration processes.
E4	Studies that are not available in full (pdf format).
E5	Studies that focus on brownfield migration processes.

Table 3.5: Inclusion and Exclusion Criteria

Search String

The search strategy encompasses the definition of a search string that aligns with the previously defined criteria and research questions, as intended by Sampaio's methodology description [74]. This string specifies mandatory words in a given study. In this specific case the used words (and their inherent derivatives - e.g. decompose, decomposition, decomposed) and synonyms which may be used in the same context are: microservice, migration or refactoring, decomposition, strategies or techniques or practices, patterns, design or architecture.

During the research process within the used research libraries, this string is applied to the advanced search mechanisms from the chosen digital libraries:

[microservice] AND [migrat* OR refactor*] AND [decompos*] AND [strateg* OR techni* OR practic*] AND [pattern*] AND [design* OR architectur*] AND [Publication Date: (01/01/2018 TO 08/31/2022)]*

3.4.2 Conducting the Search for Primary Studies

The second step to the Systematic Mapping Study method is to apply the defined search string to the chosen digital libraries, specified during the review protocol definition in section 3.4.1. As specified in the boundaries subsection of the protocol, only accessible resources published after January 2018 are considered. This search was conducted on the 28th of August 2022 and its results were: ACM Digital Library with 8 results, IEEE Xplore has 4 results, and B-On had a total of 7 results, In total the primary studies search found 19 studies, given that there weren't any duplicates. The full list of found resources is in appendix E.

3.4.3 Screening

This step's outcome is a list of resources/studies that adequate according to the defined inclusion and exclusion criteria. This exclusion task encompasses the analysis of the title, abstract, and keywords of each study to which the criteria are applied. In case of uncertainty it is possible to consider reading the conclusion. In table 3.6 it is possible to analyze the screened list of studies.

ID	Title	Author(s)	Publication Year
S1	A dataflow-driven approach to identifying microservices from monolithic applications	Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao	2019
S2	Analysis of the criteria adopted in industry to extract microservices	Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello	2019
S3	An experience report on the adoption of microservices in three Brazilian government institutions	Welder Luz, Everton Agilar, Marcos César de Oliveira, Carlos Eduardo R. de Melo	2018
S4	Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices	Thelma Colanzi, Aline Amaral, Wesley Assunção, Arthur Zavadski	2021
S5	GreenMicro: Identifying Microservices From Use Cases in Greenfield Development	Deepali Bajaj, Anita Goel, S. C. Gupta	2022
S6	How Can We Cope with the Impact of Microservice Architecture Smells?	Xiang Ding, Cheng Zhang	2022
S7	Identification of microservices from monolithic applications through topic modelling	Miguel Brito, Jácome Cunha, João Saraiva	2021
S8	Microservice Decomposition via Static and Dynamic Analysis of the Monolith	Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, Dan Kröger	2020
S9	Modernizing Legacy Systems with Microservices: A Roadmap	Daniele Wolfart, Wesley K. G. Assunção, Ivonei F. da Silva, Diogo Domingos	2021
S10	Unsupervised learning approach for web application auto-decomposition into microservices	Muhammad Abdullah, Waheed Iqbal, Abdelkarim Erradi	2019

Table 3.6: Screened Studies

3.4.4 Classification System

At this stage a classification system must be presented with the intent of classifying the screened studies. Furthermore, classification must match the previously defined research questions [74]. Figure 3.4 presents the adequate categories to classify and analyze data.

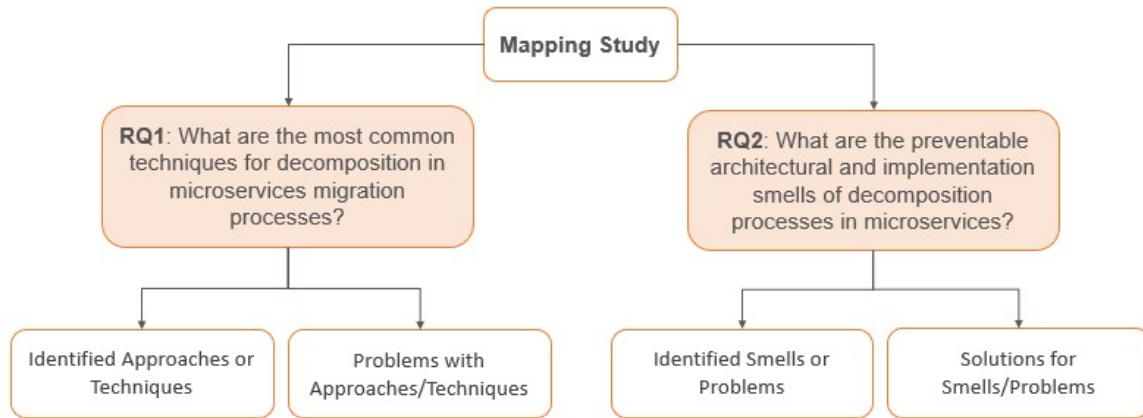


Figure 3.4: Classification System

3.4.5 Data Extraction and Aggregation

This stage intends to extract and record data from the previously screened studies and to classify it according to the previously defined system. From research question 1 (RQ1), the documents that mention approaches and techniques and the problems related to these approaches. Research question 2 (RQ2) splits into identified preventable smells or problems and their solutions.

Table 3.7 organizes sources by defined categories, in accordance with the research questions.

	Categories	Source ID
RQ1	Identified Approaches or Techniques	S1, S2, S4, S5, S6, S7, S8, S10
	Problems with Approaches/Techniques	S1, S2, S6, S7, S9, S10
RQ2	Identified Smells or Problems	S2, S3, S4, S6, S9
	Solutions for Smells/Problems	S2, S6, S9

Table 3.7: Sources classified according to pre-defined system

3.5 Analysis and Report

In this section, and as the last step of the proposed methodology, data was extracted, presented, and analyzed. It intends to answer the previously defined research questions, which is the reason behind its organization, which means that, this section will be split by research question. In each segment both categories will be analyzed. Throughout this analysis, studies are referenced through the unique identifiers presented in section 3.4.3.

Although some studies are referenced in more than one category, to avoid just paraphrasing due to category inherent similarity, some sources have been reported only in one section.

Every source study identifies microservices migration as a crucial "prerequisite" [51], which is expected to guarantee the microservice's modular scope, given that the migration process' success is highly dependent on the chosen and appropriate approach.

3.5.1 Research Question 1 (RQ1)

This research question encompasses the identified approaches and problems within those or other approaches whether by applying them into a case study, analysing already existing and implemented systems, or implementing decomposed systems. It is relevant to mention that most studies reference the previously identified decomposition patterns (section 3.3.2) as a foundation for the proposed methods.

S1 proposes a dataflow-driven approach to decomposition that encompasses 4 steps:

- Requirements definition: this step only entails functional requirements (business logic).
- As this is a dataflow-driven method, the processes' flows must be specifically defined (as this is a crucial step for decomposition).
- Identify dependencies between processes and datastores.
- Decompose system and identify candidate microservices.

This approach was then implemented in a case study project, which raised some potential limitations regarding heavy reliance on dataflow diagrams, which raises concerns on their definition and management, omission of other important factors, such as frequency of communication and performance, and method efficiency.

S5 proposes a methodology for decomposition in greenfield systems based on use case analysis, given the lack of legacy code. This study applies this approach to four applications which demonstrated successful results. It focuses on assessing functional requirements and clustering associated use cases by then applying a clustering algorithm to a previously defined similarity matrix. S5 references a lack of scientific consensus on decomposition acceptance criteria (and evaluation) [7].

S7 proposes a microservice identification approach through topic modelling (domain defined concepts). One of this models' concerns is modularity quality (conceptual and structural), which is relevant within the microservices scope. This study also references lack of efficiency in its process related to the vast amount of work regarding permutations, due to their abstraction limitations [10].

S8 proposes the visualization of the monoliths (legacy system) live tracing to identify bounded contexts via dynamic analysis. This is done by employing a static software analysis tool to map source code packages for identifying "ambiguities in the corresponding service boundaries" [49]. This study does not refer any limitations with its work.

In a more traditional representation, S9 proposes that the starting point for decomposition is responsibility definition (for each microservice) and promotes the decomposition by business capability approach. Furthermore, it encourages the use of the Single Responsibility Principle (SRP), which is predictable due to microservice's modular scope [86].

S10 proposes an automatic decomposition process for web applications that specifically targets scalability and performance system characteristics. This method accesses application logs and from them proposes microservices mapped URLs. This method was implemented in an actual web application on a cloud infrastructure. However, implementation was not

efficient at all and needed several experiments in order to formulate a possible answer, which is reflected in the conclusion: "Building an automated solution to improve the performance of a legacy monolithic web application is a challenging task", because, as stated previously, building successful automated support platforms is not probable because it always omits specific system characteristics, whether that is performance or scalability [1].

Most proposed non-traditional studies require a lot of planning and documentation, which is not particularly efficient and, therefore, not attractive for companies who wish to accelerate implementation. This issue was actually raised by [51].

3.5.2 Research Question 2 (RQ2)

This research question will focus on addressing issues with migration processes and, if relevant, their existing/proposed solutions to mitigate the effects, based mostly on applied specialist surveys and systematic mapping studies.

S2 focuses on considered criteria when decomposing a legacy system into microservices, it generally reports a lack of software support for this process and states that "it is unlikely we will be able to design appropriate techniques and tooling support for microservice extraction". Furthermore, as it is relevant the most analyzed characteristics were (by order of relevancy) [14]:

- *Cohesion* which specialists report only analyzing it manually and being a key factor for identifying domain entities.
- *Coupling* which specialists report having difficulty analysing dynamically, due to lack of software support, as previously mentioned.
- *Requirement analysis* is the third most important criteria, specifically specialists focused mostly on functional requirements and mainly their performance, regarding non-functional.

Most of this study's smells focused on lack of supporting software for the modularity scope of microservices decomposition.

S3 analyzes the decomposition and migration process of three governmental systems and surveying participants. It firstly states that the bigger problem with this architecture and its implementation was automation, given the necessary requirements for maintenance, even while proving to be beneficial for these systems. They state that management's involvement with migration will facilitate and benefit decomposition, which inherently increases the chance of migration success. Furthermore, S3 reports a lack of scientific consensus on decomposition and its evaluation criteria (which some identified source studies tackle) [52]. S1 corroborates this limitation by stating that there are no comprehensive metrics/methods that support decomposition or system evaluation [51].

S4 tries to analyze a potential bridge between practitioners and theoreticals with a survey designed for industrial and academic investigation. This study identifies a close relation between what is applied with what is suggested by academic researchers. It states that the industry opts for an incremental approach to migration and usually decompose systems by business capability, which is related to other identified methods inefficiency (stated in 3.5.1) [17].

General lack of tools in software migration compel practitioners to work alongside (and closely) with theoreticals [52] [17].

S6 analyzes the impact of architectural smells in the microservices architectural type by performing a systematic review. This study reinforces the single responsibility principle (SRP) and modularity scope of this architecture by suggesting that decomposition must be carefully planned and that microservice independence management must consider data migration to ensure migration success. These solutions try to mitigate mismanagement, macro and nano services, and high coupling.

S9 promotes a microservices roadmap that mentions every step of the implementation process, this research question addresses decomposition difficulties, which is specified as one of the steps to the process. This study suggests that incremental implementation is the best approach by specifying that "due to the complexity of this activity, practitioners will learn how to do it better for each microservice decoupled from the legacy" [86].

Chapter 4

Value Analysis

This chapter presents this thesis' primary goal Value Analysis (VA). It is relevant to state that *value* is a subjective term when considered from different perspectives, such as consumer, manufacturer, and designer ([80]). Nevertheless, value must be measurable and possible by pondering expectations.

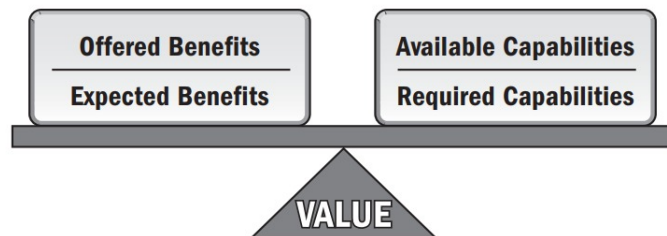


Figure 4.1: Value Concept [80]

Lawrence Miles, the founder of the value identification methodology, stated that value is influenced by *performance* and *cost* ([59]). Consequently, the value increases when the cost decreases while maintaining performance or increases within the consumer's acceptable price range. Then, performance and cost relate to the product's functionality to raise the value to customers.

However, Thiry states that value has evolved to a more "customer-oriented notion" ([80]) and defines it with *performance/fitness* and *resources* (product's cost, human resources, and capabilities).

Defining and evaluating a product's value is relevant given the market's characteristics, mainly competition ([59]), to maintain the product's long-term relevancy.

4.1 Innovation Process

Koen states that the innovation process (New Concept Model (NCD)) entails three distinct stages: or Front End of Innovation (FEI), New Product Development (NPD), and Commercialization ([48]).

Figure 4.2 identifies three components that define NCD, namely:

- The engine is fueled by effective leadership and the internal aspects of the company, although research hasn't yet corroborated this factor as critical to success ([63]).

- In the above layer, the FEI stage defines its five key elements, further explained in section 4.2.
- The circle's border represents external factors that affect every stage of NCD, such as strategies, competition, and organizational structure.

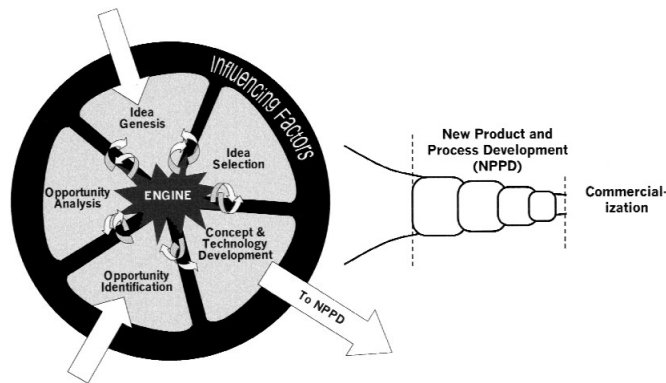


Figure 4.2: New Concept Development Model [Adaptation from [48]]

Front End encompasses all activities before New Product Development, although they are complementary stages. Contrary to the NPD stage, which entails formal actions, the front end is usually unpredictable and unstructured.

4.2 Front End of Innovation (FEI)

As a part of the New Concept Development Process, the Front End of Innovation (FEI) represents every development element until the concept's approval. FEI does not limit element definition order or combination, which means that redoing activities is usual and can prevent further costs and project cycle time growth. These characteristics are possible given to the tight coupling between elements ([39]).

4.2.1 Opportunity Identification

Opportunity identification in a given business field translates into the process of detecting previously unknown market deficiencies. Deficiencies range from competitive threats to technological advantages to production improvements affecting cost and time variables. It should also encompass the required resources to achieve effective and efficient concept progression ([39]).

Given its uncertain characteristics, Opportunity Identification's essence stands on proper methods and sources that may be formal or not, such as road mapping, technology trend analysis, competitive intelligence analysis, customer trend analysis, market research, and scenario planning.

These methods focus on identifying teams' insight, capabilities, and skills about the project, organizing business strategies to recognize competitive threats, and envisioning the uncertain future in a systematized approach ([39]).

As explained previously in section 1.2, RPM's developers have focused on maintaining the solution; however, developers have identified issues related to accumulated technical debt throughout the continued development.

Other companies that have already gone through this migration process implemented microservices' to tackle common issues, such as maintainability and modifiability. Therefore, and given the identified references of successful restructuring processes, this migration presents itself as a solution for FARO.

4.2.2 Opportunity Analysis

This element explores identified opportunities to assess if they are worthy of pursuit by gathering market interest and technological information. These efforts can be a part of a formal process or iterative to reduce the concept's attractiveness uncertainty and assess business capability ([39]).

Methods and approaches are the same as opportunity identification, although they are focused on resource management and detail on attractiveness in Analysis. Opportunity Analysis encompasses ([39]):

- Strategic Framing: contextualize the new concept into the company's market.
- Market Segment Assessment: framing the identified opportunity into the defined market segment.
- Competitor Analysis: focuses on researching and planning for competitive advantage and determining competitors.
- Customer Assessment: targets customer problems.

It is typical to return to this element to complement identified knowledge.

Throughout the Context and State-of-the-Art chapters (chapters 2 and 3), the reasoning behind choosing microservices as a solution for presented legacy system problems is presented.

The main challenges are scalability, modifiability, and maintainability, which hinder the development process of the application. Developers have identified increased latency in creating new functionalities and cloud implementation limitations regarding system complexity. This solution aims to tackle accumulated technical debt by facilitating the application's readability and deployment process, ultimately facilitating modifiability and functionality delivery to clients.

4.2.3 Idea Genesis

By Cambridge's definition, genesis is "the origin of something" ([34]). Accordingly, Idea genesis represents the birth and development of ideas and is a relatively unstable element of NCD. It benefits from functional teams and stakeholders' direct participation in the iterative discussions that may affect opportunity identification, emphasizing that NCD is a nonlinear process ([39]).

This element must encourage participants' creativity, specifically by pondering particular ideas or concept deficiencies. Idea Genesis can happen through formal activities such as

brainstorming or, in contrast, through communicated failed attempts in developments or specific customer requests.

Several steps are necessary to achieve the proposed solution, specifically:

- Aggregate information on good practices and patterns regarding the microservices implementation;
- Map the legacy application's architecture and identify its issues;
- Define and describe good practices regarding RPM's decomposition and define the microservices - additionally, this must encompass a responsibility definition for each service;
- Implement at least two microservices using the identified patterns, determining trade-offs by considering service communication, data consistency, and established system issues;
- Determine good practices regarding the implementation of monitorization and continuous documentation and implement them by identifying trade-offs;

4.2.4 Idea Selection

Ideas are typically abundant in businesses, which creates a need for methodologies that support idea filtration. However, having limited information in the early stages hardens establishing a formal decision process ([39]). The remaining elements of FEI are necessary to make better decisions by identifying influencing factors and new directives from the company.

The decision, at this stage, encompasses some risk, given the emotional characteristics of the initial trigger, which usually is an individual's judgment ([39]). However, ideas can fade if businesses do not allocate resources, such as funding or human resources, representing management support.

Businesses must ensure expectation management to maintain a positive mindset and approach by encouraging creativity and making ideas more attractive.

Saaty defined the Analytic Hierarchy Process (AHP) in 1980, used for decision-making by enabling the determination of idea selection and the reason behind it ([73]).

AHP can be partitioned into four steps ([73]):

1. First, define the problem and determine which decision needs to be determined.
2. Next, structure the decision hierarchy tree, as seen on figure 4.3, by dividing it from top to bottom as:
 - Goal: Choose the methodology to use;
 - Criteria: The defined criteria are also present in the Experimentation and Evaluation chapter (chapter 7) - relevance, scalability, usability, and reliability;
 - Alternatives: The choices of methodology to fulfill the proposed goal;
3. Construct comparison matrices that evaluate each criterion in comparison with another;
4. Finally, apply calculations regarding priorities to every defined criterion;

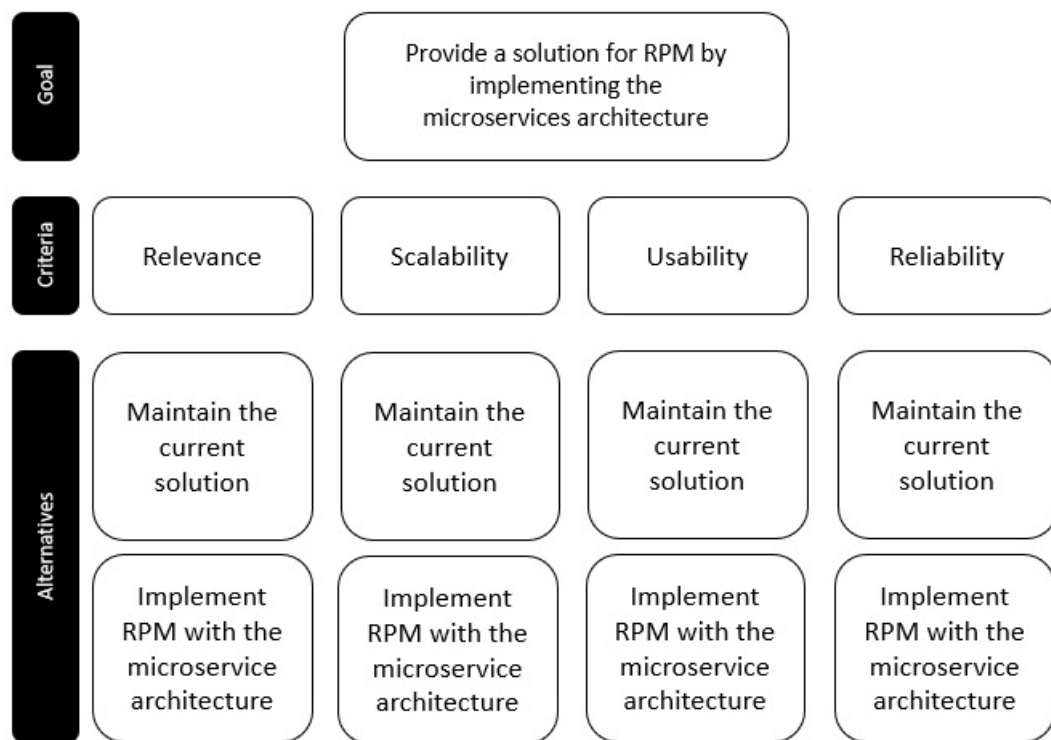


Figure 4.3: Decision Hierarchy Diagram

Image 4.3 define criteria that can be further specified as:

- **Relevance:** determining if the identified solution addresses the legacy issues;
- **Scalability:** does the solution address scalability;
- **Usability:** considering system readability and its inherent modifiability characteristic;
- **Reliability:** considering downtime is lower than up-time and if these characteristics are monitored;

The next stage focuses on comparing these criteria, however AHP focuses on creating a tangible (fundamental) scale for idea criteria (table 4.1). This scale is applied into table 4.2, which is then normalized in table 4.3 for further calculations.

Level	Definition	Description
1	Equal Importance	The two methodologies contribute equally for the goal
3	Weak Importance	It is possible to identify a small improvement when comparing these methodologies
5	Strong Importance	It is possible to identify a large improvement when comparing these methodologies
7	Very Strong Importance	It is possible to identify a massive improvement when comparing these methodologies
9	Certain Importance	The difference between each methodology is very clear
2,4,6,8	Mediators	These are applied when searching for middle ground

Table 4.1: Fundamental Scale by Saaty [Retrieved from [73]]

Furthermore, four criteria were identified, which means that, for the purpose of the AHP:

$$n = \text{criteria quantity} = 4$$

	Relevance	Scalability	Usability	Reliability
Relevance	1	1/2	1/3	1/6
Scalability	2	1	2	1/3
Usability	3	1/2	1	1/5
Reliability	6	3	5	1
Total	12	5	25/3	17/10

Table 4.2: Identified Criteria Comparison Matrix

	Relevance	Scalability	Usability	Reliability
Relevance	1/12	1/10	1/25	5/51
Scalability	1/6	1/5	6/25	10/51
Usability	1/4	1/10	3/25	2/17
Reliability	1/2	3/5	3/5	10/17

Table 4.3: Identified Criteria Comparison Matrix Normalized

Table 4.4 results from the calculation of the average of each row. This values were defined considering management's perspective and direction.

	Relative Priority
Relevance	0.080
Scalability	0.201
Usability	0.147
Reliability	0.572

Table 4.4: Identified Criteria Priority Vector

According to [20], it is crucial to validate the consistency ratio (CR) value, which requires the presented formula's calculation, given that its value should be less than 0,1, with M being the Criteria Comparison Matrix (table 4.2) and PV being the Priority Vector (table 4.4):

$$\begin{aligned}
 M \times PV &= \lambda_{max} \times PV \Leftrightarrow \\
 \Leftrightarrow \lambda_{max} &= \begin{bmatrix} 0.32483 \\ 0.84567 \\ 0.60190 \\ 2.39 \end{bmatrix} / PV \Leftrightarrow \lambda_{max} = \begin{bmatrix} 0.32483 \\ 0.84567 \\ 0.60190 \\ 2.39 \end{bmatrix} / \begin{bmatrix} 0.080 \\ 0.201 \\ 0.147 \\ 0.572 \end{bmatrix} \Leftrightarrow \\
 \Leftrightarrow \lambda_{max} &= \left(\frac{0.32483}{0.080} + \frac{0.84567}{0.201} + \frac{0.60190}{0.147} + \frac{2.39}{0.572} \right) / 4 \Leftrightarrow \lambda_{max} = 4.1351
 \end{aligned}$$

Then, with this value it becomes possible to calculate the consistency ratio (CR) by calculating the Consistency Index (CI) and using the predefined value of RI, which was retrieved from table ?? in appendix B.1, in accordance to the number of criteria.

$$CR = \frac{CI}{RI} \quad IC = \frac{\lambda_{max} - 4}{4 - 1} = 0,045033$$

$$CR = \frac{0,045033}{0,90} \simeq 0,05004$$

The same comparison principle is then applied to every identified criteria regarding the alternative solutions, which were specified by considering what was clarified by management. Table 4.5 focuses on comparing the solutions regarding the relevance criterion.

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1	1/6	7/12
Implement Microservices	6	1	7/2

Table 4.5: Relevance Comparison Matrix

Table 4.6 evaluates which solution facilitates scalability the most.

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1	1/5	3/5
Implement Microservices	5	1	3

Table 4.6: Scalability Comparison Matrix

Table 4.7 focuses on usability aspects.

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1	1/4	5/8
Implement Microservices	4	1	5/2

Table 4.7: Usability Comparison Matrix

Finally, table 4.8 considers reliability on these systems.

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1	1/2	3/4
Implement Microservices	2	1	3/2

Table 4.8: Reliability Comparison Matrix

These matrices are then normalized in order to calculate priorities, which are available in appendix B.2, to construct the priority matrix, which is then multiplied to the Priority Vector to calculate priority.

$$\begin{bmatrix} 1/7 & 1/6 & 1/5 & 1/3 \\ 6/7 & 5/6 & 4/5 & 2/3 \end{bmatrix} \times \begin{bmatrix} 0.080 \\ 0.201 \\ 0.147 \\ 0.572 \end{bmatrix} = \begin{bmatrix} 0.2645 \\ \mathbf{0.7350} \end{bmatrix}$$

It is possible to conclude that the best alternative is the implementation of microservices, in accordance with the criteria analysis via AHP.

4.2.5 Concept Development

As the only gate between FEI and NPD, the concept definition element results in investment plans and a business proposition that usually address the company's goals and the projects' fitness regarding corporate strategies while pondering inherent risk factors.

Currently, the legacy application is already commercialized by FARO. This migration process will improve the solution by tackling identified issues, specifically project modifiability and maintainability.

4.3 Value Analysis

This section focuses on identifying and describing the potential value offered by this solution. As presented previously in this chapter, value is a subjective term that can vary in accordance with the point of view. However, there are common characteristics regarding the definition of value, specifically: performance, capability, appeal, and cost.

4.3.1 Value for the Customer

The Repeat Part Management application includes three extremely tightly coupled services, which harden system debugging. The consequence of this is human resource allocation to resolve constant breaking changes, which translates into not being able to allocate them into the innovation of the application.

The system's no readability and tight-coupling characteristics reflect an increased latency in delivering value (new functionalities) to the customer in the long term, reflecting on the business's market value.

Clearly defining responsibilities in microservices allows developers to quickly pinpoint the causalities and fix them, allowing management to allocate them to other planned tasks, positively impacting the business' market value.

Microservices will allow developers to follow management's goals for RPM, defined in section 2.3.

4.3.2 Perceived Value

Woodall represents perceived value as a client's comparisons of benefits and trade-offs in a longitudinal perspective by splitting it into four temporal positions ([87]).

These perceived values for the customer positions are related through time and are:

1. Ex Ante predicts how clients perceive the solution and how they ponder benefits before purchase;
2. Transaction value translates into the real-time trade experience expectation;
3. Ex Post reflects the perceived value after acquiring the solution;
4. Disposition reflects the attributed value to the solution after extensive use;

In table 4.9 these temporal positions are presented regarding this thesis' solution proposition.

	Benefits	Trade-offs
Ex Ante	Knowledge Aggregation; Decoupling of Services; Lifting of Restrictions regarding innovation;	Complexity regarding the implementation of microservices encompasses a significant amount of time and effort regarding research;
Transaction	Code readability; Services clear responsibility definition; Documentation regarding this new solution;	Costs regarding training developers; Costs regarding maintenance; Company's resource allocation for implementation;
Ex Post	Increased readability of the system; Increase in productivity; Greater scalability of the application;	Resource allocation costs;
Disposition	Greater ability to pinpoint issue causalities; More frequent value delivery; Project readability;	Maintenance and Development Costs; Costs regarding resources such as cloud-environment

Table 4.9: Benefits and Trade-Offs regarding the longitudinal perspective of value for the customer

4.3.3 Value Proposition

Communicating the value proposition allows businesses to efficiently articulate the benefits of the proposed solution per the described client's problem by contextualizing it with the available alternatives and the target client.

This thesis focuses on implementing a microservices-based application on a brownfield domain based on Repeat Part Management (RPM), which has accumulated technical debt, according to the developers' team. The technical debt created several issues in the development cycle of the legacy application. It hindered the new functionalities innovation process given the latency regarding debugging and issue fixing. Furthermore, the amount of breaking changes increased, which relates to the gradual tightening of application coupling.

Repeat Part Management's development became increasingly limited, which restricted progress and specific goals such as the eventual implementation of a cloud-based environment (AWS).

Implementing this architecture is complex and must entail knowledge aggregation on migration design patterns and good practices. This solution will allow developers and management to implement planned functionalities by facilitating project readability and loosening service coupling; additionally, given the characteristics of this architecture, this solution allows developers to be aware of errors and quickly pinpoint failure causalities. Maintaining the current solution as an alternative to this implementation would only encourage further technical debt accumulation.

The goal is to tackle the challenges of developers and management brought on by the legacy application and the inherent accumulation of technical debt.

4.4 Business Model Canvas

Alexander Osterwalder proposed the Business Model Canvas to contribute to business model conceptualization ([63]). It comprises nine different elements, which display the value analysis graphically by describing how means are combined to deliver value to identified customer segments ([63]).

- **Key Partners:** Identify key partners by pondering their motivations and intentions in optimizing the solution—additionally, critical suppliers from whom resources are acquired.
- **Key Activities:** Identify critical actions to achieve the proposed value addition.
- **Key Resources:** Determine the required human or physical resources to guarantee project development and, inherently, the proposed value.
- **Cost Structure:** Considering development costs related to physical and human resources to encourage contemplation on cash flow.
- **Revenue Streams:** It is essential to determine the leading direct/indirect investors of this proposed value regarding the cost structure.
- **Value Proposition:** Define this concept's immediate value proposition by determining the customers' problems (which this concept solves).
- **Customer Relationships:** Define the customers' association expectation to the value proposers - and how they will be involved.
- **Customer Segments:** Determine who will be most affected by these modifications and, inherently, interested customers.
- **Channels:** Define possible communication approaches with the client regarding how they will be involved in development.

Figure 4.4 represents the application of this model on the thesis' solution.

Key Partners FARO. Information Sources. Microsoft.	Key Activities Research. Decomposition. Microservices Design and Development. DevOps Configuration.	Value Proposition Allow Cloud database implementation. Improved development time. Responsibility definition in microservices. Improvements in the application's scalability. Allow Multi-User Access to Documents.	Customer Relationships Evaluation of system complaints. Requirement gathering.	Customer Segments Developer teams at FARO. Product Owner.
	Key Resources Developer Team. Sphere Cloud's Services. Public Documentation. Legacy System.		Channels JIRA. Confluence. Team Meetings. Project Meetings. Microsoft Teams.	
Cost Structure System infrastructure costs (Cloud storage). Infrastructure Costs (Equipment and Software). Human Resources.			Revenue Streams FARO's internal investment;	

Figure 4.4: Business Model Canvas

Chapter 5

Design

As discussed previously, designing the architecture of an application is generally a complex concern, however, in microservices, it becomes specially concerning given the inherent decomposition process. In this section architectural drivers, as well as their specific interest in the project, the functional requirements are presented and described, as well as the proposed design to be implemented and its inherent trade-offs.

5.1 Design

This section consists on presenting alternative designs to the defined requirements and direct referencing related to used design patterns by applying researched techniques. Furthermore, the ubiquitous language will be provided as to make domain concepts clear.

By defining the architectural drivers for this solution it is easier to implement them and relate them to design decisions, which will be possible by using their identification (ID).

Ubiquitous Language

As explained previously in section 3.2, particularly in the Domain-Driven Design subsection, the business model, and its inherent concepts, is directly related to the system, which encourages teams to share terms and ease communication. In table 5.1 these relevant terms are presented.

Concept	Description
Program	A set of repeated tasks inherent to a quality inspection.
Document	General information relevant to Repeat Part Management.
Statistics	Abstract Object related to inspection statistics (e.g. low tolerance).
Feature	A feature represents a detail of a whole part (e.g. the face of a cube).
Session	A session represents a single program run.
Session Readings	A session may encompass readings of given features (e.g. the length of the cube's face).
Report	A review document of one or several sessions and their readings.
Spc	While still standing as Statistical Process Control, Spc is a complex object that entails several crucial values for application in quality assurance.

Warning Rule	These rules delineate what are the adequate values for given spc values.
--------------	--

Table 5.1: Ubiquitous Language Definition

5.1.1 Attribute Driven Design (ADD)

Attribute Driven Design (ADD) is a methodology for designing and documenting a software architecture concerning quality attribute requirements. This approach is meant to be applied iteratively. However, given that this project focuses on decomposition, one cycle is sufficient [85].

ADD introduces two crucial concepts: inputs and outputs. Inputs are functional requirements, design constraints, and quality attributes (usually identified by stakeholders, defined in section 2.3). Outputs are system roles, responsibilities, properties, and software relationships. Table 5.2 identifies these input and output concepts and their definitions.

Concept	Definition
Functional Requirements	Identify system functionalities and specify needs.
Design Constraints	Stakeholders' decisions that have an immediate impact on the final architectural design.
Quality Attributes	Requirements that indicate patterns exhibited throughout the system.
Role and Responsibilities	Aggregation of provided functionalities, data, or information.
Properties	Software elements might have relevant additional information such as name, type, protocols, or others.
Software Relationship	Specification of element association.

Table 5.2: ADD Concepts [85]

Throughout this section these concepts will be utilized to describe system context, design, and decisions - by specifying project concerns, quality attributes, and constraints. Given that this application will be developed without using code from the legacy application it is a part of greenfield development, as stated previously, this means that all defined input and output are assigned to the whole application [85].

5.1.2 Architectural Drivers

This section identifies requirements and their prioritization and their importance regarding two factors, specifically stakeholders and project impact by using High (H), Medium (M), or Low (L) rankings. As stated throughout the document, the goal for this approach is to design a microservices-based application related to an already existing system, while disregarding developed functionalities and approaches by focusing only on domain analysis and needed functionalities.

The tables within this section will encompass requirement identification, description, and priority. To clarify, priority will be presented as two letters (e.g. (H,L) - which would mean that the requirement would be of high relevancy to stakeholders and low impact to the project).

Concern Definition

Design decisions that must be made, but are not stated explicitly in goals or requirements are defined in table 5.3.

ID	Concern	Priority
CRN01	Project will only be developed by one person.	(L,M)
CRN02	Legacy project code analysis and deep understanding of legacy domain.	(L,H)
CRN03	Frontend will not be implemented.	(L,M)

Table 5.3: Project Concerns

Constraint Definition

Project limitations or restrictions that concern technical or organizational aspects, which usually limit alternatives are defined in table 5.4.

ID	Constraint	Priority
CON01	C# language and .Net framework are mandatory (versions are optional).	(H,H)
CON02	Adopt open-source technologies.	(H,M)
CON03	The application must follow principles and good practices related to the microservices architecture.	(H,H)
CON04	The system must communicate through REST API.	(L,H)
CON05	No access to CAM2's source code.	(L,H)
CON06	Database must be Relational and be developed with MySQL.	(H,M)
CON07	The application must encompass all legacy system's functionalities.	(H,H)

Table 5.4: Project Constraints

Quality Attributes Definition

Quality attributes are valuable implementable characteristics for users and developers. The table provides the required foundation for achieving quality (specifying that it does not encompass every good practice, but only those specified by stakeholders). Table 5.5 encompasses an Attribute column, which represents the actual design concept to which the description refers to.

ID	Attribute	Description	Priority
QA01	Readability	Developers have recorded significant difficulty in understanding the used terms for concepts related to a lack of an ubiquitous language.	(M,L)
QA02	Maintainability	Developers from both CAM2 and RPM teams have recorded difficulty regarding legacy project navigation due to technical debt related to previous architectural decisions.	(H,L)
QA03	Modifiability	Stakeholders are trying to expand RPM's functionality, however adding new functionalities to the system is becoming increasingly difficult.	(H,L)

QA04	Modularity	The new system must be in accordance to the single responsibility principle.	(M,H)
------	------------	--	-------

Table 5.5: Project Quality Attributes

5.1.3 Functional Requirements

Figure 5.1 presents a total of 7 use cases, which are directly correlated to the legacy system, considering CON07. This means that these functionalities represent available capabilities in RPM's legacy system. This is a further description of the presented use cases, which must be designed and considered within the new system:

- **UC1 Access published documents:** As a client, I want to be able to see which are the documents, which I have published through CAM2.
- **UC2 Check for available programs:** As a client, I want to be able to check what are the defined programs for the published documents.
- **UC3 Run Programs:** As a client, I want to be able to run programs defined for documents, in order to guarantee quality in the document's part.
- **UC3.1 Visualize running program:** As a client, I want to visualize the running program, while I manually execute tasks, in order to keep track on the program's process.
- **UC4 Check for program's results statistics:** As a client, I want to be able to register the results of the programs I ran and their history, in order to guarantee the quality of my products throughout time.
- **UC5 Create reports regarding results:** As a client, I want to be able to create reports related to the result's statistics, in order to keep records of products inspections.
- **UC5.1 Choose a report template from existing options:** As a client, I want to choose a report template from a list, in order to guarantee efficiency in documentation.

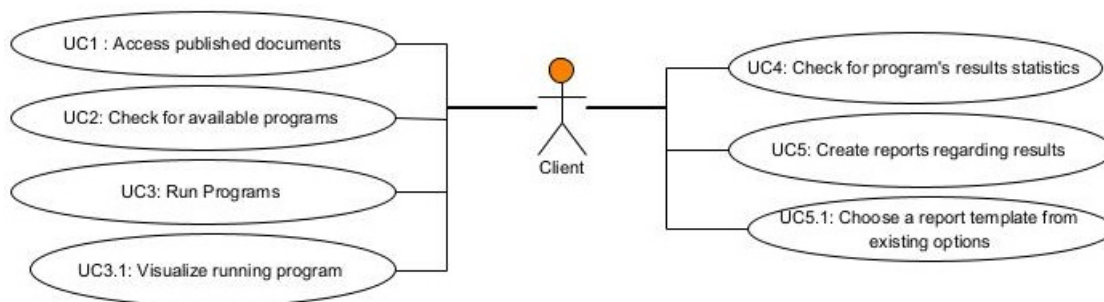


Figure 5.1: Functional Requirements

Due to time constraints, implementing all the Use Cases was not possible, the chosen use cases to implement were: UC1, UC2, UC5, and UC5.1, given their correlation and characteristics, chosen after the design phase described further in section 5.2.

5.2 System Decomposition Alternatives

This section will present the chosen decomposition strategy and subsequent alternatives, related to the decomposition systematic mapping study done on chapter 3 (section 3.4).

Considering the characteristics of this architecture, specified in section 3.2, applying decomposition based on technical aspects may lead to highly coupled services, thus hindering the development process. Furthermore, it is essential to be aware of Nano and Mega Microservices, described by [82], which may also affect service coupling. This explains why some approaches were discarded when highly automatic in scope, or when these approaches are unavailable (private software).

5.2.1 Decomposition by Business Capability Alternative

To better understand RPM's legacy domain in figure 5.2 are the main subdomains that it provides. This is a general view of the functionality pool available in the legacy system which must be decomposed as to be available in the microservices-based architecture.

The client is able run defined *programs* (U1), view those inspections *statistics* throughout time (U2), generate *reports* concerning those statistics and general information (U3), and access published *documents* (U4).

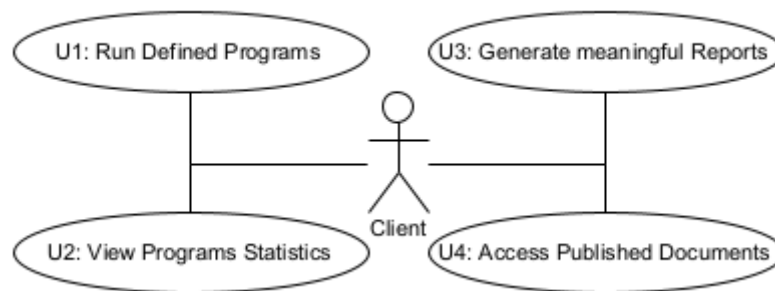


Figure 5.2: Legacy Use Cases

By following the Decomposition by Business Capability pattern, identified in section 3.3.2, the decomposition of this application would simply be put as shown in figure 5.3, where each main capability directly translates into a service with defined responsibilities. However, it seems that RPM's context is not broad enough to guarantee this approach's success, given that it is used to decompose more business abstract systems. Furthermore, applying this strategy can hinder system in a long term perspective given that it does not consider complex component relationships and might result in a nano or macro service.

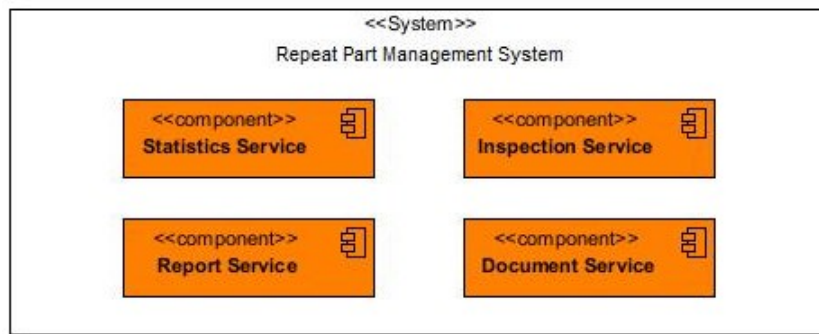


Figure 5.3: Decomposition by Business Capability - Logical View Alternative

For further decomposition strategies to be used as alternatives it is necessary to define the domain model, and its inherent bounded contexts (defined further in section 3.2), which may define clear limits within the domain that can act independently with their own entities, value objects, and aggregates.

5.3 Applied Decomposition

In this section, the applied decomposition will be specified and detailed. The final domain model, its bounded contexts, and aggregates, which were formulated with the Decomposition by Subdomains approach presented previously and agreed upon managements' referral.

5.3.1 Responsibility Separation

Table 5.6 directly describes the defined services' responsibilities. As there are two relevant front end use cases, it is relevant to underline the fact that this is not the thesis goal and it will not be implemented, however, responsibilities for their implementation are defined.

Service	Responsibility
Document Service	This service will be fully responsible in keeping program information updated as well as general information about sessions, so It is capable to allow program running visualization.
Report Service	This service is responsible for creating, managing, and storing generated reports.
Session Service	Session Service is responsible for managing program running data, particularly relevant to SPC, furthermore, It will be responsible for program run's statistics visualization.

Table 5.6: Services and Responsibilities

Context Map

In the microservices architecture, it is very important that relationships between bounded-contexts are well-defined, as well as their integrations - this is done by presenting a context map. Figure 5.4 represents the identified bounded contexts and their relationships and

dependencies. It represents the Session Context, Document Context, and Report Context, which have specific relationships between them.

The Report Context is particularly dependent on data from Sessions and conforms to its data model, this means that this Report Context (Downstream) accepts what the Session (Upstream), which means that the upstream, although having its own domain model, conforms to what the upstream sends. This happens with the document context also. Upstream-Downstream relationships define that upstream contexts are independent from the downstream, while the downstream is not but does not expose its domain to upstream.

As the Session context and the Document context also share a relationship, the Session is more independent in a domain model perspective and it need an Anti-Corruption Layer (ACL) to guarantee that data is consistent and updated with other contexts. The Document context encompasses the general information about defined programs, which are use in Session, however, Session does not conform its domain making this ACL necessary.

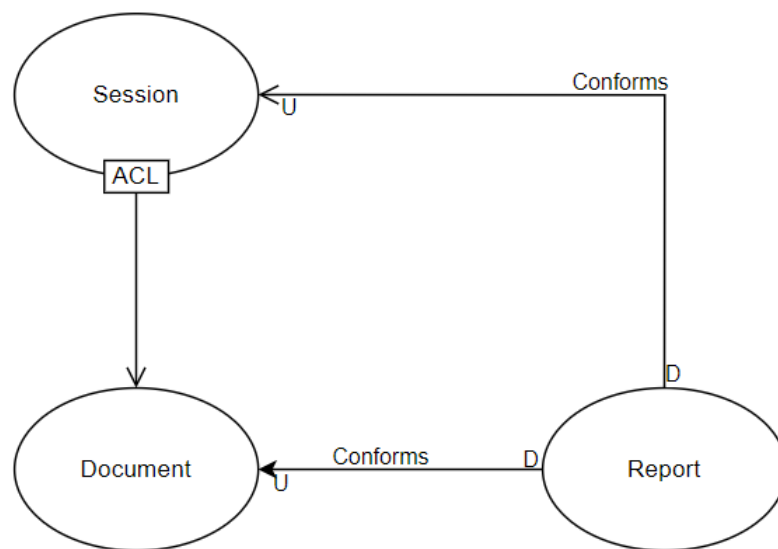


Figure 5.4: Context Map

5.3.2 Document Service Domain Model

Figure 5.5 presents the document' bounded context, which splits into two aggregates: the *Program* aggregate and *Document* aggregate. Each aggregate has its own aggregate root to ensure consistency by being the only entry point for the aggregate's operations - these roots are usually identified by the «Root» stereotype, generally at the top of the object.

As this context is responsible for keeping program information updated regarding programs and past sessions it entails objects related to both of these. Specifically, the document aggregate encompasses the *Document* object, which is an Entity, and the *CadFile* and *FcdFile*, which have all the information related to the product or part depicted in the *Document*.

The *Program* aggregate encompasses general information about document activity, such as *Program(s)* and their activities (*Activity*), which can be related to some specific feature (defined in CAM2) and has a related measuring *Device*. Given that a *Program* can be ran several times, each time is represented by a *Session*. Although the Session Service is responsible for the Session and Statistical Information, having the information about the

success is relevant for the business. For that reason, the *Session* entity is related to *Session Statistics*, which relate to each feature's *Statistic(s)*.

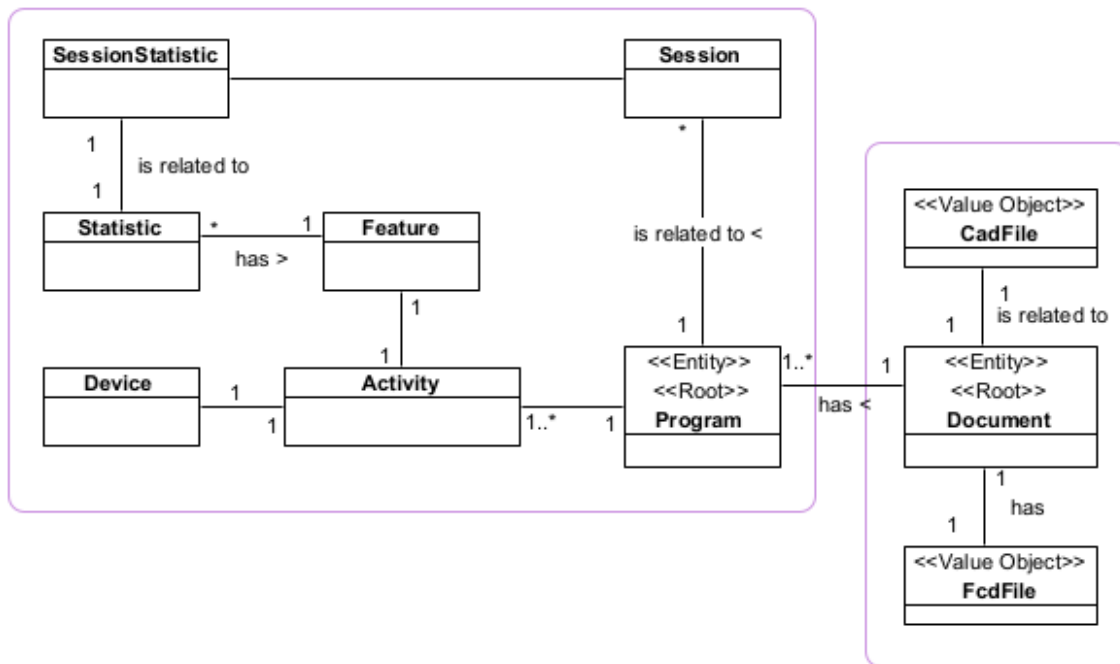


Figure 5.5: Document Microservice Domain Model

5.3.3 Report Service Domain Model

Figure 5.7 represents the domain model for the Report context, which is divided into two Aggregates related to: *Report* aggregate and *Feature* aggregate. The Feature aggregate relates to details of the whole part (described in the ubiquitous language table in section 5.1), while the *Report* aggregate focuses on organizing data according to Report's data necessities.

The report aggregate has an aggregate root *Report*, which represents the whole report object, which must have an associated template (chosen in CAM2, but fully kept in the RPM system). *Session Information*, which is related to the first part of the Report (general information section) is an Entity that keeps information about the used device (which can be changed from the one kept in Document Service), *Inspection* general information, the *Organization and File Names*, and contact information.

The Feature aggregate keeps every information related to the next section of the report (statistical section). As every statistic is presented concerning each feature, it made structural sense to make *Feature* the root for it and associate a *Statistics* object that keeps the represented domain primitives.

Figure 5.6 shows a real-life report scenario, specifically the feature section, in which:

- *Features*: Circle 1, Circle 3, Circle 5, Circle 7, ...
- *Statistics*: Represented by the aggregation of all the values, such as *Actual*, *Nominal*, *dev*, ...

The chosen names for these objects are related to actual used variables, which makes it easier to understand their placement and usefulness within the domain.

 World	actual	nominal	dev	tolerance	pass/fail	oot
 Circle 1	actual	nominal	dev	tolerance	pass/fail	oot
Center.x	-70.020mm	-70.000mm	-0.020mm	±0.050mm	Pass	
Center.y	-0.010mm	0.000mm	-0.010mm	±0.050mm	Pass	
Center.z	0.000mm	0.000mm	0.000mm	±0.050mm	Pass	
Diameter	20.045mm	20.001mm	0.043mm	±0.050mm	Pass	
Circularity	0.001mm		0.001mm	0.050mm	Pass	
 Circle 3	actual	nominal	dev	tolerance	pass/fail	oot
Center.x	-0.004mm	0.000mm	-0.004mm	±0.050mm	Pass	
Center.y	70.013mm	70.000mm	0.013mm	±0.050mm	Pass	
Center.z	0.000mm	0.000mm	0.000mm	±0.050mm	Pass	
Diameter	20.026mm	20.001mm	0.024mm	±0.050mm	Pass	
Circularity	0.001mm		0.001mm	0.050mm	Pass	
 Circle 5	actual	nominal	dev	tolerance	pass/fail	oot
Center.x	70.021mm	70.000mm	0.021mm	±0.050mm	Pass	
Center.y	-0.001mm	0.000mm	-0.001mm	±0.050mm	Pass	
Center.z	0.000mm	0.000mm	0.000mm	±0.050mm	Pass	
Diameter	20.031mm	20.001mm	0.030mm	±0.050mm	Pass	
Circularity	0.005mm		0.005mm	0.050mm	Pass	
 Circle 7	actual	nominal	dev	tolerance	pass/fail	oot
Center.x	0.017mm	0.000mm	0.017mm	±0.050mm	Pass	
Center.y	-70.005mm	-70.000mm	-0.005mm	±0.050mm	Pass	
Center.z	0.000mm	0.000mm	0.000mm	±0.050mm	Pass	
Diameter	20.043mm	20.001mm	0.042mm	±0.050mm	Pass	
Circularity	0.004mm		0.004mm	0.050mm	Pass	

Figure 5.6: Report Content Example

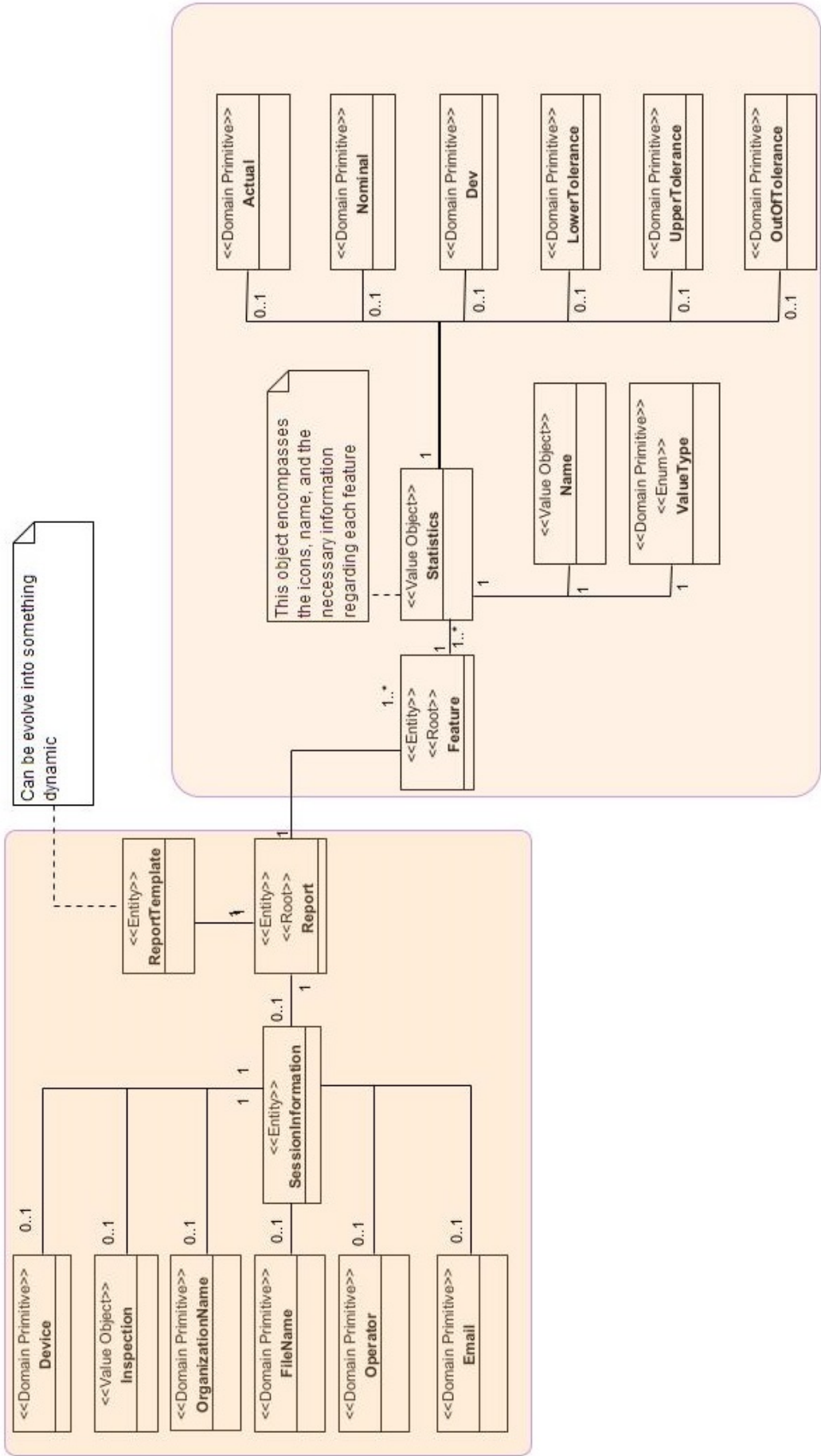


Figure 5.7: Report Microservice Domain Model

5.3.4 Session Service Domain Model

This domain focuses on the session context, which responsibility is to manage program running, its data, and necessary SPC calculations for analytics facilitation. This service is also split into two aggregates which have mostly split responsibilities: *Session Readings* and *Spc Data*.

The *Session Readings* is responsible for keeping program run data, while *SpcData* has the responsibility for calculations, which explains why most of this domain entails domain primitive values, mainly used for actual functionality calculations and not too much business logic. *Session Readings* encompasses *Measured Data*, related to several features (*SessionFeatureReadings*), which have several statistical values, and must have a tolerance state relevant for session information.

Currently, there is only a unique set of rules regarding *WarningRuleSet*, which is implemented as an abstract to facilitate future implementation of other rule sets.

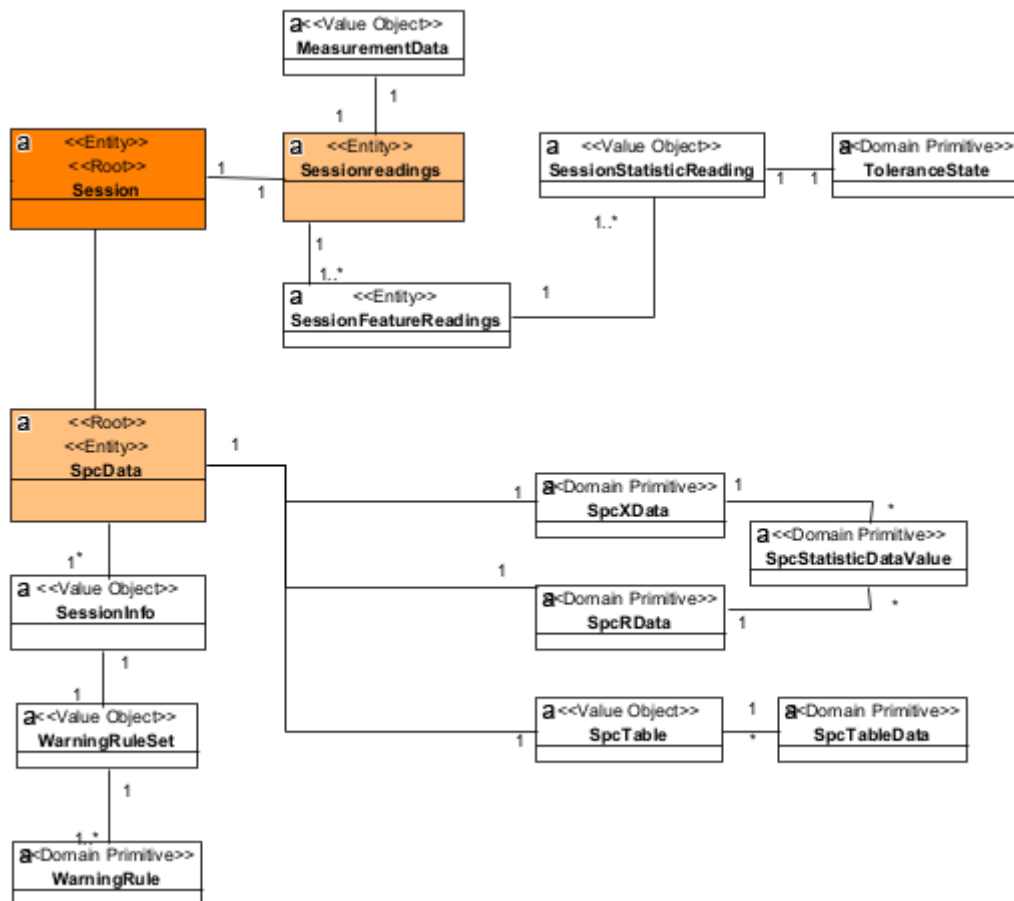


Figure 5.8: Session Microservice Domain Model

Chapter 6

Implementation

After identifying the design decisions in the previous chapter, this chapter's goal is to present the proposed solution's development approaches and the achievement of the requirements approach. As said previously, the front-end is not this thesis goal or responsibility, which means it isn't implemented.

This section presents architecture software descriptions based on multiple and concurrent views, also known as the 4+1 model. These are the model's views [50]:

- **Logical View:** intends to contextualize the system's functionalities and identify communication protocols and software design elements (e.g. component diagram).
- **Process View:** defines responsibility and logical element's collaboration.
- **Development View:** defines observable dependencies and coupling between software's packages in a developer's perspective.
- **Physical View:** also known as deployment view, which defines the physical implantation of nodes.
- **Scenarios View:** defines use cases workflow by describing sequences of interactions between software elements.

Figure 6.1 represents these architectural views and their relations:

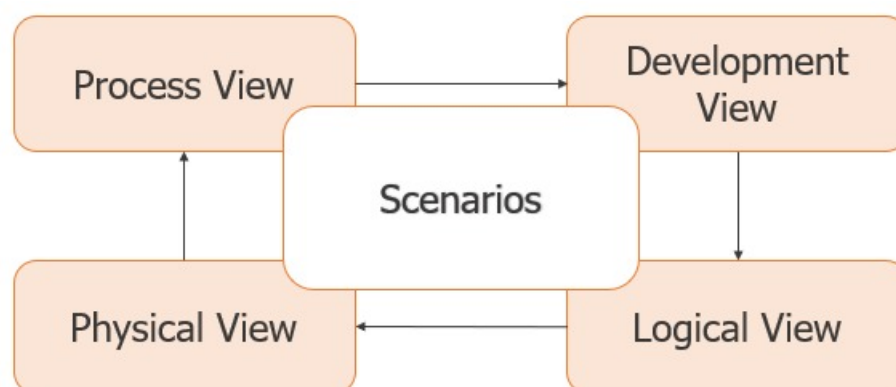


Figure 6.1: 4+1 model [Adapted from [50]]

6.1 Logical View

A component perspective is shown in Figure 6.3, which specifies the decomposed system. As said previously, only CRUD functionalities were developed regarding the implemented microservices, which means that implementation did not encompass external application relationships. Any further implementation decisions about components presented in this system are presented in section 6.

As seen in Figure 6.3, two of the identified decomposed microservices were implemented, chosen for their domain's low coupling, given that the implementation of a message broker was not this thesis' focus. Furthermore, the database per service pattern was used in accordance with microservices good practices.

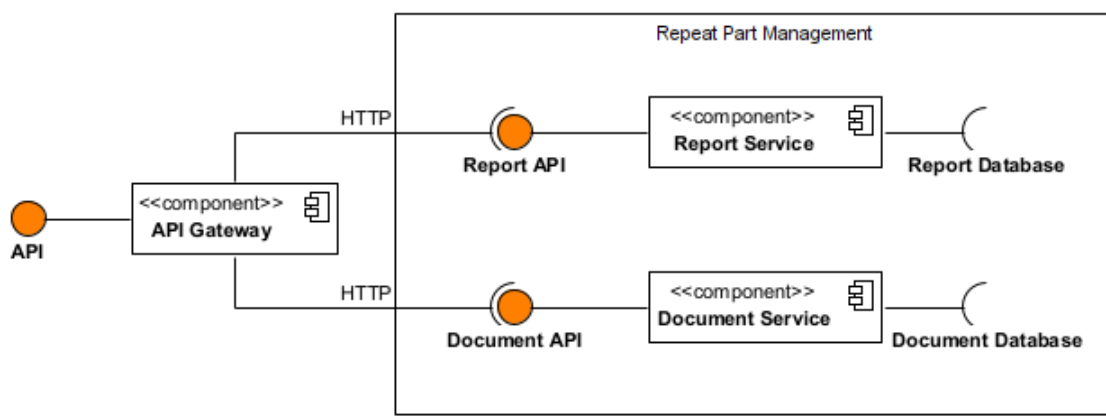


Figure 6.2: Implemented Logical View Diagram

These microservices share common data, such as Document identification (*DocumentID*), which means that when a report is generated (and the identification must be checked) the Document Service must confirm that the *DocumentID* is valid. This can be implemented through a HTTP request to the Document API, however, when other microservices are developed and added into the system this implementation can become quite inefficient (regarding performance), which might justify the implementation of a Message Broker.

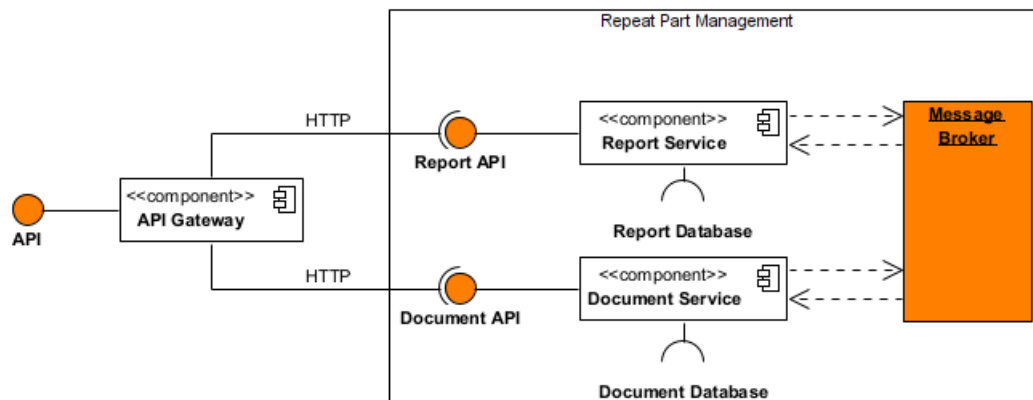


Figure 6.3: Logical View Alternative (with Message Broker)

Message Brokers enable systems to communicate by translating messages into a protocol, which is specially useful when these services are written in different languages, which is not necessarily relevant in these context currently, given the CON01 and CON06 constraints defined in table 5.4. Message Brokers are also able to validate and route these messages. A relevant pattern is the Publisher/Subscriber, which distributes messages according to which services are subscribed to it.

6.2 Process View

Clean Architecture, formerly known as onion architecture is a domain centered approach to dependency organization. A common software practice that deals with application complexity is introducing layers [78].

All project dependencies should be directed toward the Core project, which serves as the design's focal point in the Clean Architecture. Interfaces, Aggregates, Entities, Value Objects, Domain Exceptions, and Domain Validators should all be part of the Core project.

The majority of the dependencies on outside resources for your application ought to be implemented in classes provided in the Infrastructure project. These classes ought to implement the Core interfaces. Multiple Infrastructure projects (such as Infrastructure.Data) may make sense if you have a very large project with a lot of dependencies, but for most projects, one project with multiple components is sufficient. The following should be a part of the infrastructure project: Repositories, Database Context, External Services,...

The Web application is the solution's entry point and deals with API endpoints, controllers, and, if necessary the frontend.

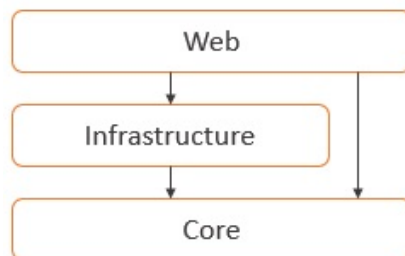


Figure 6.4: CleanArchitecture Overview [Adapted from [78]]

Figure 6.5 specifies the used layers in the architecture of the report and document services, similar to the onion architecture, which is commonly used in C# projects, that should be concerned with dependency flow [78]. This diagram encompasses the report and document solution's software elements. The displayed components: *Web*, *Core*, and *Infrastructure* were developed considering the clean architecture rules explained previously. However, *SharedKernel*, a DDD concept, holds common types, which are typically referenced by *Core*. In this case, the *SharedKernel* encompasses the Response and Request layer and the Base classes, specifically Base Entity and Base Value Object.

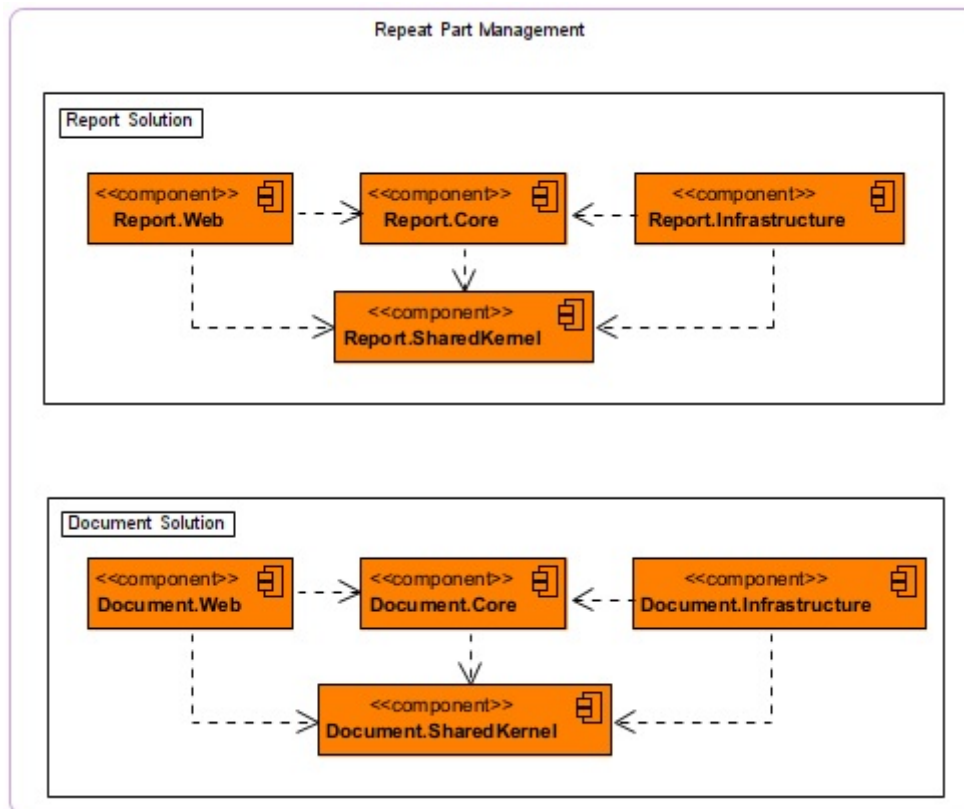


Figure 6.5: Process View Diagram

6.3 Development View

In the process view, it was possible to understand the general components of each solution (when referring to "solution" it relates to the project's actual structure, also known as the actual .sln file in C# solution).

Figures 6.6 and 6.7 represent the structure of the application in a development perspective, as they are the same, by presenting specific components of these projects.

The web project is responsible for controllers and API endpoints and represents the entrance point for this project. The infrastructure project relates to data and data management by encompassing services, repositories, migrations, and the database's context. The core is responsible for all business logic, and is split into aggregates, and domain validators (guarantee object validity within the system). Furthermore, the SharedKernel has the DTOs alternatives which are the Requests and Responses, and the general Base Classes (Entity and ValueObject).

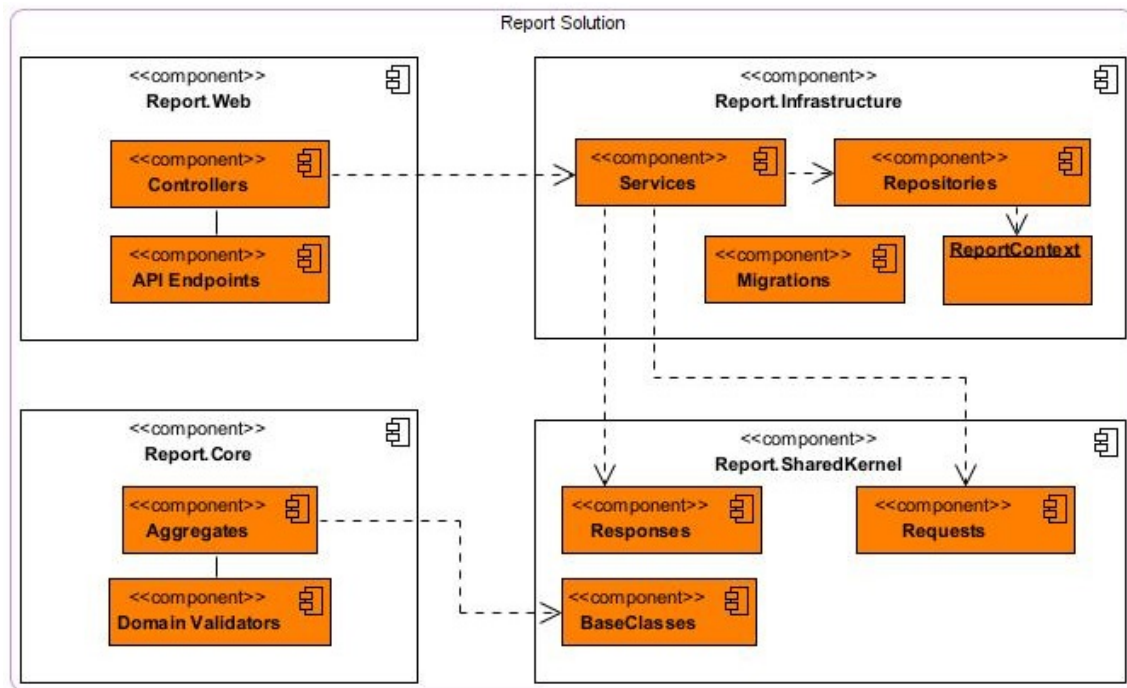


Figure 6.6: Report Solution Development View Diagram

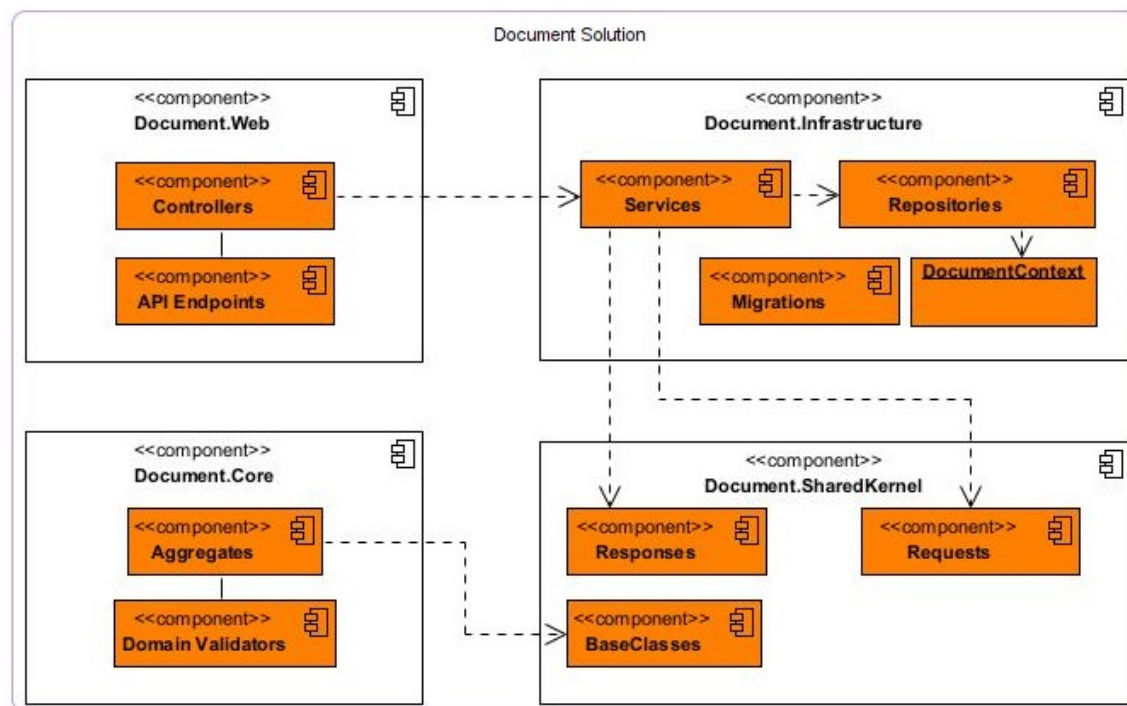


Figure 6.7: Document Solution Development View Diagram

As the component dependencies are the same in both solutions, the sequence diagram example focuses only on creating a report and is partially presented in figure 6.8. From an

external client (machine or actual client) a request is sent (HTTP POST request) to Repeat-Part Management's API gateway, which main goal is to forward it into the responsible service to be processed.

Services' point of entry are the defined controllers (in this specific UC the *ReportController*), which then coordinate these requests. The request is in JSON format, which is changed to a Request form before proceeding into the service. This is due to the use of the Request/Response objects, commonly used in C#, which can facilitate the possible integration of a message broker and replaces Data Transfer Objects (DTOs). While having an identical purpose, Request and Response classes usually improve system readability and are specific to functionalities, making it a bad practice to reuse the same data transfer object for every request/response, which also increases workload and decreases reusability - implemented due to the microservice's size and responsibility-spectrum size.

This request is then sent to the responsible service through its interface (*IReportService* in this UC's specific instance). This service is then responsible for creating the necessary object models (*Report*), which must then be saved in the database. This object is then sent to the responsible repository's interface - repository pattern is used for data and domain model mediation.

The report creation is the chosen example of the system's flow given that it was structured to be consistent and it represents the general use case. The only non-represented components are the interfaces (hidden for diagram simplification) - services and repositories are always accompanied by a respective interface, this means that controllers do not access service but their interfaces; the same happens between services and repositories, given that services access the repositories respective interfaces. Furthermore, controllers, services, and repositories are created for each aggregate root, which means that, in the Report Microservice there are two of each: *ReportController*, *ReportService*, *ReportRepository*, *FeatureController*, *FeatureService*, and *FeatureRepository* (and the service's and repositories respective interfaces).

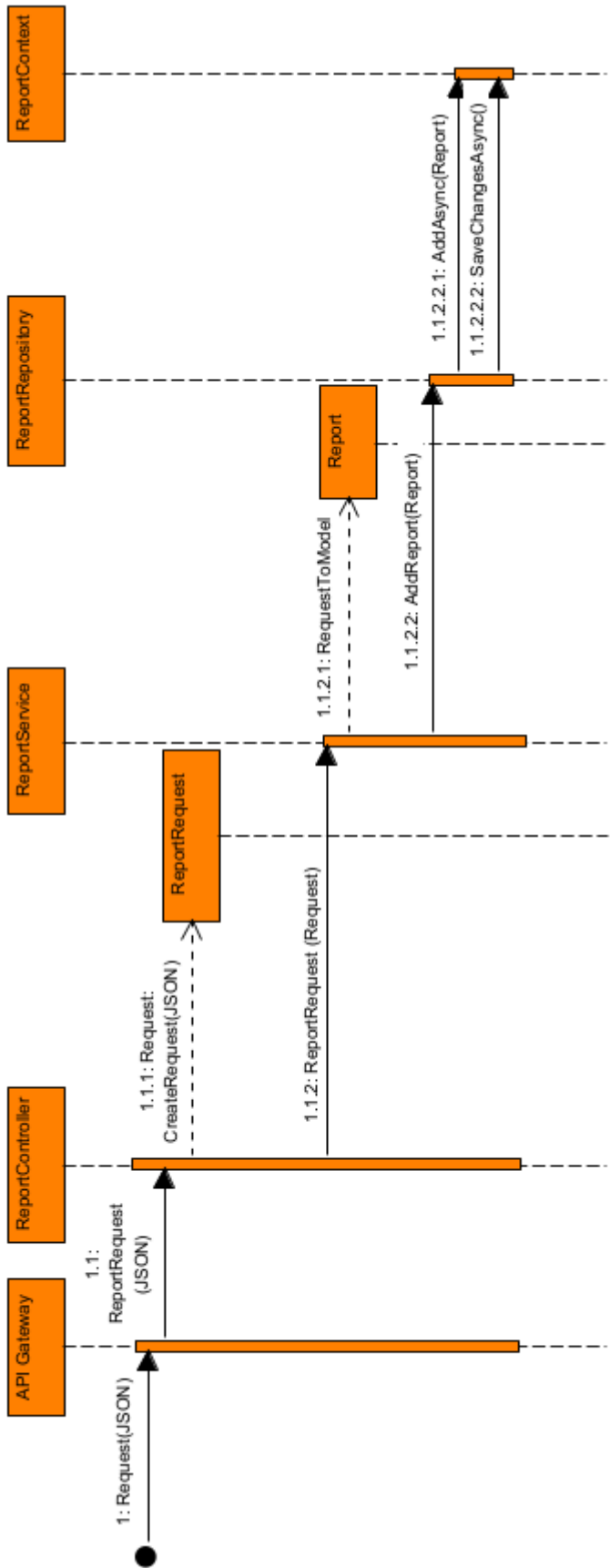


Figure 6.8: Create Report Sequence Diagram

6.4 Physical View

Figure 6.9 represents the physical view of the implemented system, which describes how the system is containerized. It is important to reference that the containers platform used was Docker for it's already extensive use in FARO.

Given the modularity scope of the microservices architecture, the evident choice of containerization is individual implementation. This approach supports the concept of low coupling by allowing for future modifications without affecting other containers.

However, this approach implies some maintenance drawbacks related with implementation and security costs, given that these containers may be increasingly susceptible when deployed into web environments - such as the legacy system.

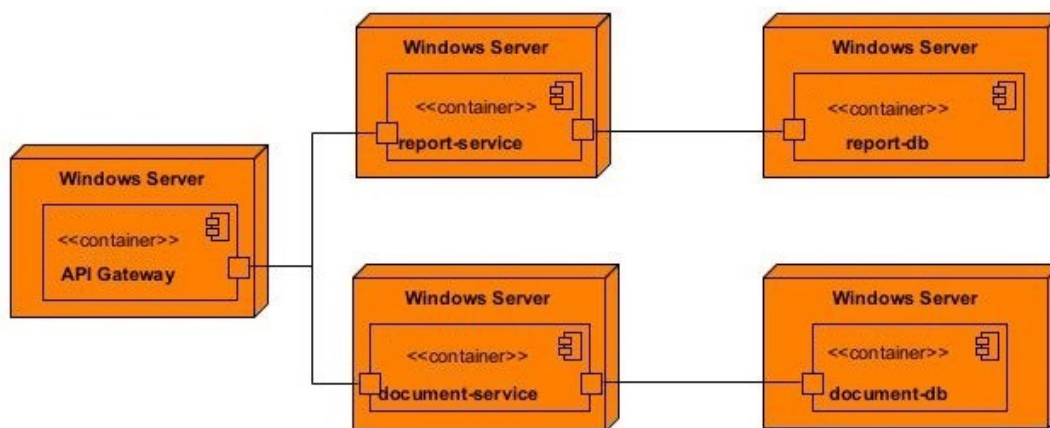


Figure 6.9: Implemented Deployment View

6.5 Patterns Implementation

In this section, implemented patterns, previously identified in chapter 3 and within the implementation views, are discussed, as well as their inherent trade-offs.

6.5.1 Domain-Driven Design

Domain-Driven Design was extensively applied by using Entity Framework Core (EF Core) in development. This framework works similarly to an Object Relational Mapper by enabling developers to work with databases directly with the defined domain model and facilitating database management [58].

Figure 6.10 is an example of how this framework can be directly used in models with annotations, represented on top of each variable within the brackets. EF Core allows developers to define domain identification (Key), foreign keys, mandatory variables, and can actually allow developers to define complex relational relationships (many-to-many), and object reference.

```

1 namespace Faro.Report
2 {
3     public class Report
4     {
5         [Key]
6         public int Id { get; set; }
7
8         [Required]
9         [ForeignKey("FK_Report_Document")]
10        public int DocumentId { get; set; }
11
12        [Required]
13        public ReportTemplate Template { get; set; }
14
15        [Required]
16        public string ReportFileName { get; set; }
17
18        [Required]
19        public SessionInformation SessionInfo { get; set;}
20
21        [Required]
22        public int SessionId { get; set; }
23
24        [Required]
25        public List<Feature> Features { get; set; }
26    }
27 }

```

Figure 6.10: Entity Framework in Report (Business Logic)

Furthermore, EF Core allows for database access through a defined model, which is defined in each microservice's context. This context allows querying and saving data [58]. Querying and saving data is done in the repositories which instance the database context.

Figure 6.11 represents how the collection of entities that may be queried are set with `DbSet<>`.

```

1 public DbSet<Report> Reports { get; set; }
2 public DbSet<ReportTemplate> ReportTemplates { get; set; }
3 public DbSet<ReportTemplateImage> ReportTemplateImages { get; set; }
4 public DbSet<SessionInformation> SessionInformations { get; set; }
5 public DbSet<Feature> Features { get; set; }
6 public DbSet<Statistics> Statistics { get; set; }

```

Figure 6.11: Context Database Set

Figure 6.12 shows how the model was created in the context object of the specific microservice (*DocumentContext* and *ReportContext*). The defined model must already be normalized (considering relational databases).

```

1  protected override void OnModelCreating(ModelBuilder builder)
2  {
3      base.OnModelCreating(builder);
4      builder.Entity<Report>(entity =>
5      {
6          entity.ToTable("Report");
7          entity.HasKey(entity => entity.Id);
8          entity.HasMany(entity => entity.Features).WithOne();
9      });
10
11     builder.Entity<Feature>(entity =>
12     {
13         entity.ToTable("Feature");
14         entity.HasKey(entity => entity.Id);
15         entity.HasMany(entity => entity.Statistics).WithOne();
16     });
17
18     builder.Entity<SessionInformation>(entity =>
19     {
20         entity.ToTable("SessionInformation");
21         entity.HasKey(entity => entity.Id);
22     });
23
24     builder.Entity<ReportTemplate>(entity =>
25     {
26         entity.ToTable("ReportTemplate");
27         entity.HasKey(entity => entity.Id);
28         entity.HasMany(entity => entity.Images);
29     });
30
31     builder.Entity<ReportTemplateImage>(entity =>
32     {
33         entity.ToTable("ReportTemplateImages");
34         entity.HasKey(entity => entity.Id);
35     });
36
37     builder.Entity<Statistics>(entity =>
38     {
39         entity.ToTable("Statistics");
40         entity.HasKey(entity => entity.Id);
41     });
42 }
43 }

```

Figure 6.12: Creating Report Model (Database Context)

6.5.2 Database Per Service

As microservices are known for scope modularity, having a database per service will encourage loose coupling by independent deployment and scalability. This means that in an architecture which scope entails the Single Responsibility Principle, it is crucial that data storage is in accordance to concern separation. By combining modular containerization, databases remain somewhat agnostic to services, and services can be easily scalable (while using the same database), something that is explained further in section 6.5.5. Also allowing databases to be implemented differently (management system: relational, non-relational, ...).

Having different databases per service allows developers to encourage loose coupling and facilitates scalability, however, fault tolerance becomes a bigger concern, for instance, if a given requests requires two microservices, and the connection to one of the microservice's

database goes offline, then there should be a fault tolerant approach, such as circuit breaker, presented in section 3.3, which was not implemented.

6.5.3 OpenAPI

Microservices require documentation, however this is a laborious task, given that software updates are reoccurring and documentation maintenance is mostly neglected. For that reason, implementing automated documentation is a necessary to allow teams to understand a system capabilities. OpenAPI allows RESTful APIs to define and display itself with minimal amount of logic [76].

OpenAPI is initiated in the Web project of each solution, where the project dependencies and main method are defined. Figure 6.13 shows the straightforward implementation.

```
1 builder.Services.AddSwaggerGen();  
2 app.UseSwagger();  
3 app.UseSwaggerUI();
```

Figure 6.13: OpenAPI Config Code

OpenAPI mainly works with annotations, that are implemented in the entry-points to the solution (the controller defined requests). First, each controller must be registered, which is represented in figure 6.14. This figure also shows that OpenAPI allows for routing customization, which was used with Ocelot, explained further in section ??, to redirect requests as well as increasing project readability.

```
1 [ApiController]  
2 [Route("api/report")]
```

Figure 6.14: OpenAPI Report Controller Registration

6.5.4 API Gateway

As defined in section 3.3, an API gateway is a point of entry to the services themselves. It allows for dynamic service instances and their locations definition. This means that when requests arrive at this point, they are routed to the appropriate service. This increases system complexity, which must be maintained.

Ocelot defines a point of entry to the defined microservices, particularly for systems developed in .Net, however, it is reliable in a REST based system. The implementation of two microservices makes the implementation of the Ocelot API gateway presented in figure 6.15.

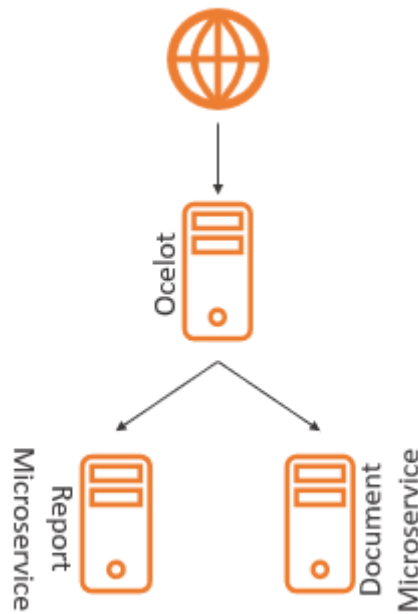


Figure 6.15: Ocelot Basic Implementation [Adapted from [64]]

Ocelot routing is based on these relevant concepts:

- **UpstreamPathTemplate:** defines the gateway's URL from which request are received then redirected to downstream.
- **UpstreamHttpMethod:** REST methods (such as GET, PUT, POST).
- **DownstreamPathTemplate:** microservice's endpoint that receives and is responsible for the request.
- **DownstreamScheme:** used protocol for communication.
- **DownstreamHostsAndPorts:** defines URL and the ports for microservices and receive requests.

These are used in figure 6.16 that shows the Get Reports routing example, which allows the requests to be redirected through. Specifically, a request is received in the upstream `https://localhost:4001/apigateway/reports` and is then redirected to the api's downstream defined port `https://localhost:4002/api/reports`.

```
1 "Routes": [{
2   "UpstreamPathTemplate": "/apigateway/reports",
3     "UpstreamHttpMethod": ["Get"],
4     "DownstreamPathTemplate": "/api/reports",
5     "DownstreamScheme": "https",
6     "DownstreamHostAndPorts": [{
7       "Host": "localhost",
8       "Port": 4002
9     }]
10 },
11
12 "GlobalConfiguration": {
13   "BaseUrl": "https://localhost:4001"
14 }
15 }]
```

Figure 6.16: Ocelot Basic Routing Example

6.5.5 Containerization

As referred previously in section 6.4, containerization was implemented in a modular approach, give the microservice approach. In the implemented microservices, the used databases were created through Docker client, given that it is a simple process. First, a MySQL docker image was downloaded and customized with the microservice name (container name, container password, and ports were edited), which must be defined within Dockerfile that actually builds the container image, then it is possible to deploy and connect to it via Docker Desktop.

Specifying ports and Dockefile is mandatory and very useful for connecting the database container to the application container via docker-compose file - generally running a multi-container system.

Regardless of the architectural style, it is a good practice to have one process in each container, given that each container has to encompass the necessary information about the process to manage and run it, however, microservices' modular and scalable scope encourages the use of the container per service approach, given that it allows for multiple versions to be implemented, individual scalability (having more than one applications running over the same database), and easier maintainability [21].

Chapter 7

Experimentation and Evaluation

This section presents the hypothesis investigation enumeration, their evaluation indicators, and how they are applied. Furthermore, this chapter's intention is to present a description of the adopted evaluation method, specifically by validation of stakeholders.

7.1 Hypothesis Identification and Evaluation Indicators

As previously mentioned, this thesis aims to identify good engineering practices and design patterns to apply in a microservices migration process, facilitating the system's modifiability and maintenance. Furthermore, identifying possible trade-offs in the implementation process and document decisions.

The Null Hypothesis (H0) evaluates the thesis' not perceived as valuable assets outcomes in the field. In contrast, the Alternative Hypothesis (H1) evaluates beneficial results.

- **Null Hypothesis (H0):**

- The system decomposition does not fulfill microservices evolution expectations.
- Stakeholders experience the legacy system's identified problems in the proposed solution;
- The solution does not comply with microservices principles;

- **Alternative Hypothesis (H1):**

- The system decomposition fulfills microservices evolution expectations.
- Stakeholders do not experience the legacy system's identified problems in the proposed solution;
- The solution complies with microservices principles;

7.2 Validation by Stakeholders

The identified hypothesis describes the possibility of no observable changes to the system and will be evaluated with this methodology.

Stakeholders validation is related to design adequacy and the already existing standard of comparison's performance that creates previous expectations. These derive from already established personal knowledge on the outcomes of "probabilistic situations" [36]. A questionnaire will be presented to stakeholders to test identified criteria satisfaction.

Considering that a great effort was made to identify system issues, it becomes necessary to evaluate the success of the implementation's outcomes. The stakeholders are a specific group that accompanied the project's development and were trusted to provide reliable feedback. This group comprises different profiles within RPM's context, such as managers, developers and quality assurance engineers with diverse experience with the legacy system, emphasizing that the questionnaire must be abstract and not compromise inquiry reliability.

Questions in the implemented questionnaire will be directly related to the defined hypothesis in section 7.1. These are the three topics that are present in the questionnaire:

- Questions regarding expectation fulfillment regarding microservices;
- Questions regarding the effect of the microservices in identified legacy problems;
- Questions regarding microservices principles implementation and identification;

The questionnaire is split into three sections, excluding the introduction. The first section entails a profile group of questions, which intend to clarify the stakeholder's job title, their experience, and their context within the company. The second section focuses on understanding use case validity and their adequacy considering initial identified issues. Finally, if the participant does have experience with microservices he is directed to a third and last section, which focuses on the implementation itself and its adequacy regarding microservices architecture theoretical specifications, specifically DDD, modularity/single responsibility principle, modifiability and maintainability.

As explained, the focus population of the questionnaire is FARO's employees who work with, or indirectly with, with RPM's legacy system. By being sent directly to them.

The questionnaire will entail scale responses between 0 and 5, with 0 being the lowest and five being the highest. The goal is to compare the average answer to the pre-determined average, which facilitates determining if stakeholders are satisfied. Table 7.1 specifies this scale.

Strongly Disapprove	Disapprove	Neutral	Approve	Strongly Approve
1	2	3	4	5

Table 7.1: Approval scale

By using this scale it is possible to assert if the proposed solution is considered adequate for the problem by stakeholders, presented in section 1.2, which includes RPM's team, CAM2's team (for their knowledge on the legacy application and issue identification), and management from both teams. The questionnaire's flow is available in appendix F.

7.3 Hypothesis Evaluation

As explained previously in section 7.1, H_0 is the null hypothesis and means that the proposed solution is not adequate, in contrast, H_1 represents adequacy. As this adequacy is evaluated by stakeholders through a questionnaire, the answer distribution will be presented. More specifically, H_0 and H_1 can be defined as:

H_0 : Stakeholders do not find the proposed solution adequate, when $H_0: \mu \leq 3$

H_1 : Stakeholders find the proposed solution adequate, when $H_1: \mu > 3$

The value of μ is calculated as the average of the answered questions, if that value is higher than 2.5, it confirms that the solution is adequate.

7.4 Questionnaire Answers Analysis

In this section, the answers to the questionnaire will be presented, analyzed and conclusions will be drawn considering the defined hypothesis. As said previously, the questionnaire can be seen fully in appendix F.

As the "About You" section focused on profiling the participant, the information that is possible to retrieve from the questionnaire, in which 11 answers were submitted from the CAM2 and RPM team, specifies the participant's experience, current job title, and the project in which they are working in. The responses from the questionnaire varied, considering the different positions within the team (junior and senior) this is the fluctuation table 7.2. From the 11 submissions, 2 submissions are from Quality Assurance Engineers (QA), 2 are from management (decision making positions), and the remaining 7 are from software developers, which can be seen graphically in graph 7.1.

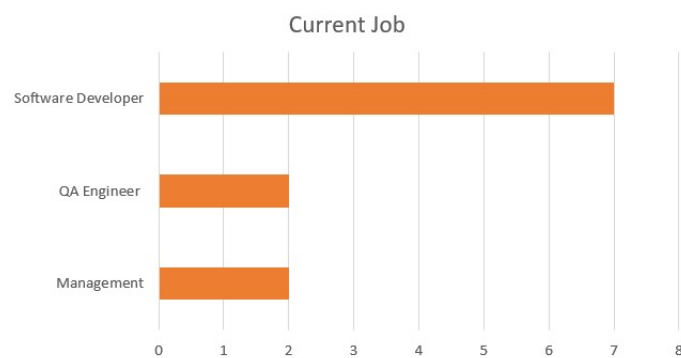


Figure 7.1: Job Title Division

Furthermore, 3 of the 11 people work directly in Repeat Part Management, and 8 work in CAM2. This distinction is related to team and project size, given that RPM's full team had 2 software developers, 1 QA, and 1 manager (who is currently working in another position in the company), which is specified in figure 7.2

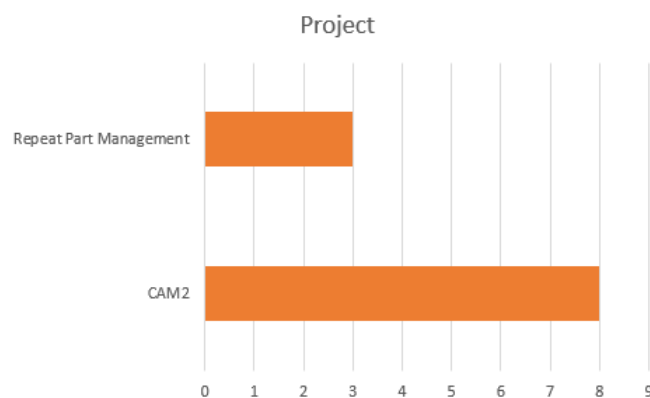


Figure 7.2: Project Division

Furthermore, from the participants submission data it is possible to conclude that management has increased experience (17 and 20 years), QA Engineers have 10 and 8 years of experience, and Software Developers are unevenly distributed, due to the existing levels distinction. Table 7.2 aggregates this data so it is easier to understand how it is distributed.

Experience years	Job title	Project
12	QA Engineer	Repeat Part Management
5	Software Developer	CAM2
7	Software Developer	Repeat Part Management
3	Software Developer	Repeat Part Management
17	Management	CAM2
2	Software Developer	CAM2
8	QA Engineer	CAM2
20	Management	CAM2
12	Software Developer	CAM2
6	Software Developer	CAM2
5	Software Developer	CAM2

Table 7.2: About You Data Aggregation

The remaining part of this section will be divided into the two most relevant sections of the questionnaire, which are crucial to evaluate the project's adequacy regarding the defined hypothesis: Identified Issues Section and Microservices Adequacy Section.

7.4.1 Identified Issues Section

Initially, the participants, that have access to the Design and Implementation chapters (chapters 5 and 6) are asked to rate if they agree with the adequacy of the microservices architecture. The results for this architectural questions are presented in table 7.3, that specifies a total of 8 strongly agree and 3 agree. This result was anticipated given that this architectural style was included within the presented drivers in section 5.1.2 as it was requested initially.

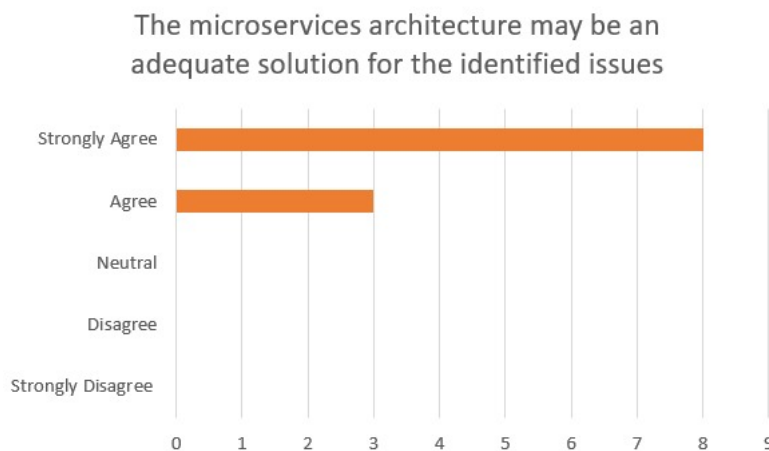


Figure 7.3: Microservices Adequacy Answers (2.1)

The requirements and design was a part of this section given that within developer teams there is knowledge basis on how to structure and present software architecture. Furthermore, both teams work with RPM's legacy system and are aware of the functionality pool, which was also presented in the requirements section in chapter 5.

So by analysing 7.4 and 7.5 it is possible to conclude that the participants abide by what was designed for RPM's legacy documented functionalities. Specifically, 9 people strongly agree that these functionalities were well-documented and adequately represent business needs and 8 people strongly agree and 3 people agree that the designed solution is prepared for the documented functionalities - which guarantees a certain level of trust within the design phase.

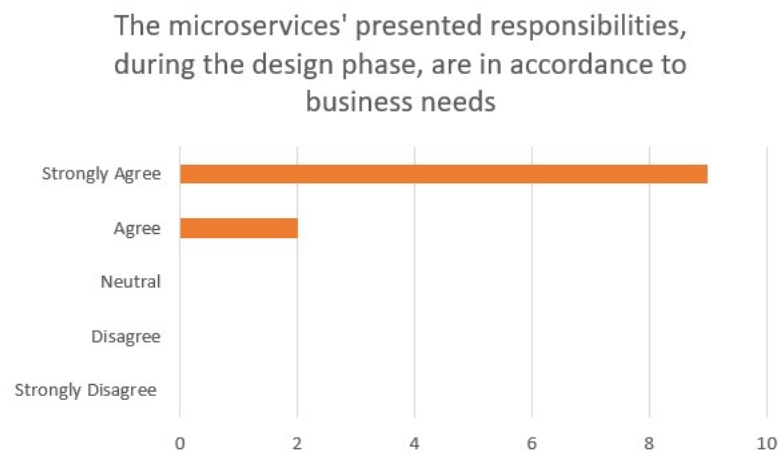


Figure 7.4: Microservices Responsibility Answers (2.2)

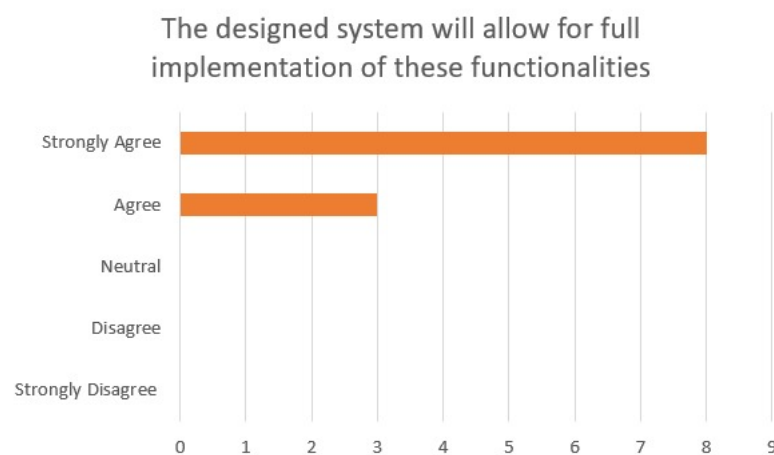


Figure 7.5: Functionality Implementation Answers (2.3)

Furthermore, as said previously, Microservices experience within the Faro context and, more specifically, the RPM's legacy context, is scarce, which makes it relevant for the question presented in 7.6, because, to answer questions in the next section, it is important that people have some knowledge on how microservices are implemented. However, this question also reduces our participant pool to 7 people in total for the implementation adequacy section.

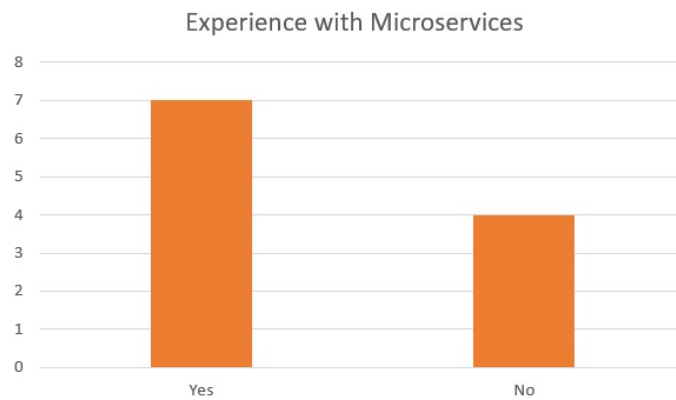


Figure 7.6: Microservices General Experience Answers (2.4)

7.4.2 Microservices Implementation Adequacy Section

From the previous section the participant population reduced to 7, which represent the ones that have any knowledge (theoretical or practical) on the microservices architecture and can give an informed opinion on the designed and implemented solution, which they had access to.

To proceed this section's first question was related to decomposition, which is associated with the design phase, by analysing figure 7.7 it is possible to assert that 6 people strongly agree that the designed solution applied DDD correctly, specifically concepts such as aggregates, bounded contexts, or entities, and 1 person agrees.

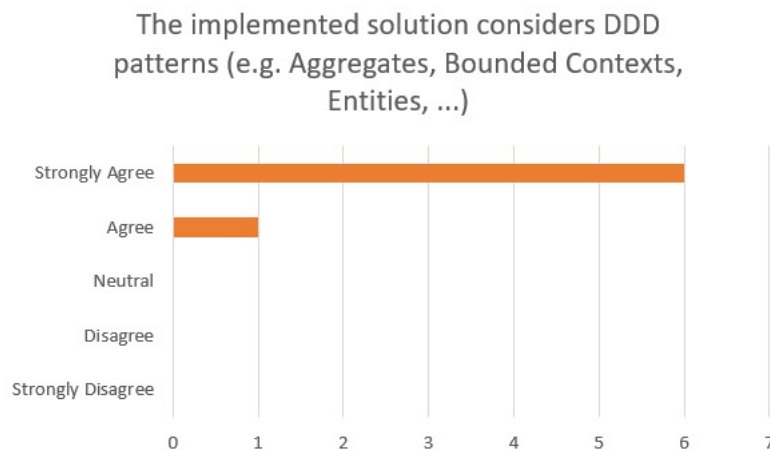


Figure 7.7: Microservices Pattern Adequacy Answers (3.1)

The next few questions relate to pattern consideration during the implementation and decomposition phases. Figure 7.8 relates to system modularity, and specifies that the answer must consider containerization (docker implementation, specified in chapter 6) and decomposition (design and concern distribution), in this questions the answers were all towards strongly agree, which is understandable given the applied containerization approach, which allows separate deployment of the system components.

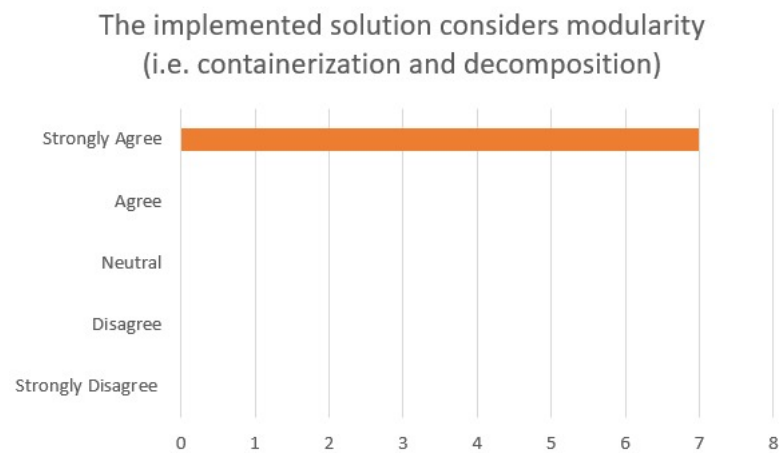


Figure 7.8: System Modularity Answers (3.2)

Figure 7.9 represents the modifiability question, which was one of the concerns identified by stakeholders initially. The purpose of this question is to understand the code and project readability and navigability, which may or not facilitate the process of adding or changing existing the system. In this question, 5 people strongly agree and 2 people agree with this characteristic.

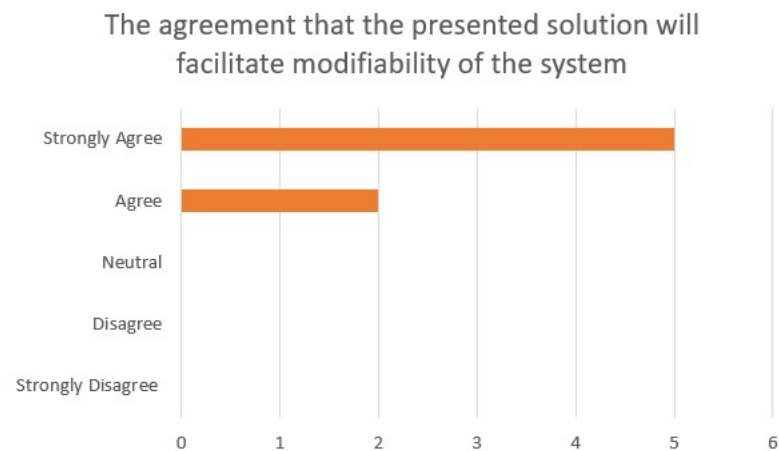


Figure 7.9: System Modifiability Answers (3.3)

Figure 7.10 relates to the maintainability characteristic which is associated with support and preservation of a system's ecosystem, so testing, deployment, or monitoring.

Maintainability is particularly laborious when compared to a monolith or a SOA architectural style because it takes more time to test a high number of small services, which need orchestration or other communication approaches (which is not the scope of this thesis) that require testing; furthermore, services can fail, and even if having more services means that it is somewhat fault-tolerant, it also means that the system must be prepared for when one of them is down to guarantee that the application does not fail completely. Deployment comes with its own downside, which is that to deploy the whole application the various components must be deployed separately.

The explained reasoning is able to clarify why the answers to this question were mainly neutral (5 answers) or agree (2 answers). These downsides were previously discussed during development.

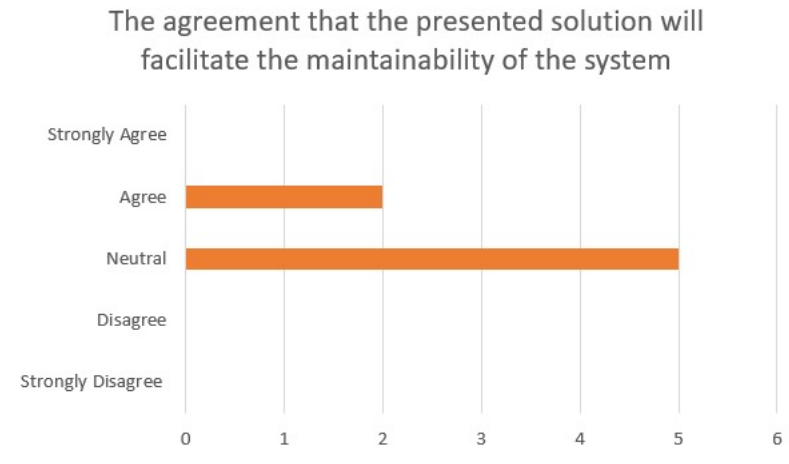


Figure 7.10: System Maintainability Answers (3.4)

7.5 Hypothesis Calculation

This section focuses on calculating the hypothesis by calculating the average value for each of the presented questions from the relevant sections, which are sections 7.4.1 and 7.4.2, that is why the identifiers start with 2 or 3. The values presented in table 7.1 are related to the used values for these calculations. Table 7.3 has all the average values per question and the overall average which will be the one considered to evaluate the hypothesis.

Question identification	Average
2.1	4.73
2.2	4.82
2.3	4.73
3.1	4.86
3.2	5
3.3	4.71
3.4	3.29
Total Average	4.59

Table 7.3: Per Question and Overall Average V alues

The overall average value is:

$$\mu = 4.59$$

This value is higher than 3, which means that H0 is refuted and that the stakeholders find the proposed solution to be adequate.

Chapter 8

Conclusion

This chapter intends to clarify project achievements regarding defined requirements presented in chapter 5. Any adversities or complications regarding this thesis development are described here.

8.1 Goal Achievement

This section will ponder on thesis goals achievement presented in chapter 1 in section 1.3. Regarding implementation achievements the final proposed solution is able to guide a future integration into a production environment, since it is currently a standalone system.

The established goals were to develop an application using the Microservices architecture by utilizing practices and design patterns considering the legacy RPM system (1), apply a decomposition strategy and a PoC (Proof of Concept) regarding the designed application (2), which are presented in table 8.1, that combines the used methodology.

Goal ID	Goal Description	Achievement
1	Design a modular system that considers the legacy RPM application.	Achieved
2	Decompose and evaluate the system.	Achieved

Table 8.1: Goal Achievement

As presented in the table above, 1 is totally achieved as presented in chapter 3 and 6, furthermore the second goal is also achieved, given that the initial goal was to decompose and design, is presented in chapter 6. However, the system was applied as a PoC system, with the intention of future integration in Faro's context.

The first goal was to design a system that considers the legacy system, the second goal was to decompose the system, which is documented in section 5.3 and its description is located within chapter 6. Finally, both goals were positively evaluated by the stakeholders with a 4.73 average, presented in chapter 7.

8.2 Threats to Validity

When considering the research, design, and implementation processes of thesis development, there are some relevant factors that hindered or impacted the taken and documented decisions and were not specified, such as:

- As the proposed internship was a target for alterations due to internal changes, the team that tackled RPM's maintenance presented a lack of general availability for further domain knowledge and technical support, which impacted the implementation, given that it was mainly done independently. Furthermore, gathering relevant information about a system with low communication frequency with developers can become less efficient;
- Low availability also made it a challenge to test this solution in a production environment in the set time frame;
- Developed use cases, specified in section 5.1.3, and as shown in the implementation chapter do not rely on each other, almost fully. This means that some relevant microservices patterns/strategies were not implemented, such as a message broker;
- As gathered previously, a big architectural smell regarding microservices migration is lack of microservices knowledge, because that may mean that future implementation may go further away from the microservices scope. In the RPM original team there was a low level of knowledge on migration strategies and maintenance of this architecture, which may hinder its internal implementation;
- Another threat to validity is the mandatory technological pool, which limits options related to microservices environment and did create a learning curve.

8.3 Future Work

Even though the previously identified goals were mostly achieved, as pondered in section 8.1, there are improvement prospects, regarding production environment testing. These are some of the possibilities:

- As this PoC was developed with the intent of integration, the recommendation of strategy would be the Strangler Pattern, which can be described as an iterative way of integrating systems, by replacing old code with new services (in the microservices specific case).
- As the increase of functionality will also involve more service communications Event Based Microservices is a good solution for communicating standalone microservices by applying the orchestration strategy.
- Furthermore, increasing the number of services will inherently be accompanied by the need for monitoring.
- As explained through Conway's Law, implementing microservices and the organization's structure are intimately related, which means that the teams' organization should be revised in order to apply this architecture and for it to be assimilated.
- Faro has CI/CD expert engineers, which must be able to apply a rigorous methodology that is understandable by all and guarantees the quality of the solution.

Bibliographic References

- [1] Muhammad Abdullah, Waheed Iqbal, and Abdelkarim Erradi. "Unsupervised learning approach for web application auto-decomposition into microservices". In: *Journal of Systems and Software* 151 (2019), pp. 243–257. issn: 01641212. doi: 10.1016/j.jss.2019.02.031. url: <https://doi.org/10.1016/j.jss.2019.02.031>.
- [2] *About DotNetZip*. url: <https://documentation.help/DotNetZip/About.htm>.
- [3] Nish Anil et al. *Applying simplified CQRS and DDD patterns in a microservice* / *Microsoft Docs*. Sept. 2021. url: <https://docs.microsoft.com/en-us/dotnet/architecture/microservices>.
- [4] ASQ. url: <https://asq.org/about-asq>.
- [5] ASQ. *Letter Q - Quality Glossary of Terms, Acronyms Definitions with Letter Q* / ASQ. url: <https://asq.org/quality-resources/quality-glossary/q>.
- [6] *Backends for Frontends pattern - Cloud Design Patterns* / *Microsoft Docs*. url: <https://docs.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends>.
- [7] Deepali Bajaj, Anita Goel, and S. C. Gupta. "GreenMicro: Identifying Microservices from Use Cases in Greenfield Development". In: *IEEE Access* 10.January (2022), pp. 67008–67018. issn: 21693536. doi: 10.1109/ACCESS.2022.3182495.
- [8] Luciano Baresi and Martin Garriga. "Microservices: The Evolution and Extinction of Web Services?" In: *Microservices: Science and Engineering* (Jan. 2020), pp. 3–28. doi: 10.1007/978-3-030-31646-4_1. url: https://link.springer.com/chapter/10.1007/978-3-030-31646-4_1.
- [9] Leonard Bass, Paul Clements, and Rick Kazman. "Software Architecture in Practice (3rd Edition)". In: *Architecture* (2013), p. 528.
- [10] Miguel Brito, Jácome Cunha, and João Saraiva. "Identification of microservices from monolithic applications through topic modelling". In: *Proceedings of the ACM Symposium on Applied Computing* (2021), pp. 1409–1418. doi: 10.1145/3412841.3442016.
- [11] Vincent Bushong et al. *On microservice analysis and architecture evolution: A systematic mapping study*. Sept. 2021. doi: 10.3390/app11177856.
- [12] *C++ libraries from vcpkg repository*. url: <https://vcpkg.info/port/polyclipping>.
- [13] Andrés Carrasco, Brent Van Bladel, and Serge Demeyer. "Migrating towards microservices: Migration and architecture smells". In: *IWoR 2018 - Proceedings of the 2nd International Workshop on Refactoring, co-located with ASE 2018* (Sept. 2018), pp. 1–6. doi: 10.1145/3242163.3242164. url: <https://doi.org/10.1145/3242163.3242164>.
- [14] Luiz Carvalho et al. "Analysis of the Criteria Adopted in Industry to Extract Microservices". In: *Proceedings - 2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice, CESSER-IP 2019* (2019), pp. 22–29. doi: 10.1109/CESSER-IP.2019.00012.

- [15] *Castle Core, including Castle DynamicProxy, Logging Services and DictionaryAdapter*. url: <https://github.com/castleproject/Core>.
- [16] Tomas Cerny, Michael J Donahoo, and Jiri Pechanec. "Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems". In: (2017). doi: 10.1145/3129676.3129682. url: <https://doi.org/10.1145/3129676.3129682>.
- [17] Thelma Colanzi et al. "Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices". In: *ACM International Conference Proceeding Series* October (2021), pp. 61–70. doi: 10.1145/3483899.3483904.
- [18] Lance Coleman. *The ASQ Certified Quality Auditor Handbook*. 2020. isbn: 978-1-951058-09-8. url: <https://asq.org/quality-resources/quality-assurance-vs-control>.
- [19] *Continuous-LINQ*. url: <https://kandi.openweaver.com/csharp/ismell/Continuous-LINQ>.
- [20] Geoff Coyle. "THE ANALYTIC HIERARCHY PROCESS (AHP)". In: (2004).
- [21] Arjav Dave. *How to Run Multiple Containers with Docker Compose*. Apr. 2022. url: <https://www.freecodecamp.org/news/run-multiple-containers-with-docker-compose/>.
- [22] Joseph DeFeo. *The Juran Trilogy: Quality Planning | Juran*. Apr. 2019. url: <https://www.juran.com/blog/the-juran-trilogy-quality-planning/>.
- [23] Hai Dinh-Tuan et al. "Development Frameworks for Microservice-based Applications: Evaluation and Comparison". In: *ACM International Conference Proceeding Series* (Nov. 2020), pp. 12–20. doi: 10.1145/3393822.3432339.
- [24] Mathias Douglas. *AutoMapper/LICENSE.txt at master · AutoMapper/AutoMapper · GitHub*. 2010. url: <https://github.com/AutoMapper/AutoMapper/blob/master/LICENSE.txt>.
- [25] *ElementFlow, FluidKit.Controls C (CSharp) Code Examples - HotExamples*. url: <https://csharp.hotexamples.com/examples/FluidKit.Controls/ElementFlow/-/php-elementflow-class-examples.html>.
- [26] Thomas Erl. *Service-Oriented Architecture*. Ed. by Greg Wiegand. Mark Taub, Dec. 2016. isbn: 978-0-13-385858-7. url: [https://sd.blackball.lv/library/Service-Oriented_Architecture__Analysis_and_Design_for_Services_and_Microservices_\(2017\).pdf#page=296&zoom=100,37,45](https://sd.blackball.lv/library/Service-Oriented_Architecture__Analysis_and_Design_for_Services_and_Microservices_(2017).pdf#page=296&zoom=100,37,45).
- [27] Essentra Components US. *What are screws heads, drives and threads?* Sept. 2021. url: <https://www.essentracomponents.com/en-us/news/guides/what-are-screws-heads-drives-and-threads> (visited on 09/09/2022).
- [28] Eric Evans. "Domain-Driven Design Reference: Definitions and Pattern Summaries". In: (2011), p. 62. url: https://books.google.com/books/about/Domain_Driven_Design_Reference.html?hl=pt-PT&id=ccRsBgAAQBAJ.
- [29] FARO. "FARO ® CAM2 ® Software". In: (2021). url: www.faro.com.
- [30] FARO. *FARO® CAM2® 3D Measurement Software*. url: <https://www.faro.com/en/Products/Software/CAM2-Software>.
- [31] FARO. *FARO® CAM2® 3D Measurement Software*. url: <https://www.faro.com/en/Products/Software/CAM2-Software>.
- [32] FARO. *Values | Company Profile*. url: <https://www.faro.com/en/About-Us/Company-Profile/Our-Values>.
- [33] *Fluorescence-Tools/mapack: Linear algebra library for .NET*. url: <https://github.com/Fluorescence-Tools/mapack>.

- [34] GENESIS. url: <https://dictionary.cambridge.org/pt/dicionario/ingles/genesis>.
- [35] Alexander Gillis. *What is Quality Assurance? - Definition from WhatIs.com*. url: <https://searchsoftwarequality.techtarget.com/>.
- [36] Gianluigi Guido. "Customer Satisfaction". In: *Wiley Encyclopedia of Management* (Jan. 2015), pp. 1–8. doi: 10.1002/9781118785317.WEOM090287. url: <https://onlinelibrary.wiley.com/doi/10.1002/9781118785317.weom090287>.
- [37] Stefan Haselböck, Rainer Weinreich, and Georg Buchgeher. "Decision guidance models for microservices - Service discovery and fault tolerance". In: vol. Part F130524. Association for Computing Machinery, Aug. 2017. isbn: 9781450348430. doi: 10.1145/3123779.3123804.
- [38] Stefan Haselböck, Rainer Weinreich, and Georg Buchgeher. "Decision Guidance Models for Microservices: Service Discovery and Fault Tolerance". In: *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems*. ECBS '17. Larnaca, Cyprus: Association for Computing Machinery, 2017. isbn: 9781450348430. doi: 10.1145/3123779.3123804. url: <https://doi.org/10.1145/3123779.3123804>.
- [39] Cornelius Herstatt and Birgit Verworn. "The 'Fuzzy Front End' of Innovation". In: *Bringing Technology and Innovation into the Boardroom* (2004), pp. 347–372. doi: 10.1057/9780230512771_16.
- [40] Humanizr/Humanizer. url: <https://github.com/Humanizr/Humanizer>.
- [41] IBM Cloud Team. *SOA vs. Microservices: What's the Difference?* May 2021. url: <https://www.ibm.com/cloud/blog/soa-vs-microservices> (visited on 09/04/2022).
- [42] ironfede/openmcdf: Microsoft Compound File .net component - pure C. url: <https://github.com/ironfede/openmcdf>.
- [43] ISO - Developing standards. url: <https://www.iso.org/developing-standards.html>.
- [44] Munezero Josélyne et al. *Partitioning Microservices: A Domain Engineering Approach*. May 2018.
- [45] Json.NET - Newtonsoft. url: <https://www.newtonsoft.com/json>.
- [46] Joseph Juran, John Wood, and Michael Wood. *Critical Evaluations in Business and Management*. 2005. url: https://books.google.pt/books?hl=pt-PT&lr=&id=v_fprq_XtFAC&oi=fnd&pg=PA50&dq=quality+control&ots=uDVqz-VPTk&sig=Yn2ts_Xs7BueSDXxFQAgX8v3Ln0&redir_esc=y#v=onepage&q=quality%20control&f=false.
- [47] John Klaess. *3 Definitions of Quality in Manufacturing and Why They Matter / Tulip*. Oct. 2021. url: <https://tulip.co/blog/definitions-of-quality-in-manufacturing/>.
- [48] P. Koen et al. "Providing Clarity and A Common Language to the "Fuzzy Front End"". In: <http://dx.doi.org/10.1080/08956308.2001.11671418> 44 (2 2016), pp. 46–55. issn: 08956308. doi: 10.1080/08956308.2001.11671418. url: <https://www.tandfonline.com/doi/abs/10.1080/08956308.2001.11671418>.
- [49] Alexander Krause et al. "Microservice Decomposition via Static and Dynamic Analysis of the Monolith". In: *Proceedings - 2020 IEEE International Conference on Software Architecture Companion, ICSA-C 2020* (2020), pp. 9–16. doi: 10.1109/ICSA-C50368.2020.00011. arXiv: 2003.02603.
- [50] Philippe Kruchten. "Architectural Blueprints-The "4+1" View Model of Software Architecture". In: *IEEE Software* 12 (6 1995), pp. 42–50.

- [51] Shanshan Li et al. "A dataflow-driven approach to identifying microservices from monolithic applications". In: *Journal of Systems and Software* 157 (2019). issn: 01641212. doi: 10.1016/j.jss.2019.07.008.
- [52] Welder Luz et al. "An experience report on the adoption of microservices in three Brazilian government institutions". In: *ACM International Conference Proceeding Series* (2018), pp. 32–41. doi: 10.1145/3266237.3266262.
- [53] Gaston Marquez and Hernan Astudillo. "Actual Use of Architectural Patterns in Microservices-Based Open Source Projects". In: *Proceedings - Asia-Pacific Software Engineering Conference, APSEC 2018-December* (July 2018), pp. 31–40. issn: 15301362. doi: 10.1109/APSEC.2018.00017.
- [54] Robin Martin. *Designing object-oriented C++ applications using the Booch method*. Englewood Cliffs, N.J. : Prentice Hall, 1995.
- [55] *Math.NET Numerics*. url: <https://numerics.mathdotnet.com/>.
- [56] *MIConvexHull*. url: <https://designengrllab.github.io/MIConvexHull/>.
- [57] Microservices.io. *Decompose by business capability*. url: <https://microservices.io/patterns/decomposition/decompose-by-business-capability.html>.
- [58] Microsoft. *Entity Framework Core*. May 2021. url: <https://learn.microsoft.com/en-us/ef/core/>.
- [59] Lawrence Miles. *Techniques of Value Analysis and Engineering*. 3rd ed. Miles Value Foundation, June 2015, pp. 1–4. isbn: 0070419264, 9780070419261. url: https://books.google.pt/books?hl=pt-PT&lr=&id=XxhbDwAAQBAJ&oi=fnd&pg=PT17&dq=Value+Analysis&ots=btXeLwpcNF&sig=TYPvhHK0lhV36LhqUC8yyQp2lJg&redir_esc=y#v=onepage&q=Value%20Analysis&f=false.
- [60] Keith Murphy. *What Is Technical Debt and How to Manage It*. June 2021. url: <https://www.outsystems.com/blog/posts/technical-debt/>.
- [61] Davide Neri et al. "Design principles, architectural smells and refactorings for microservices: a multivocal review". In: *Software-Intensive Cyber-Physical Systems* 35 (1-2 Aug. 2020), pp. 3–15. issn: 25248529. doi: 10.1007/S00450-019-00407-8/FIGURES/5. url: <https://link.springer.com/article/10.1007/s00450-019-00407-8>.
- [62] Susana Nicola. "ANÁLISE DE VALOR INESC-TEC". In: (2021).
- [63] Christian Nielsen. *Frameworks for understanding and describing business models*. Mar. 2014. url: https://www.researchgate.net/publication/273634456_Frameworks_for_understanding_and_describing_business_models.
- [64] Tom Pallister. *Ocelot Documentation Release 1.0.0*. 2022.
- [65] Neil Peterson and Adam Boeglin. *Domain analysis for microservices*. Dec. 2021. url: <https://docs.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis>.
- [66] John Piela. *Why Netflix Moved to a Microservices Architecture | ProgrammableWeb*. Apr. 2016. url: <https://www.programmableweb.com/news/why-netflix-moved-to-microservices-architecture/elsewhere-web/2016/04/02>.
- [67] Johnson Pontus and Mathias Ekstedt. "The grand unified theory of software engineering". In: (2005), p. 235. url: https://books.google.com/books/about/The_grand_unified_theory_of_software_eng.html?hl=pt-PT&id=TLcceL3NEiMC.
- [68] Peihua Qiu. *Introduction to Statistical Process Control*. CRC Press, 2013. url: <https://books.google.pt>.
- [69] *QR.net - Your QR code generator to create QR codes*. url: <https://qr.net/>.
- [70] Chris Richardson. *Microservices patterns with examples in Java*. Manning Publications, 2019. isbn: 1638356327.

- [71] Anna Rud. *Why and How Netflix, Amazon, and Uber Migrated to Microservices: Learn from Their Experience – HYS Enterprise*. July 2019. url: <https://www.hys-enterprise.com/blog/why-and-how-netflix-amazon-and-uber-migrated-to-microservices-learn-from-their-experience/>.
- [72] J. P. Russel. *The ASQ auditing handbook*. fourth. William A. Tony, 2012. url: <https://pdfroom.com/books/the-asq-auditing-handbook-principles-implementation-and-use/Pe5xQj0GdnN>.
- [73] Thomas L Saaty. "Decision making with the analytic hierarchy process". In: *Int. J. Services Sciences* 1 (1 2008), pp. 83–98.
- [74] Alberto Sampaio. "Improving Systematic Mapping Reviews". In: *ACM SIGSOFT Software Engineering Notes* 40.6 (Nov. 2015), pp. 1–8. issn: 0163-5948. doi: 10.1145/2830719.2830732. url: <https://dl.acm.org/doi/10.1145/2830719.2830732>.
- [75] *Serilog — simple .NET logging with fully-structured events*. url: <https://serilog.net/>.
- [76] Smartbear. *OpenAPI Specification*. url: <https://swagger.io/specification/>.
- [77] Larry W Smith. "Project Clarity Through Stakeholder Analysis". In: (2000). url: www.stsc.hill.af.mil.
- [78] Steve Smith. *Common web application architectures | Microsoft Learn*. Sept. 2022. url: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>.
- [79] César Souza. *The Accord.NET Framework*. 1999. url: <http://accord-framework.net/license.txt>.
- [80] Michel Thiry. *Project Management Institute A Framework for Value Management Practice*. 2nd. Project Management Institute, Inc, 2013. isbn: 9781628250183. url: www.PMI.org.
- [81] Johannes Thönes. "Microservices". In: *IEEE Software* 32 (1 Jan. 2015). issn: 07407459. doi: 10.1109/MS.2015.11.
- [82] Rafik Tighilt et al. "On the Study of Microservices Antipatterns: A Catalog Proposal". In: *ACM International Conference Proceeding Series* (July 2020). doi: 10.1145/3424771.3424812.
- [83] Ken Tucker, Corey Vincent, and Nigel Sampson. *Caliburn.Micro*. url: <https://caliburnmicro.com/>.
- [84] John Wade. *Greenfield vs. Brownfield Software Development*. Sept. 2018. url: <https://synoptek.com/insights/it-blogs/greenfield-vs-brownfield-software-development/>.
- [85] Rob Wojcik et al. "Attribute-Driven Design (ADD), Version 2.0 Software Architecture Technology Initiative". In: (2006). url: <http://www.sei.cmu.edu/publications/pubweb.html>.
- [86] Daniele Wolfart et al. "Modernizing legacy systems with microservices: A roadmap". In: *ACM International Conference Proceeding Series* (2021), pp. 149–159. doi: 10.1145/3463274.3463334.
- [87] Tony Woodall. "Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis". In: *Academy of Marketing Science Review* 12 (Jan. 2003).

Appendix A

Task Goals

A.1 Tasks

The goal is to directly relate these tasks to them whether the relation is due to implication (drivers implicates the implementation of given requirement), due to its significance in implementation, or even influence/awareness in implementation. In table A.1 those relations are set for the non-functional requirements, while in table A.2 the functional requirements are specified.

Requirement	Concern	Constraint	QA
T1	CRN03		QA01 QA02
T2	CRN01 CRN03	CON03	QA02 QA03 QA04
T2.1.	CRN01 CRN03	CON03	QA01 QA03 QA04
T3	CRN03 CRN05	CON03	QA01 QA02 QA03 QA04
T3.1.	CRN01 CRN04 CRN05	CON03	QA01 QA02 QA03 QA04

Table A.1: Architectural drivers and non-functional requirements relation

Requirement	Concern	Constraint	QA
TF1	CRN01 CRN03	CON01 CON02 CON03	QA01 QA02 QA03 QA04
TF2	CRN02 CRN04	CON01 CON03 CON04 CON06	QA01
TF3	CRN04	CON01 CON02 CON03 CON04	QA02 QA04
TF4	CRN02	CON02 CON03 CON04 CON06	QA02 QA04
TF5	CRN01 CRN02 CRN05	CON03 CON04	QA01 QA02 QA03 QA04

Table A.2: Architectural drivers and functional requirements relation

Appendix B

Auxiliary AHP Data

B.1 Random Index (RI) Table

Table B.1 encompasses the reference values for Indexes.

1	2	3	4	5	6	7	8
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.42

Table B.1: Random Index (RI) Table [Adapted from [62]]

B.2 Criteria Normalized Matrices

Tables B.2, B.3, B.4, and B.5 represent the normalized matrices values.

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1/7	1/7	1/7
Implement Microservices	6/7	6/7	6/7

Table B.2: Relevance Comparison Normalized Matrix

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1/6	1/6	1/6
Implement Microservices	5/6	5/6	5/6

Table B.3: Scalability Comparison Normalized Matrix

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1/5	1/5	1/5
Implement Microservices	4/5	4/5	4/5

Table B.4: Usability Comparison Normalized Matrix

	Maintain Current Solution	Implement Microservices	Priority
Maintain Current Solution	1/3	1/3	1/3
Implement Microservices	2/3	2/3	2/3

Table B.5: Reliability Comparison Normalized Matrix

Appendix C

Architectural Patterns Research Results

C.1 Monitoring Patterns

This section specifies patterns used in the monitorization process of the development cycle, which tracks modifications and behavior.

1. **Distributed Logging:** Microservices produce information logged to local storage, limiting database usage, hindering data aggregation, and inherently analyzing it. Implementing logging systems responsible for log aggregation in a single repository allows for easier consistency maintenance in the formatting and monitoring process. However, log repositories require infrastructure configuration and uniformization of logging ([82]).
2. **Monitoring:** Given the basis of the microservices concept as communicating services that fulfill one business capability ([82]), their behavior and performance monitorization becomes essential to guarantee availability. The implementation of logging solely is not sufficient in the absence of health checking approaches. Insufficient monitoring hinders failure tracking and causes identification ([82]). It is essential to consider that this pattern also requires configuration, maintenance, and microservice refactoring.

Appendix D

RPM's Third Party Software

D.1 Third Party Software

This section presents, in table D.1 the legacy solution's third party softwares.

Software	Description
Accord Framework	The core of the Accord.NET Framework, which contains basic classes such as general exceptions and extensions used by other framework libraries ([79]).
AutoMapper	A convention-based object to object mapper that uses a fluent configuration API to define mapping strategies and algorithm to match source destination values ([24]).
Caliburn Micro	A small yet powerful framework designed for building applications across XAML platforms. Its strong support for MV* patterns will enable you to build your solution quickly, without the need to sacrifice code quality or testability ([83]).
Castle Core	Castle Core, including DynamicProxy, Logging Abstractions and Dictionary Adapter ([15]).
Continuous LINQ	Continuous LINQ is a .NET Framework 3.5 extension that builds on the LINQ query syntax to create continuous, self-updating result sets. With Continuous LINQ, you write a query and the results of that query are continuously updated as changes are made to the source collection or items within the source collection. CLINQ has tremendous value in GUI development and is especially useful in binding to filtered streams of data such as financial or other network message data ([19]).
DotNetZip	DotNetZip is a fast, free class library and toolset for manipulating zip files. Use VB, Csharp or any .NET language to easily create, extract, or update zip files ([2]).
FluidKit Controls	This is a WPF library containing a powerhouse of controls, frameworks, helps, tools, etc. for productive WPF development ([25]).
Humanizer	Humanizer meets all .NET needs for manipulating and displaying strings, enums, dates, times, timespans, numbers and quantities ([40]).

MIConvexHull	This project is a convex hull algorithm and library for 2D, 3D, and higher dimensions. The code can also be used to compute Delaunay triangulations and Voronoi meshes of the input data ([56]).
Mapack	Mapack is a .NET class library for basic linear álgebra computations ([33]).
Math.NET Numerics	Math.NET Numerics is the numerical Foundation of the Math.NET project, aiming to provide methods and algorithms for numerical computations in science, engineering and every day use ([55]).
Newtonsoft.Json	Json.NET is a popular high-performance JSON framework for .NET [45].
OpenMCDF	This is managed by .net component that allows client application to manipulate COM structured storage files, also known as Microsoft Compound Document Format files ([42]).
Polyclipping	The Clipper library perfoms clipping and offsetting for both and polygons, All four Boolean clipping operations are supported – intersection, union, difference and exclusive-or. Polygons can be of any shape including self-intersecting polygons ([12]).
QrCode.Net	The goal of the project is providing na easy to use, fully managed .NET library for handling QR code according to ISO/IEC 18004 ([69]).
Serilog	Simple .NET logging with fully-structured events ([75]).

Table D.1: Third Party Software in the Legacy System

Appendix E

Conducting the Search for Primary Studies

E.1 Primary Studies

In this section the primary studies are listed in tables E.1 and E.2 by alphabetical order.

Title	Author(s)	Publication Year
A dataflow-driven approach to identifying microservices from monolithic applications	Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao	2019
Analysis of the criteria adopted in industry to extract microservices	Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello	2019
An experience report on the adoption of microservices in three Brazilian government institutions	Welder Luz, Everton Agilar, Marcos César de Oliveira, Carlos Eduardo R. de Melo	2018
Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices	Thelma Colanzi, Aline Amaral, Wesley Assunção, Arthur Zavadski	2021
Deployment and communication patterns in microservice architectures: A systematic literature review	Işıl Aksakalli, Turgay Çelik, Ahmet Burak Can, Bedir Tekinerdoğan	2021
Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study	Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rodrigo Bonifácio	2019
GreenMicro: Identifying Microservices From Use Cases in Greenfield Development How Can We Cope with the Impact of Microservice Architecture Smells?	Deepali Bajaj, Anita Goel, S. C. Gupta	2022

Table E.1: Primary Studies

Title	Author(s)	Publication Year
How Can We Cope with the Impact of Microservice Architecture Smells?	Xiang Ding, Cheng Zhang	2022
Identification of microservices from monolithic applications through topic modelling	Miguel Brito, Jácome Cunha, João Saraiva	2021
Leveraging the Layered Architecture for Microservice Recovery	Pascal Zaragoza, Abdelhak-Djamel Seriai, Abderrahmane Seriai, Anas Shatnawi, Mustapha Derras	2022
Microservice Decomposition via Static and Dynamic Analysis of the Monolith	Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, Dan Kröger	2020
Modernizing Legacy Systems with Microservices: A Roadmap	Daniele Wolfart, Wesley K. G. Assunção, Ivonei F. da Silva, Diogo Domingos	2021
Monolith to Microservice Candidates using Business Functionality Inference	Shivali Agarwal, Rounak Sinha, Giriprasad Sridhar, Pratap Das, Utkarsh Desai, Srikanth Tamilselva, Amith Singhee, Hiroaki Nakamura	2021
Mono2Micro: a practical and effective tool for decomposing monolithic Java applications to microservices	Kalia Anup, Rahul Jin Xiao, Maja Saurabh Sinha	2021
MSN: A Playground Framework for Design and Evaluation of MicroServices-Based sdN Controller	Sisay Arzo, Domenico Scotece, Riccardo Bassoli, Daniel Barattini, Fabrizio Granelli, Luca Foschini, Frank Fitzek	2022
On revisiting energy and performance in microservices applications: A cloud elasticity-driven approach	Igor Fontana de Nardin, Rodrigo da Rosa Righi, Thiago Lopes	2021
QoS-aware placement of microservices-based IoT applications in Fog computing environments	Samodha Pallewatta, Vassilis Kostakos, Rajkumar Buyya	2022
The parallel agile process: Applying parallel processing techniques to software engineering	Raffo David, Doug Rosenberg, Barry W. Boehm, Bo Wang. Kan Qi	2019
Unsupervised learning approach for web application auto-decomposition into microservices	Muhammad Abdullah, Waheed Iqbal, Abdelkarim Erradi	2019

Table E.2: Primary Studies (continued)

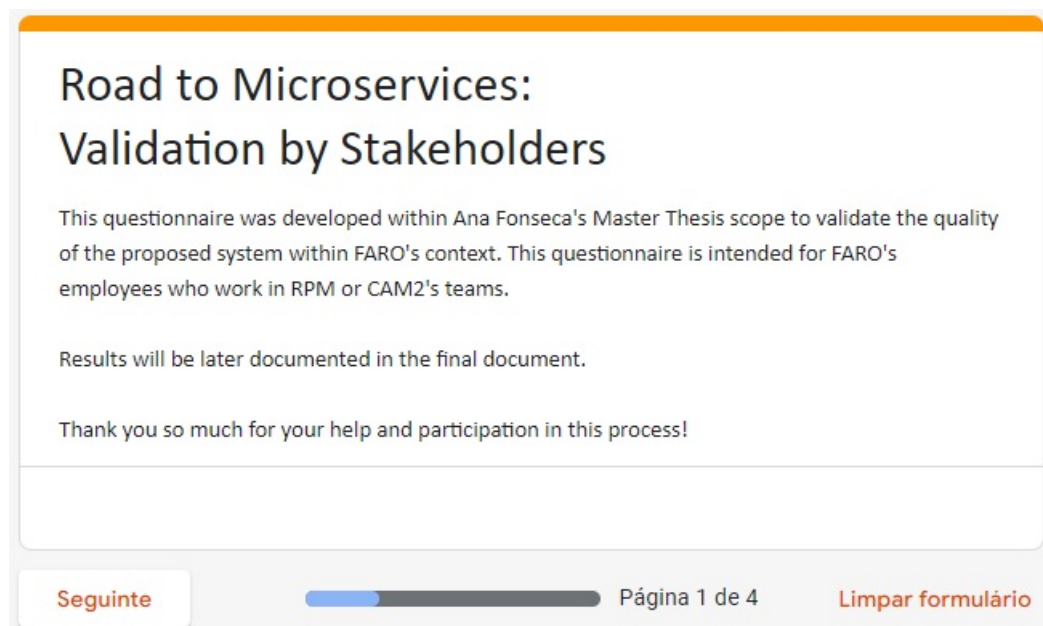
Appendix F

Evaluation

F.1 Stakeholder Questionnaire

In this appendix, the stakeholder questionnaire will be presented by section. The questionnaire is split into three sections, disregarding the introduction, which has no related questions.

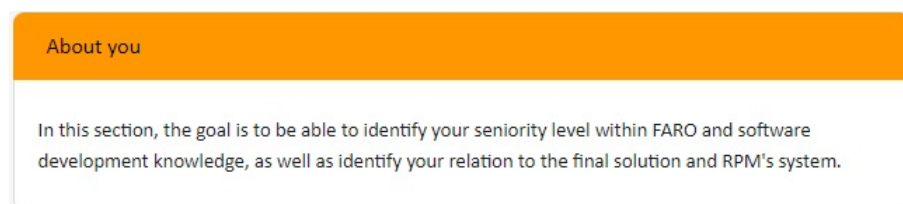
Figure F.1 is the presented introductory text.



The screenshot shows a web form titled "Road to Microservices: Validation by Stakeholders". The text inside the form reads: "This questionnaire was developed within Ana Fonseca's Master Thesis scope to validate the quality of the proposed system within FARO's context. This questionnaire is intended for FARO's employees who work in RPM or CAM2's teams." followed by "Results will be later documented in the final document." and "Thank you so much for your help and participation in this process!". At the bottom, there is a "Seguinte" button, a progress bar, "Página 1 de 4", and a "Limpar formulário" button.

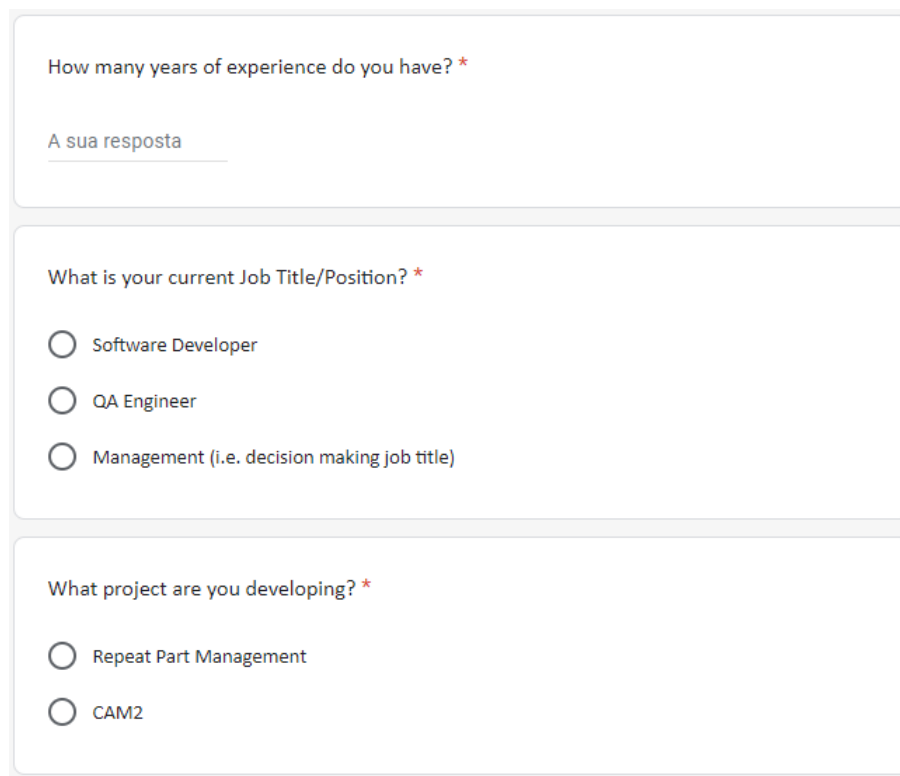
Figure F.1: Questionnaire Introductory Text

When clicking next, it redirects the participant to the next section, which focuses on acquiring the participants profile. It's description is in figure F.2 and questions in figure F.3.



The screenshot shows a web form titled "About you". The text inside the form reads: "In this section, the goal is to be able to identify your seniority level within FARO and software development knowledge, as well as identify your relation to the final solution and RPM's system."

Figure F.2: About You Introductory Text



How many years of experience do you have? *

A sua resposta _____

What is your current Job Title/Position? *

☐ Software Developer

☐ QA Engineer

☐ Management (i.e. decision making job title)

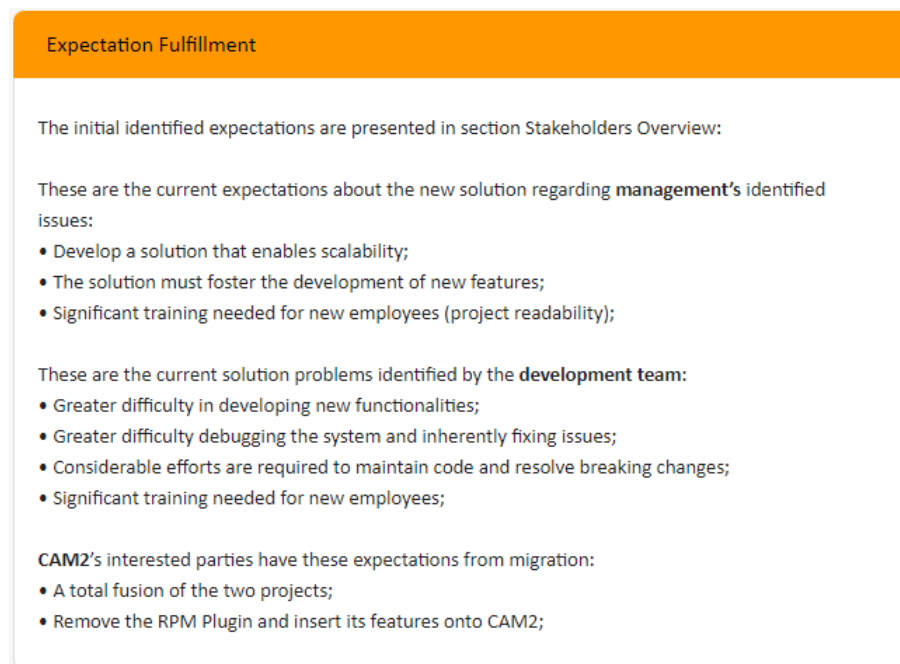
What project are you developing? *

☐ Repeat Part Management

☐ CAM2

Figure F.3: About You Questions

When the participant is finished with the about you mandatory questions, the questionnaire goes to the Expectation Fulfillment section, present in figures F.4, F.5, and F.6.



Expectation Fulfillment

The initial identified expectations are presented in section Stakeholders Overview:

These are the current expectations about the new solution regarding **management's** identified issues:

- Develop a solution that enables scalability;
- The solution must foster the development of new features;
- Significant training needed for new employees (project readability);

These are the current solution problems identified by the **development team**:

- Greater difficulty in developing new functionalities;
- Greater difficulty debugging the system and inherently fixing issues;
- Considerable efforts are required to maintain code and resolve breaking changes;
- Significant training needed for new employees;

CAM2's interested parties have these expectations from migration:

- A total fusion of the two projects;
- Remove the RPM Plugin and insert its features onto CAM2;

Figure F.4: Expectations Introductory Text

Do you agree that the microservices architecture may be an adequate solution for the identified issues?

1 2 3 4 5

☐ ☐ ☐ ☐ ☐

Do you agree that the microservices' presented responsibilities, during the design phase, ^{*} are in accordance to business needs?

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ ☐ Strongly Agree

Figure F.5: Expectations Questions - Part 1

Do you agree that the designed system will allow for full implementation of these functionalities?

```

graph LR
    Client((Client)) --- UC1([UC1 : Access published documents])
    Client --- UC2([UC2: Check for available programs])
    Client --- UC3([UC3: Run Programs])
    Client --- UC3_1([UC3.1: Visualize running program])
    Client --- UC4([UC4: Check for program's results statistics])
    Client --- UC5([UC5: Create reports regarding results])
    Client --- UC5_1([UC5.1: Choose a report template from existing options])
  
```

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ ☐ Strongly Agree

Do you have experience with Microservices? ^{*}

☐ Yes

☐ No

Figure F.6: Expectations Questions - Part 2

If during the previous Expectations section, the participant confirmed microservice experience, the questionnaire redirects to a third section: Evaluation regarding Implementation, which is presented in figures F.7, F.8, and F.9.

Evaluation regarding Implementation

In this section the focus is on the implementation itself and its adequacy regarding microservices architecture theoretical specifications, specifically DDD, modularity/single responsibility principle, modifiability and maintainability.

Figure F.7: Implementation Introductory Text

Do you think that the implemented solution considers DDD patterns (e.g. Aggregates, Bounded Contexts, Entities, ...)? *

1

2

3

4

5

Strongly Disagree

☐
☐
☐
☐
☐

Strongly Agree

Do you think that the implemented solution considers modularity (i.e. containerization and decomposition)? *

1

2

3

4

5

Strongly Disagree

☐
☐
☐
☐
☐

Strongly Agree

Figure F.8: Implementations Questions - Part 1

Do you think that the presented solution will facilitate modifiability of the system? *

1

2

3

4

5

Strongly Disagree

☐
☐
☐
☐
☐

Strongly Agree

Do you think that the presented solution will facilitate maintainability of the system? *

1

2

3

4

5

Strongly Disagree

☐
☐
☐
☐
☐

Strongly Agree

Figure F.9: Implementations Questions - Part 2