Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Programa de Pós-Graduação em Sistemas e Computação
Mestrado Acadêmico em Sistemas e Computação

A Migration Metaprocess from a Monolithic Architecture to Microservices

Henrique David de Medeiros

Natal-RN

Junho de 2025

Henrique David de Medeiros

A Migration Metaprocess from a Monolithic Architecture to Microservices

Qualificação de Mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Sistemas e Computação.

Linha de pesquisa:
Sistemas Integrados e Distribuídos

Orientador
Prof.ª Dra. Thais Vasconcelos Batista

Coorientador
Prof. Dr. Everton Ranielly de Sousa Cavalcante

PPgSC – Programa de Pós-Graduação em Sistemas e Computação

CCET – Centro de Ciências Exatas e da Terra

UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Junho de 2025

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Medeiros, Henrique David de.
A migration metaprocess from a monolithic architecture to microservices / Henrique David de Medeiros. - 2025.
114 f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-graduação em Sistemas e Computação. Natal, RN, 2025.
Orientação: Profa. Dra. Thais Vasconcelos Batista.
Coorientação: Prof. Dr. Everton Ranielly de Sousa Cavalcante.

1. Computação - Dissertação. 2. Microservices - Dissertação. 3. Migration - Dissertação. 4. Legacy systems - Dissertação. 5. Software modernization - Dissertação. 6. Metaprocess - Dissertação. I. Batista, Thais Vasconcelos. II. Cavalcante, Everton Ranielly de Sousa. III. Título.

RN/UF/CCET CDU 004(043.3)

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

HENRIQUE DAVID DE MEDEIROS

"A Metaprocess for Migrating Monolithic Architectures to Microservices"

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Sistemas e Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte.

Prof. Dr. EVERTON RANIELLY DE SOUSA CAVALCANTE
Coordenador do PPgSC

Banca Examinadora:

gov.br Documento assinado digitalmente
CARLOS ANDRE GUIMARAES FERRAZ
Data: 26/06/2025 18:40:23-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. CARLOS ANDRÉ GUIMARÃES FERRAZ**

gov.br Documento assinado digitalmente
EVERTON RANIELLY DE SOUSA CAVALCANTE
Data: 26/06/2025 18:28:46-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Interno(a): **Dr. EVERTON RANIELLY DE SOUSA CAVALCANTE**

gov.br Documento assinado digitalmente
THAIS VASCONCELOS BATISTA
Data: 16/08/2025 11:27:14-0300
Verifique em https://validar.iti.gov.br

Presidente: **Dr.ª THAIS VASCONCELOS BATISTA**

gov.br Documento assinado digitalmente
HENRIQUE DAVID DE MEDEIROS
Data: 21/08/2025 13:08:56-0300
Verifique em https://validar.iti.gov.br

Discente: **HENRIQUE DAVID DE MEDEIROS**

Natal, 26 de junho de 2025.

A Migration Metaprocess from a Monolithic Architecture to Microservices

Author: Henrique David de Medeiros
Advisor: Thais Vasconcelos Batista
Co-advisor: Everton Ranielly de Sousa Cavalcante

Abstract

Microservice-based architectures have gained popularity for their ability to handle the complexity of cloud service-oriented systems and meet requirements for availability, maintainability, and scalability. However, the process of modernizing legacy systems to adopt microservices is often time-consuming and lacks a systematic approach. There is a need for clear guidance on which activities and artifacts are essential when migrating applications to microservices and how to establish a well-defined process to achieve the expected benefits. This work introduces the Metaprocess for Microservice Migration Kernel (M3K), which serves as a foundation for migrating monolithic applications to microservice architectures. The primary objective of this proposal is to establish a comprehensive metaprocess that development teams and organizations can use as a basis for defining their microservice migration processes. M3K leverages OMG's Essence Standard, providing a solid framework for defining software development practices, activities, and work products, and facilitating the smooth transition to microservice architectures.

Keywords: Microservices, migration, legacy systems, software modernization, metaprocess.

A Migration Metaprocess from a Monolithic Architecture to Microservices

Autor: Henrique David de Medeiros
Orientador: Thais Vasconcelos Batista
Coorientador: Everton Ranielly de Sousa Cavalcante

Resumo

As arquiteturas baseadas em microsserviços ganharam popularidade por sua capacidade de lidar com a complexidade dos sistemas orientados a serviços em nuvem e atender aos requisitos de disponibilidade, manutenibilidade e escalabilidade. No entanto, o processo de modernização de sistemas legados para adotar microsserviços costuma ser demorado e carece de uma abordagem sistemática. Há necessidade de uma orientação clara sobre quais atividades e artefatos são essenciais para se migrar aplicações para microsserviços e como estabelecer um processo bem definido para alcançar os benefícios esperados da arquitetura. Este trabalho apresenta o Metaprocesso para o Kernel de Migração para Microsserviços (M3K), que serve como base para a migração de aplicações monolíticas para arquiteturas de microsserviços. O objetivo principal deste trabalho é estabelecer um metaprocesso abrangente que equipes de desenvolvimento e organizações possam usar como base para definir seus processos de migração de microsserviços. O M3K aproveita o *Essence Standard* da OMG, fornecendo uma estrutura sólida para definir práticas, atividades e produtos de trabalho de desenvolvimento de software e facilitar a transição para arquiteturas de microsserviços.

Palavras-chave: microserviço, migração, sistemas legados, modernização de software, metaprocesso.

List of Figures

List of Tables

Acronyms

BPMN Business Process Model and Notation

M3K Metaprocess for Microservice Migration Kernel

OMG Object Management Group

REST REpresentational State Transfer

SOA Service-oriented Architecture

SPEM Software & Systems Process Engineering Metamodel

Contents

1 Introduction

Software modernization is defined as major changes to software, either in the technical context or in the business logic of the organization using the software, e.g. the migration process from legacy application to microservice architecture. Some criteria are fundamental for the decision to modernize applications, such as the usability of the system, the end of technological support, and changes in business processes (Koskinen et al., 2005). As applications grow in size and complexity, academia and industry have acknowledged that traditional monolithic applications − a single deployment unit, where all functionality in the system is deployed together (Newman, 2019) − have limitations in terms of scalability, maintainability, and innovativeness. In this context, the modernization of monolithic legacy systems to microservices emerges as an approach to improve the flexibility, scalability, and efficiency of systems, allowing companies to remain competitive in a constantly evolving business environment.

Microservices are autonomous and independent services with reduced size and complexity, implemented and deployed separately, each offering specific functionalities (Richardson, 2018). Microservice architecture contributes to managing growing complexity, decomposing large distributed systems into a set of small services. As the services are small, they are easy to maintain, fast to run, and benefit code understandability. Moreover, microservices allow each service to be independently scalable and best suited to its resource requirement, better fault isolation, where one misbehaving component will only affect that specific service, and better for experimenting and adopting new

technologies, choosing to use whatever language and frameworks that suit the service.

Software migration is defined as the division and transfer of software to a new platform or technology to satisfy new requirements (Wagner, 2014) and although the migration from service-oriented architectures to microservices is widely endorsed, it brings several challenges, such as the choice of the right services, the complexity in distributed systems, and when to adopt microservices (Richardson, 2018).

Choosing the right services is one of the challenges exploited in the microservice literature. The idea is to try to define an algorithm for decomposing a legacy system into services, avoiding incorrect decomposition that yields a system with coupled services that must be deployed together. The adoption of microservices relies on understanding the business needs and why microservice is the best solution for the problem. The process needs to consider that not every application benefits from the microservice architectural style, and it's necessary to make careful decisions before investing efforts to migrate applications to microservices.

Another important challenge is related to the deployment of functionalities that span multiple services. It is necessary a careful coordination between various development teams, where a rollout plan must be created to sequence the service deployments according to the dependencies between services.

Modernizing legacy systems to microservices is a non-trivial and time-consuming task (Taibi et al., 2017), involving application decoupling, database migration, data splitting, communication among services, effort estimation, DevOps infrastructure effort, and people's minds. Modernization does not mean that the legacy application is outdated, but that the quality of attributes are not more satisfied by the legacy architecture. The lack of understanding of which activities and artifacts should be considered during the application migration may hamper the expected benefits, mainly because there is no understanding of migration methodologies or consensus on the efforts on migration

procedures (Hassan et al., 2020).

Some well-known huge companies, such as Uber, Amazon, and Netflix, are references in the use, and migration, of microservices, taking a lot of advantages from the architecture to their systems (Rud, 2023). However, not all systems are eligible for using the microservice architecture. Amazon Prime Video team is an example that decided to migrate back to a monolithic architecture. The Video Quality Analysis team noticed that running the infrastructure at a high scale was expensive and presented scaling bottlenecks. As a result, the architectural team chose to pack all components into a single process and migrate from a microservice-based architecture to a monolithic, reducing over 90% of the infrastructure cost (Kolny, 2023).

Although some works in the literature have addressed microservice migration, most of them focus on specific domains or methodologies to select the possible candidates for microservice from a monolithic architecture or even report the experience resulting from an architecture migrated to microservices. Therefore, there is a lack of common ground for defining general steps in migrating an application to a microservice-based architecture. A high-level process (i.e., a metaprocess) could assist organizations and software developers in executing efficient migrations, enhancing activities throughout the migration process, and enabling them to define their migration approaches, focusing on monitoring, implementation, and deployment steps.

The migration process from a monolithic architecture to microservice architecture can be carried out in two ways: (i) through a rebuilding process, in which another application is created with the required changes, and (ii) through an architecture modernization process, performing a refactoring in the application (Doležal et al., 2022). Using outdated programming languages and databases and design flaws can make it impossible to refactor the application, leading to the creation of a new application.

This work aims to fill the gap of a general process to guide the migration to the

microservice architecture. It introduces the ongoing efforts to define the Metaprocess for Microservice Migration Kernel (M3K) to support migration between a monolithic application to microservice-based architecture. M3K intends to serve as a foundation for building a general-purpose process that development teams and organizations can follow to perform their microservices migration processes and track their migration results.

M3K relies on the standard proposed by OMG Essence (Object Management Group, 2018), which offers a collection of theoretically grounded shared components and a straightforward visual language, facilitating teams in formulating, exchanging, and reusing software engineering methods and practices. M3K builds upon and expands the elements defined in Essence to establish methodologies for tracking the migration to microservice architectures. These practices can be combined into a more comprehensive approach, as a metaprocess that supports the microservice migration process.

1.1 Research Problem

Modernizing an architecture is a challenging task, especially considering that they are often carried out without experience in software migration, performed in an ad-hoc way, and with systems that acquire many dependencies (Ponce et al., 2019). Jacobson et al. (2019) defines two challenges in migrating monolithic applications to a microservices-based approach: (i) finding an effective way to decompose microservices, which is one of the main approaches in the literature; and (ii) finding effective ways to change microservices and update them once they are already deployed.

When analyzing the migration process from monolithic architectures to microservices, each work chooses to determine specific activities for the context of the application, which are defined as follows:

1. Some process defines which functionalities of the monolithic application are strong candidates to become microservices, based on static code analysis, dynamic analysis,

or in an ad-hoc way.

2. Some works (Di Francesco et al., 2018; Ponce et al., 2019) define a migration process with a low level of detail, not being possible to replicate it. These works do not detail an explicit process. They present concepts and not a sequence of steps to be followed.
3. Some works (Bucchiarone et al., 2020) focus only on showing the results obtained by the migration, not specifying the processes adopted to obtain the final product.
4. Some works (Auer et al., 2021; Taibi et al., 2017) define the migration in an ad-hoc way, which does not present a reference behind the developed process.
5. Some works (Abgaz et al., 2023; Michael Ayas et al., 2023) present different activities to make the migration based on the literature but do not consider aspects such as stakeholders and competencies.

1.2 Research Questions

The following research questions (RQs) are associated with the goals of this work:

RQ1: *What activities are present in a migration process from monolithic applications to microservice-based architecture?*

Rationale: Understand the activities done in migration processes to create a metaprocess that can be used to guide the migration to the microservice architecture.

RQ2: *How to define a migration metaprocess from monolithic architectures to microservice-based architectures using the Essence standard?*

Rationale: Propose the steps necessary for a metaprocess for migrating from a monolithic architecture to a microservice-based architecture applying the Essence standard concepts.

RQ3: *How to validate the proposed migration metaprocess?*

Rationale: Propose a process, instantiated from the metaprocess, defining specific activities from the activity spaces provided by M3K to each of the three areas of concern, and applying it in a proof-of-concept in a legacy application.

1.3 Goals

The general goal of this work is to propose a Migration Metaprocess from a Monolithic Architecture to a Microservice-based following the Essence standard. This metaprocess provides a supporting approach for the migration process, which companies can rely on to migrate their architectures using a well-defined generic process. The following specific goals support the general goal:

G1: Identify the processes defined in the literature for the migration from monolithic architectures to microservices;

G2: Identify the frequent activities in each migration process;

G3: Understand the definition of the Essence standard and its components;

G4: Understand the application of the Essence pattern in the elaboration of processes;

G5: Model the metaprocess extending the Essence concepts and its elements;

G6: Evaluate the metaprocess by deriving a process and applying it in a proof-of-concept

G7: Survey developers that work in migration from legacy applications to microservices architecture.

1.4 Methodology

To address the abovementioned issues related to migration to microservice architectures, this work presents the *Metaprocess for Microservice Migration Kernel* (M3K). It intends to help companies and developers create migration processes from monolithic architectures to microservices, considering steps from the perspective of authors directly affected by the migration, means of work, and development, as well as the elaboration of the deployment process. Figure 1.1 presents the methodology adopted for defining the metaprocess.

In this work, we present the process of defining a metaprocess to serve as the basis for a migration process. The migration between applications is performed in the context of software engineering and software development, in which we propose transforming a monolithic architecture into a microservices architecture.

The elaboration of the metaprocess starts with an analysis of the literature of which activities are essential in a migration process and understanding Essence Standard. For the elaboration of the Metaprocess for Microservice Migration Kernel (M3K), a literature exploratory search was carried out on the processes of migration from monolithic architectures to microservice-based, as well as on the development of Essence's application and microservice architecture.

From the Exploratory Research in the Literature step about the activities of the migration process and Essence standard, it is possible to define a metaprocess, considering all the elements of the standard (alphas, activity spaces, and competencies). Each element is designed to focus from a development process to a migration process.

After the definition of the metaprocess, the Evaluation step aims to define a process, derived from the metaprocess, and apply it to a legacy application, as a proof-of-concept. In addition, survey developers who have experience in the migration process.

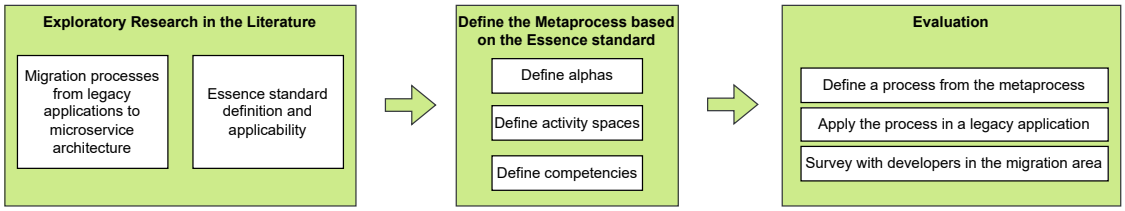

Figure 1.1: Methodology for the elaboration of the Metaprocess

Figure 1.2 presents an overview of the elaboration of the metaprocess. From the definition of the problem, the goals are defined (Section 1.3), focusing on being able to define a microservice architecture from a monolithic application that can be applied in any context of the application. Each goal focuses on answering the proposed research questions (Section 1.2).

In the end, with the answers to the research questions and the sequence of steps of (i) Exploratory Research in the Literature, (ii) Defining the Metaprocess based on Essence standard, and (iii) Evaluation, the metaprocess can be used to help company and developers to migrate their applications using a theoretically grounded proposal migration process.

1.5 Outline

The remaining chapters are organized as follows. Chapter 2 presents the conceptual basis of this work, such as microservice architecture, Essence standard, and metaprocess and process definition. Chapter 3 presents M3K, explaining each concept, defining Alphas, Competencies, and Activity Spaces, and the use of the metaprocess, presenting the practices. Chapter 4 discusses related work, describing works that present activities for a migration process from legacy systems to a microservice-based architecture. Chapter 5 instantiates M3K and presents a survey with developers. Finally, Chapter 6 presents the final remarks of the work.

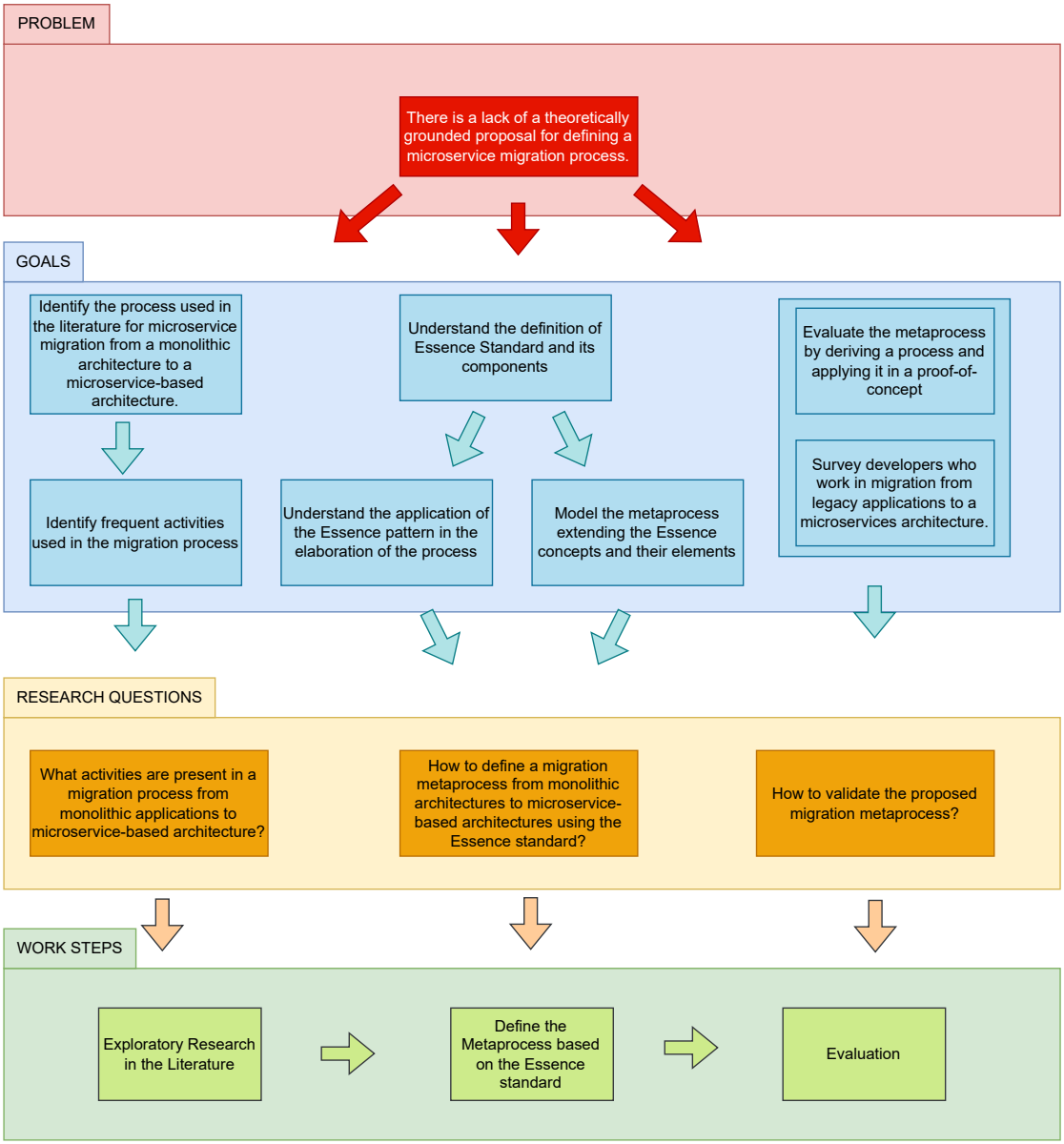

Figure 1.2: Steps for the model of the Metaprocess

2 Fundamentals

This chapter presents some basic concepts for this work. Section 2.1 presents the microservices architecture, from features to comparisons with the monolithic architecture. Section 2.2 presents the OMG Essence standard, its elements, and its characteristics. Section 2.4 presents the theoretical foundation of metaprocesses and processes.

2.1 Microservice architecture

The idealization of microservices comes from the Service-oriented Architecture (SOA), defined by Newman (2015), as a design approach in which several services collaborate to plan a set of capabilities, focusing on reusability for maintenance and code rewrites. Among the main challenges of SOA are the communication protocols (e.g., SOAP), with the heterogeneity of systems, presenting different protocols; vendors middleware, in which companies insert more functionalities than necessary to be used in their packages; and lack of definition of the granularity of services, ranging sizes from small to large services, affecting performance and transaction management. Microservices were designed to think about solutions to these challenges.

Microservices are deployment-independent units, having components that support interoperability through a lightweight communication protocol, such as REpresentational State Transfer (REST) calls, which is a communication mechanism that uses the HTTP protocol; and messaging. (Bucchiarone et al., 2020). Fachat (2019) defines microservices as

small parts broken from a system so that they are implemented and executed individually, ensuring greater time-to-market for new functionalities.

Figure 2.1 presents an example of microservice architecture. There is a representation of two users, one using a Web application and another a Mobile application, and four microservices, each one with its database. Both users make requisitions to the API Gateway, which is responsible for forwarding each request to a specific microservice. When the request receives a call, the operations will be executed, using the database or not.

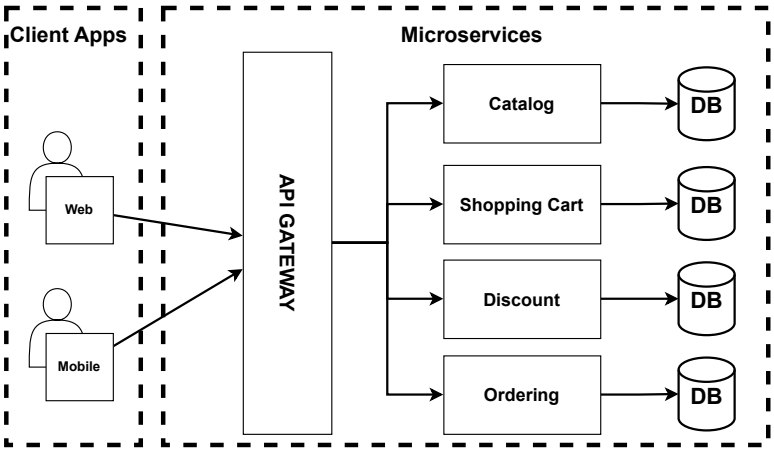

Figure 2.1: Microservice Architecture Abstraction. Font: Ozkaya, 2023

SOA and microservices have similar advantages, such as dynamism, modularity, distributed development, and integration of heterogeneous systems (Bucchiarone et al., 2020). Microservices and SOA rely on service-based architectures, but there is a variety in terms of service characteristics. Richards (2015) presents some main differences, such as service classification, service ownership and coordination, service granularity, and component sharing, presented in table 2.1.

Table 2.1: Different characteristics between SOA and Microservice. Font: Richards (2015)

Characteristics	**SOA**	**Microservice Architecture**
Classification of Services	SOA's service classification is divided over four types: business service, enterprise services, application services, and infrastructure services	Microservice architectures are divided into two types of services: functional services and infrastructure services
Ownership and Coordination	Coordinate with multiple groups , resulting in more time and effort in development, testing, deploying and maintaining.	Development teams own the infrastructure and functional services, resulting in less effort and time in development, testing, deploying and maintaining service process.
Service Granularity	Range in size, from small application services to large enterprise services, encompassing much more business functionality.	Generally small, with single-purpose services that do only one thing, not relying upon multiple services to complete a single business request.
Component Sharing	Share-as-much-as-possible architecture style	Share-as-little-as-possible architecture styles

2.1.1 Challenges

Microservice architectures present challenges, such as data integrity and consistency between services, as well as the specification of system interfaces, version compatibility, and application and infrastructure security (Bucchiarone et al., 2020). Among these challenges, we can highlight the design challenges, such as lack of fault tolerance, self-healing, variability of characteristics, and the granularity of microservices. The more microservices, the greater the network communication costs and the complexity

of their distribution.

Another challenge in the development of microservices refers to (i) coordinating the implementation of communication between microservices, (ii) coordinating tasks performed by each team so that a new version can be developed to meet new requirements, while an old version can be maintained to meet other customers, and (iii) carrying out tests to validate the entire flow of actions in the system.

The operation of microservices is another challenge, especially considering that more services, execution environments, and API call invocations are needed, resulting in increased consumption of computational and network resources. The operational complexity of microservices can be circumvented through the flexibility of the environment, increasing and decreasing the necessary resources, or migrating from one environment to another, resulting in the challenge of locating and coordinating systems with large services.

2.1.2 Benefits

Microservices should be designed to be small to enable rapid development of new versions that can coexist with, or even replace, business logic (Bucchiarone et al., 2020). When a version of a microservice becomes obsolete for a company, it is enough to implement a new version of the microservice and modify only it. It is not necessary to stop the entire system and make it unavailable to customers. This feature avoids losses of financial value for companies.

Microservices offer a range of advantages, making them an appealing architectural choice. First and foremost, microservices provide *dynamism* and *scalability*, allowing new instances of the same service to be launched to distribute the system's load effectively. Their *modularity* and *reusability* enable easy adaptation to suit various application needs (Bucchiarone et al., 2020; Newman, 2015).

Additionally, microservices contribute to the system's *resiliency*, as they can isolate component failures, ensuring uninterrupted operation of the rest of the system. Furthermore, they enable *distributed development*, permitting different teams to work on services concurrently, with well-defined communication contracts through interfaces. This fosters the implementation of new functionalities or microservices without disrupting the entire system (Bucchiarone et al., 2020; Martínez Saucedo et al., 2025; Newman, 2015).

Lastly, microservices facilitate the *integration of heterogeneous and legacy systems*, as they allow for the use of diverse technologies within each microservice, connected via standardized communication protocols. Though this flexibility can have some drawbacks, such as the need for qualified professionals to manage maintenance and changes, microservices prove more adaptable compared to monolithic applications when it comes to altering programming languages, databases, or frameworks (Bucchiarone et al., 2020; Newman, 2015). Overall, microservices present a compelling solution for building flexible, scalable, and resilient software systems.

2.2 Essence

Essence is an OMG standard that provides a foundation with elements, a language, and a framework that enables teams to define, share, and reuse Software Engineering methods and practices (Object Management Group, 2018). The origins of Essence date to 2009, lying on the foundation of the Software Engineering Methods and Theory (SEMAT) initiative. Upon the observation of the lack of a solid theory, proven principles, and mature practices in Software Engineering, such an initiative aimed at establishing a set of essential, widely-agreed elements supporting software development efforts along with a simple language for describing methods and practices (Jacobson et al., 2012).

Despite being relatively recent, Essence has already demonstrated its usefulness

in industrial settings with both small and large teams, mainly focusing on software practitioners' work (Jacobson et al., 2019; Jacobson et al., 2013). In comparison to other well-known standardized specifications and notations such as the Business Process Model and Notation Business Process Model and Notation (BPMN) and the Software & Systems Process Engineering Metamodel Software & Systems Process Engineering Metamodel (SPEM), Essence is theoretically founded and provides better support for defining and enacting processes (Fonseca et al., 2021).

The primary element of the Essence standard is the *kernel*, which establishes a common ground for carrying out Software Engineering endeavors and assembling methods. In addition, a simple visual language supports describing practices that can be used both to represent the kernel and describe practices and methods in terms of the kernel elements. These features allow defining generic and domain-purpose practices upon a shared vocabulary that teams can share, adapt, and compose into larger methods, besides tailoring them for their respective organizations and applications.

Essence defines four types of elements: *Alpha*, *Work Product*, *Activities*, and *Competency*, which are explained in this section.

2.2.1 Alphas

Alpha is aspects of the software we want to understand, monitor, direct, and control, denoting important elements in the evolution process. Each alpha presents states to describe progression through a lifecycle of the development process.

Figure 2.2 presents the Essence Kernel to development endeavors. *Customer*'s concern area is related to those that operate or will have some benefits from the migration, as *stakeholders* that use the system and some *opportunity* that arose during the operation of the system. *Solution*'s area is related to what will be delivered, as *requirements* of the system or even a *software system* that will be developed. *Endeavor*'s area is the effort to

be made to reach the desired goal, in the case of software development, the initial *work* and *teams* to make the necessary effort in *ways of working*.

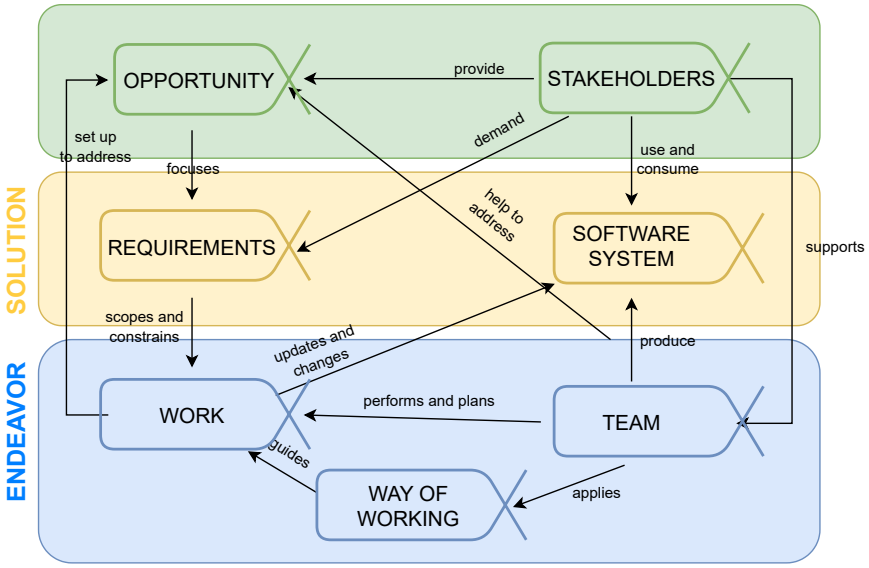

Figure 2.2: Essence Alphas

For each concern area, there are alphas related to it. In *Customer* area, the *Opportunity* alpha presents an idea that can be successful or a failure, these alphas are dedicated to understanding the needs of the users and how the building of this final product can change the environment for which it was designed. *Stakeholders* alpha is another *Customer* area. This alpha represents those that are directly related to the construction of the idea and involved in the decision of the project.

Solution concern area presents *Requirements* and *Software System* alphas, those related to the product that will be developed to deliver value to the stakeholder. *Requirements* alpha presents the stakeholders' view of what is expected from the software system, which will serve as a guide for features to be developed. While the *Software System* alpha represents the actual application to be developed, the product being developed. For Essence, there are 3 important characteristics of a *Software System*: *functionality*, *quality*, and *extensibility*. The *functionality* is directly related to the functionalities reported in

requirements; the *quality* with how good the application was projected, if it does not present errors or crashes; and *extensibility*, the system can change in functionality or environments but it will work as desired.

In *Endeavor* concern area, *Essence* defines three alphas to achieve the goal of the opportunity: *Team*, *Work* and *Way of Working*. The *Team* alpha is related to the workers that create the software system, composed of competencies and skills necessary to execute the development process. *Work* is related directly to the Team Alpha, which is responsible for preparing, coordinating, tracking, and completing the system, resulting in a sequence of activities. *Way of Working* defines how the work should be done, planning and defining practices and tools, which requires that every worker project this work.

Each alpha results in *Work Products* that will be developed in a certain way. By itself, an alpha can not be tangible, that is why Essence defines Alpha States. The Alpha States is a progression of the lifecycle of an alpha, helping developers to face the risks in development, understanding, and dealing with them (Ng et al., 2013). These states are defined sequentially so that a state can only be started after its previous state has finished. The completion of a state is performed based on a checklist of tasks for the state being analyzed.

Figure 2.3 presents the requirements' alpha states defined by Essence.

The *Requirements states* are divided into six activities, which are organized sequentially, given that the Bounded state can only be started when the Conceived state has been finished, that is, the checklist corresponding to the state has been completed. Requirements alpha starts in the Conceived state, focusing on understanding the need for a new system, and is defined as a checklist that (i) all stakeholders agree to create a system, (ii) a definition of the users that will use the system, (iii) the stakeholders that will fund the system is defined and (iv) there is a clear opportunity in developing the system. All the other alpha states and their respective checklists can be found on Essence

documentation.

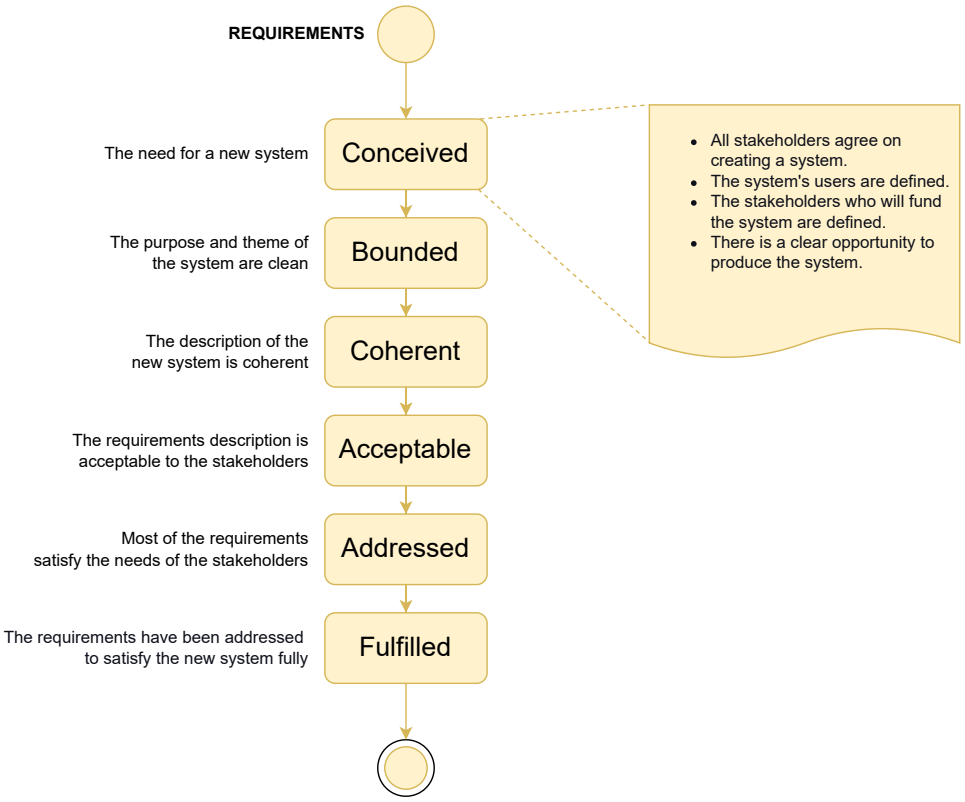

Figure 2.3: Alpha States from Requirements Alpha

2.2.2 Work Product

Work Product can be defined as tangible things that provide evidence to verify the achievement of alpha states. In Essence, *Work Product* alpha is not only defined as a final product, but it requires all the ways of working to generate this product, all the requirements, and effort to deliver a software system with quality. This alpha can require different levels of details, which will vary from different points of view, such as programmers' level of knowledge, customer requirements, regulatory requirements, and organizational policies. Figure 2.4 presents some examples of work products from practices, with the *Code* as the programming practice.

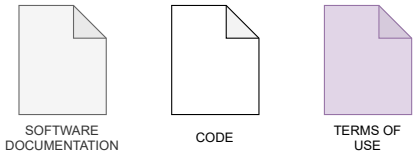

Figure 2.4: Work Product Examples

2.2.3 Activity Spaces

Activities are actions that practitioners do, bound to a specific practice. In Essence, it is not defined that any activities are to be made, which depends on a lot of variables, such as stakeholder needs, team understanding of the application, and so on. Essence defines *Activity Spaces*, these spaces are a set of generic labels that will be customized depending on the software that will be developed. These spaces are responsible for guiding the team during the development process divided by Essence's areas of concern.

In the development of software applications, the Essence defines Activity Spaces in each area of concern. In the *Customer* area, it is defined the following activities spaces: *Explore Possibilities*, *Understand Stakeholders Needs*, *Ensure Stakeholders Satisfaction* and *Use the System*. *Explore Possibilities* defines the activities related to the understanding of the opportunity, identification of the stakeholders, and the goals to achieve with the software system. The *Understand Stakeholders Needs* is the activity space to define activities to understand stakeholders' needs, identifying and working on requirements from stakeholders. *Ensure Stakeholder Satisfaction* is another activity space related to the stakeholder's needs, sharing results and progress of the development of the stakeholders' application. In the last activity space, *Use the System*, validate the use of the application by the stakeholder in a real environment, checking how the application helps the stakeholders.

Solution area defines six activities spaces: *Understand the Requirements*, *Shape*

the System, *Implement the System*, *Test the System*, *Deploy the System* and *Operate the System*. The *Understand the Requirements* activity space is related to the goal of defining the functionalities that the system will have to do. In *Shape the System* activity space structures the system, modeling and architecting the system that will be produced. *Implement the System* build the system, including all the steps from implementation, test, integration, bug fix, and unit tests. *Test the System* takes the stakeholder's requirements and validates if all of them were satisfied. *Deploy the System* takes the tested system and deploys it in a production environment to be used by the stakeholders. And *Operate the System* supports the use of this application in the production environment, checking how the system is operating.

Endeavor area defines five activity spaces to form the team and track the work done: *Prepare to do the Work*, *Coordinate Activity*, *Support the Team*, *Track Progress* and *Stop the Work*. *Prepare to Do the Work* activity is related to the necessary pre-organization to carry out tasks and work. *Coordinate Activity* plan and re-plan every task in the work, coordinating the team's work. *Support the Team* activity is related to helping the team in every challenge, collaborating, and improving their ways of working. *Track Progress* is responsible to track and assess the progress of the team. And *Stop the Work* activity is responsible for stopping the development and handover of the team's responsibilities.

2.2.4 Competency

Competency are the abilities needed when applying a practice. Each practice requires competencies from their executors, when considering developing a software system, every programmer must understand how to execute their tasks. In some cases, when the team does not have the right understanding of how to do some work, a specialist can help teach how to achieve the desired goal. In figure 2.6, Essence presents the common

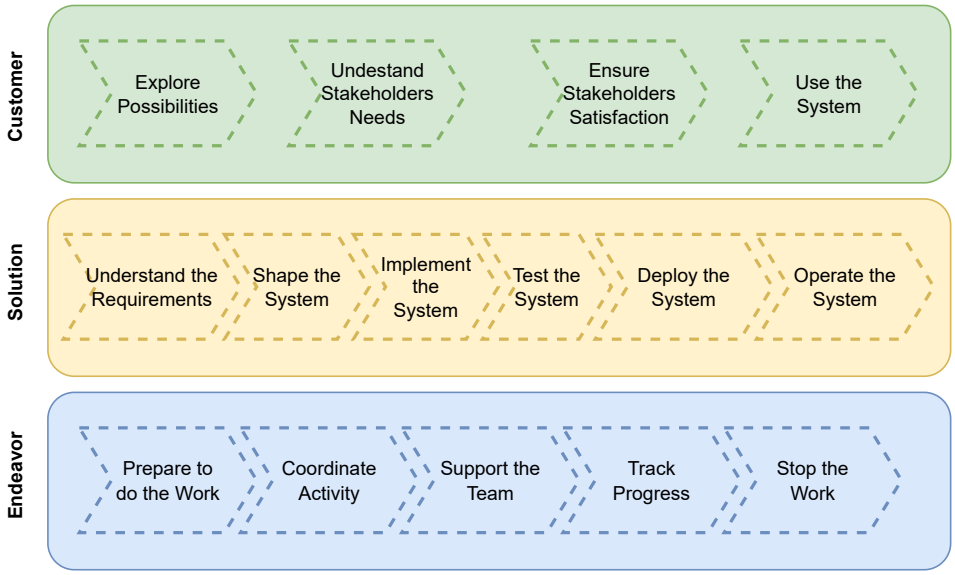

Figure 2.5: Essence Activity Spaces

competencies in every development of a software system.

In the *Customer* area, the team must understand the context of the application to be developed, with their logical business and technical aspects. The *Stakeholder Representation* competency is related exactly to the understanding of the team about *Stakeholders* needs, their point of view and the ability to communicate and gather all the requirements needed.

In *Solution* area, Essence defines more three competencies: *Analysis*, *Development*, and *Testing*, to capture and analyze the requirements and build and operate the system. The *Analysis* competency is related to the understanding of the requirements, as well as the opportunities in the software that will be developed. In *Development* competency, it defines the ability to design, program, and code the software system, delivering software with quality, standards, and norms defined by the team. And *Testing* competency to be able to test the system, ensuring usability and satisfying the requirements.

The last area of concern is the *Endeavor* area, taking all competencies related to the ability to organize development and manage their workload. *Essence* defines two

competencies in the *Endeavor* area: *Leadership* and *Management*. *Leadership* competency is related to the capacity to inspire and motivate the team, making them achieve the goals proposed by the development of the system and delivering the system with quality to the stakeholders. And *Management* competency is related to the ability to coordinate, plan, and track the work done by each team.

Figure 2.6: Essence Competencies

2.2.5 Practices

Essence defines Practices as a precise and active form of language for describing software engineering activities in designing a system. The creators of Essence define the practices that help produce high-quality code. Essence uses shapes and icons to define such actions, such as the use of alphas, activities spaces, and the necessary skills

for those practices, in addition to the product with which you want to create. Each element presents a connection with other elements so that the execution of this practice guarantees the final product with which it was designed.

In figure 2.7 a practice example proposed by Jacobson et al., 2019 is presented. From a *Write Code* activity, the goal is to produce a *Work Product*, a *Code*, resulting in the *Software System*, that are described by the *Code*. For the activity two competencies: Development and Testing, will receive as input the *Requirements* progressing with the build of the *Software System*.

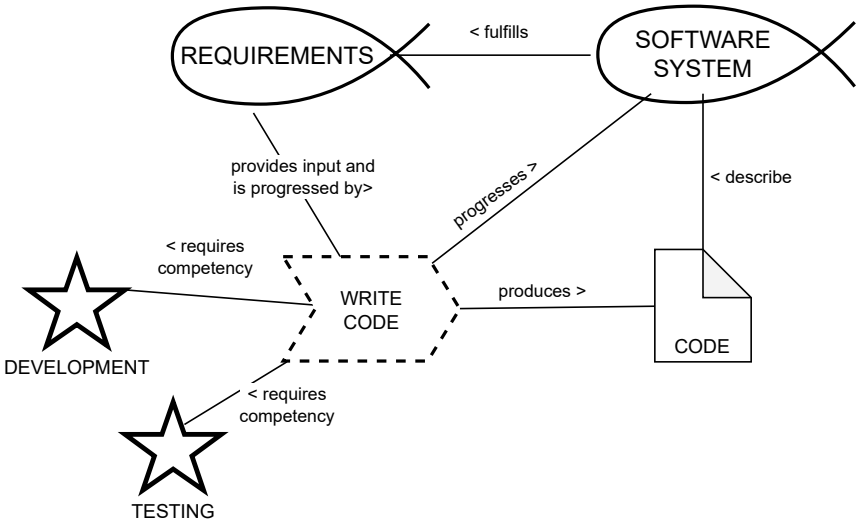

Figure 2.7: Essence Practice Example - Source: Jacobson et al., 2019

2.3 Essence Choice

The choice to use Essence over other software specifications was a design decision, in which we listed Essence's advantages over BPMN and SPEM. BPMN is an Object Management Group (OMG) specification released in 2014, to focus on application users, creating drafts and initial processes for those who implement the processes to be applied (Sahay et al., 2023). SPEM is a specification released by OMG in 2008, focusing on

compliance points to ensure that the objectives of the applications being implemented are achieved ("Systems Process Engineering Meta-Model Specification", 2008).

Fonseca et al., 2021 presents a systematic review comparing the three specifications (SPEM, BPMN and Essence), looking for characteristics, advantages and disadvantages, research trends, and the list of tools that support the implementations, from 40 selected articles. The author details that BPMN supports the native execution capability of the process, but not all tools perform the implementation correctly; SPEM does not offer native support for these implementations, requiring extension applications; while Essence has different types of execution and native support, being the base of the system. As a final result, the authors do not detail the preference of one specification over another, because each specification seeks to focus on a specific purpose.

Elvesæter et al., 2013 compares Essence with SPEM and concludes that although the specifications have some similarities, through basic concepts of each language, Essence contains some different concepts and additional properties that allow support for enactment, such as the ALPHAs, in addition to the possibility of monitoring, tracking and associated work to be done in each alpha, such as alpha states.

2.4 Metaprocess and process

Metaprocess is defined as a set of human interventions and tools that, together in the software process, aim to define a set of rules, strategies, and policies to be defined by an organization. Meanwhile, the software process is a sequence of activities to be performed to produce a software system, satisfying specifications that are used to generate a metaprocess (Nguyen et al., 1994).

The generation of a metaprocess is carried out by the need to improve the quality of the product development and delivery processes, in addition to reducing production costs. After generation, the metaprocess must go through an evaluation

stage, so that a professional in the area will analyze from the outside world: (i) defining new performance policies and organizational procedures; (ii) from the organization's point of view, modifying the organization's infrastructure, improving processes and rules in the environment in which it works; (iii) in the project environment, through feedback from work units that are suffering from problems, such as lack of performance, quality, reallocation of employees or modifications of initial requirements; and finally, (iv) individuals, analyzing needs through a design process that does not require high levels of complexity.

Some characteristics of the metaprocess are defined from the formalism, by the way, it was described or represented, not having any formal relationship with any other component; of the method, describing the methods and models that are used for the evaluation of the product and the quality process; and support tools, which denote mechanisms and services to modify the functioning of the process.

3 The Metaprocess for Microservice Migration Kernel (M3K)

The Metaprocess for Microservice Migration Metaprocess Kernel (M3K) is a modeling metaprocess for the migration of applications with a monolithic architecture to a microservices-based architecture, using as a basis the software development standard developed by OMG, the Essence.

The initial version of M3K was presented at the XXXVII Brazilian Symposium on Software Engineering (Medeiros et al., 2023), which was an opportunity to show how necessary is a definition of a metaprocess to auxiliary enterprises and developers in the migration process from legacy applications to a microservice-based architecture.

This chapter is organized as follows: section 3.1 presents the alphas for preparing the migration process; section 3.2 shows the activity spaces defined for the migration preparation; and section 3.3 presents the skills needed to prepare the process.

3.1 Alphas

The alphas proposed for M3K are divided into the same concern areas as Essence: *Customer*, *Solution*, and *Endeavor*. The *Customer area* focuses on system users, that is, how the migration will help stakeholders to improve the currently existing solution through existing motivations. The *Solution area* focuses on performing the migration

through modeling and refactoring of the existing architecture and obtaining the final result, the created microservices. And finally, the *Endeavor area*, seeks to structure the work teams both from the point of view of refactoring the modeling and the source code, in addition to how the work will be structured to obtain a Work Product as a final result. In M3K, seven alphas are defined and divided into 3 concern areas, presented in figure 3.1.

In the *Customer* area, two alphas are defined: *Stakeholders* and *Objectives*. The *Stakeholders* alpha seeks to specify and detail the application's stakeholders, that is, from end users to companies, all those who in some way benefit from the migration process of the desired application. This alpha is directly connected with Essence's *Stakeholders* alpha since everyone must be aware that in the development or migration of an application, all the roles of all agents in the processes must be clear. The *Objectives* alpha should not only detail but demonstrate that all actors in the process understood what they wanted to achieve with the migration process. This alpha is related to Essence's *Opportunity* alpha, but the objective proposed in the elaboration is the definition of a metaprocess for migrating an application with monolithic architecture to an architecture based on microservices, which differs from Essence, in which opportunities are related to the advantages of developing a new application based on an idea.

In the *Solution* area, two alphas are defined. The *Architectural Changes* alpha is defined for carrying out the modeling of the restructuring of architectural changes to adapt the old architecture based on microservices. The alpha can be mapped to the alpha of *Requirements* in Essence, so that in M3K the focus is on changes to the current architecture and not the inclusion of new requirements imposed by *Stakeholders*. Finally, the definition of the generation of our final product, which are the microservices that will be generated and implemented in alpha *Services*. As with Essence's *Software System* alpha, the main objective is to generate a final product, so that in M3K a set of

microservices is generated, maintaining the behavior of the system.

When we analyzed the *Endeavor area*, we defined the *Architectural Team*'s alpha, defining the teams and proposing work strategies for the elaboration of actions that could favor those responsible when structuring the architecture and be able to help in defining the development teams and how they environments of each service must be structured, from components to communication models between microservices. The *Development Team* alpha defines the teams responsible for implementing the microservices so that with the help of the architectural team it is possible to implement this microservice for testing stages and delivery of the final product to stakeholders. The *Architectural Team* and *Development Team* alphas are directly related to the *Team* and *Work* simultaneously Essence alphas, mainly because each team has specific work to be performed to ensure that the migration goes smoothly. Finally, the *Services Environment* alpha is defined, in which, in addition to defining the environments in which each microservice will be operated, it defines how the work will be carried out, considering the actual communication between the microservices and how they should be operated on. This last alpha is directly related to Essence's *Ways of Working*, in addition to defining the necessary work for migration, it guarantees the execution of microservices.

Alpha states are central elements in the elaboration of any alpha, especially when we consider that they are the representation of each one of them, representing the progress made within that alpha. M3K defines a set of alpha states so that it is possible to monitor the progress of each alpha, to target the migration from a monolithic architecture to one based on microservices.

Figure 3.2 defines the alpha states of the customer area. Figure 3.2(a) presents the alpha states for Stakeholders alpha, which is defined in a sequence of states: Use, Find, Solution, Track, and Solve. Each state is defined as follows:

1. *Use*: the first step before any decision, the main goal of this stage is to analyze

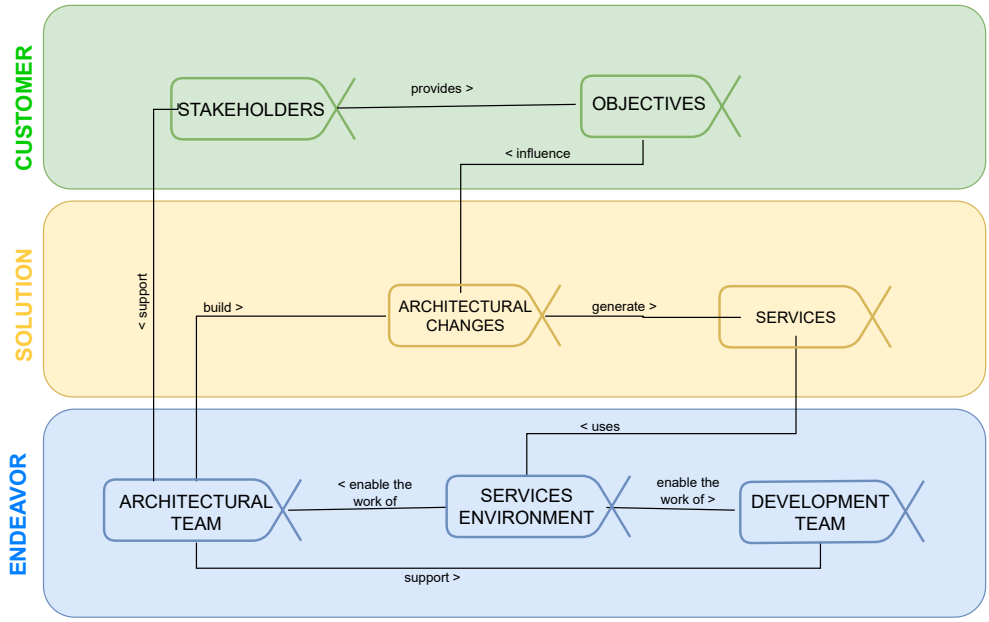

Figure 3.1: M3K Alphas

how the original application works. This step is necessary to make the stakeholders understand how the application works. When considering large companies, most of them work with many applications and each one is responsible for making a specific work, that makes the stakeholders only understand the basis of that application.

2. *Find*: During the use of the application, stakeholders could analyze what are the points that need to change.
3. *Solution*: After analyzing and deciding which points of the application need to change, it's necessary to decide which way should be taken if it is a logical problem or a structural problem in the architecture. Both of the solutions are linked with other alphas, that belong with changes in the architecture and talk with developers and software architectures that can help with the problem.
4. *Track*: All the migration process needs the participation of the stakeholder, being responsible for helping with any question about the system operation and necessary changes in the logical business; and

5. *Solve*: This last stage is responsible for making sure that the migration is concluded and everything planned for the Stakeholder alpha is finished. During the migration process, the stakeholder has a main role, he understands the needs of the application and how these changes will benefit the operation of the company.

Figure 3.2(b) presents Objectives alpha, composed of Operation, Work, Study, and Needs. Each alpha is explained as follows:

1. *Operation*: Before understanding how the application works, it is necessary to understand what is the job of the company, what their work is, and how the application benefits the company. This step is important because provides a basis for knowledge about what's expected in the application;
2. *Work*: The domain of the application provides a full understanding of what the application should do and how provides changes for the company their functionality;
3. *Study*: Different from the Domain stage, the application stage provides the use of the application. The main stage when we are considering making a migration is the application stage, it's from here that is possible to understand how the application works which possible services can be taken from the system; and
4. *Needs*: The Needs stage takes on paper which changes it's necessary to migrate to microservice architecture, what is important to consider during the migration, and each other's characteristics that the team finds important.

In figure 3.3(a), the states related to the alpha of Architectural Changes are presented. The states are defined from the foundation that all of them must be elaborated with a focus on the solution of the migration that is being carried out. The following sequence of states is defined:

1. *Agreement*: To carry out migration activities from a monolithic architecture to one based on microservices, all migration actors must have the same agreement on the objectives for which the migration is to be carried out and the expected results.

Figure 3.2: Alpha States from *Customer* area of concern. (a) *Stakeholders* alpha. (b) *Objectives* alpha.

2. *Solution*: In this state, it is necessary to dedicate checklists to verify whether the use of microservices proposes to solve the problems that were pointed out for the need for migration.
3. *Creation*: The Creation state is dedicated to creating the new architecture, and defining its components and microservices.
4. *Projection*: After creation, it is necessary to validate that the new architecture corresponds to a microservices architecture, based on the characteristics of the architecture.
5. *Development*: In this state, the entire process with the objective of architectural modeling is finalized and then passed on to the development team to act on the changes.

6. *Migration*: The last alpha step corresponds to ensuring that the migration was successfully performed at the architectural point.

In figure 3.3(b), the alpha states of Services are defined. In this alpha state, we define all states related to the generation/implementation of microservices. The following states are defined:

1. *Definition*: The Definition state guarantees that the microservice to be developed is well defined, with all its components and environments ready to be used.
2. *Implement*: In this state, the implementation stage is carried out at the microservice code level, carrying out refactorings in the business logic of the original application.
3. *Operating*: This state is dedicated to guaranteeing the operability of the system, either through visual use or calls to the microservice endpoints.
4. *Test*: The test state ensures that the application, in addition to working, is correct, performing tests of expected values and results.
5. *Ready*: In this state, the application is completely ready to be used, so it must already be in an environment available for use.
6. *Use*: Finally, the usability step is defined so that it is possible to verify how end users are using the application.

The Endeavor stage runs through all the activities that are developed to carry out the architecture migration, being defined from 3 alphas, which have already been analyzed in the previous chapters and soon the states of each one of them will be presented.

In figure 3.4 (a), the states corresponding to Architectural Team alpha are defined. In this alpha, the following states are presented with their respective descriptions:

1. *Study*: In this state, the teams that will participate in the migration between architectures are defined.
2. *Goals*: The Goals state intends to confirm that all members of the architectural team can understand the objectives that are proposed with the migration, as well as the

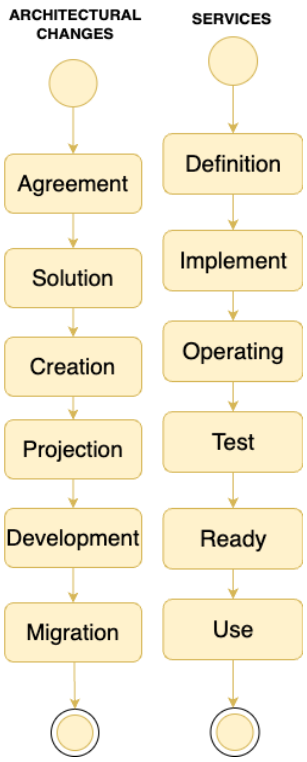

Figure 3.3: Alpha States from *Solution* area of concern. (a) *Architectural Changes* alpha. (b) *Services* alpha.

possible problems and solutions that may arise during the execution of the process.

3. *Tasks*: Tasks aim to define the tasks that will be performed in the migration, punctuating their assignment to the teams.
4. *Changes*: In this state, the necessary architectural changes are made for the continuation of the migration.
5. *Manager*: The Manager status proposes to carry out the coordination and monitoring of migration tasks throughout the evolution process of the software.
6. *Delivery*: Finally, a new architecture delivery step is defined.

Figure 3.4 (b) defines the states corresponding to Services Environment alpha. Each of the states is defined below:

1. *Define*: In this state, the environments in which each microservice will need are

defined, note that despite the technology currently working around containers and cloud computing, here we define in addition to instances, base images of the system, among other details that may be necessary during the development of the new architecture.

2. *Available*: This state is defined to guarantee the availability of the environment, that is, that the developers have the environment available for them to implement the solution.
3. *Components*: In this state, the necessary components for the deployment of the services are defined, for example, gateways or load balancers, any component that the architectural team analyzed as necessary for the execution together with the microservice.
4. *Deploy*: The Deploy state is defined to deploy the microservices, making them available for consumption by the Stakeholders.
5. *Track*: Finally, the state of Track, in which the execution of the environment in which the microservices will be used is monitored.

Figure 3.4 (c) presents the representation of the states of the *Development Team* alpha, which are detailed below:

1. *Understand*: In this state, all participants of a given microservice to be created must understand their role and what the need for that microservice for the migration, so that they have clear goals of what they want to achieve.
2. *Division*: Division is the state responsible for defining what actions each participant in the microservice will have in its elaboration, whether in business logic refactorings, communication protocols, or tests, among others that depend on the microservice's needs.
3. *Implement*: In this state, the actual implementation at the code level is defined, in which the developers will be responsible for creating a microservice that meets the

need indicated in the Understand state.

4. *Test*: The Test state defines the tasks needed to ensure that in addition to the application working, it is returning the correct results. Here you can apply unit testing techniques and functional code refactoring.
5. *Delivery*: The delivery state aims to deliver the implementation of the new microservice so that it can be used by stakeholders.

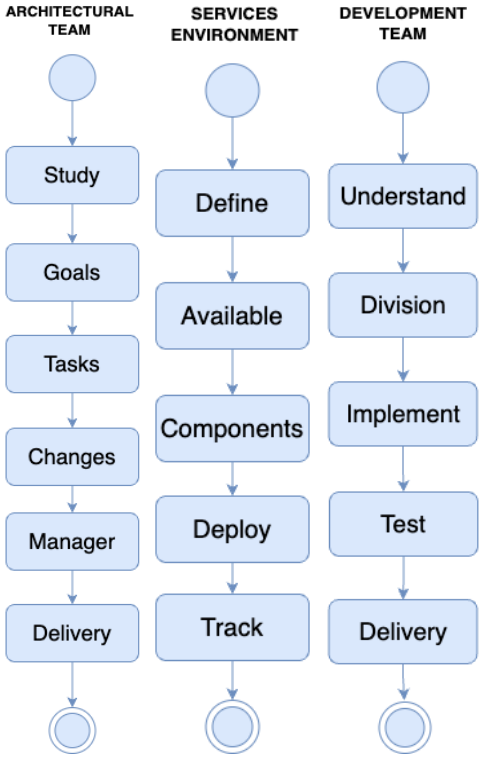

Figure 3.4: Alpha States from *Endeavor* area of concern. (a) *Architectural Team* alpha. (b) *Services Environment* alpha. (c) *Development Team* alpha.

3.2 Activity Spaces

The goal of defining a metaprocess is to make it possible for developers and companies to use processes that are generic enough to define their migration processes.

Therefore, just as Essence does not define activities, M3K does not establish static activities for carrying out a migration, because the process would be restricted to a certain context. Thus, we define the *Activity Spaces*, to serve as a guide of activities to be performed during the migration process.

The Activity Spaces will also be divided into the three areas of concern that run throughout this work, Customer, Solution, and Endeavor, so these activities spaces are explained in detail below, figure 3.5. Within a concern area, each activity space can only be started after the previous activity has been completed, facilitating the monitoring of the process.

In the Customer area, three activity spaces are defined: Study Microservice Architectures, Analysis of the Current Application, and Validation of the Microservice Architecture. The activity space of Study Microservice Architectures is intended to define activities related to the study of microservices. Understanding microservices is essential for the process of migrating from monolithic architectures to microservices, mainly because it takes all the advantages that the architecture can offer to the solution. Another activity space is the Analysis of the Current Application, to develop activities to understand and analyze the application to be migrated, as well as to analyze the advantages that microservices can provide the application after migration. Finally, in the Customer area, the activity space Validation of the Microservice Architecture presents the development of activities related to the satisfaction of stakeholders, ensuring that the application meets the needs of stakeholders.

In the Solution area, the following activities spaces are defined: Designing the new Architecture, Refactoring, Integrating Microservices, Evaluating the Microservice Architecture and Evolve the Architecture. The activity space Design of the new Architecture is the activity space determined for the elaboration of activities to carry out the design of the solution, and model, and realize the idealization of how the architecture will

be implemented. The activity space Refactoring is responsible for defining activities to implement and refactor the application from the monolithic version to the microservices version. The activity space Integrating Microservices proposes to define activities that carry out the communication of microservices, both among themselves and external access to microservices. The activity space Evaluate the Microservice Architecture proposes to define the activities related to the forms of evaluation of the architecture. And the Evolve the Architecture activity space provide a evolution process to the microservice, providing maintenance and update during the existence of the service.

It is important to differentiate the activities spaces Validation of the Microservice Architecture, from the Customer area, and Evaluate the Microservice Architecture, from the Solution area. Both approaches define validation as the main activity, but from the Customer's point of view, the activity is dedicated to understanding and validating from the Stakeholders' point of view, based on the definition of validation protocols; while in the case of the activity space defined in the Solution area, validation is carried out from an architectural point of view, satisfying the characteristics that an architecture based on microservices must satisfy.

In the Endeavor area, we define four activity spaces. The first, Coordinate Activity, defines activities to coordinate the activities that are being carried out in the whole process; the Environment Designation is dedicated to activities that control the environments in which the microservices will be applied, as well as all tasks related to the deployment of microservices in the environments; the Review Architecture activity space provide changes during the migration process ; and Stop the Work to define the end of the work, as well as deliver all the work done on the project. The Endeavor area proposes to define activities that can be correlated through practices with any other tasks and alphas so that they can always be present in the generation of a process.

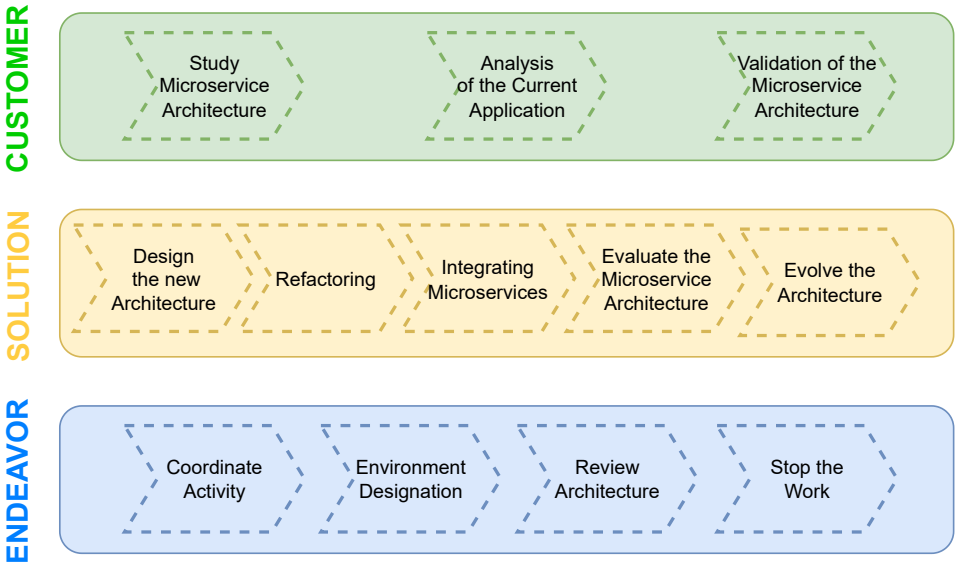

Figure 3.5: M3K Activity Spaces

3.2.1 Activity definition

The definition of activities for defining a sufficiently generic process, such as a metaprocess, to migrate architectures is a complex task, especially when we consider that each application goes through different challenges. Therefore, the use of the definition presented by Essence, on Activity Spaces, proposes to facilitate the use of the metaprocess.

The activities proposed by Essence for the Activity Spaces are designed for building an initial application, but when we analyze them from an architecture migration point of view, especially focusing on architectural views, some tasks are not satisfied, requiring a greater understanding of both the that needs to be worked on, determined in the Customer stage; from the point of view of application refactoring and integration between the different microservices, in the Solution stage, and in how the work must be defined and which roles each actor in the process must follow. In table 3.1 is presented the mapping of Activities Spaces from M3K to Essence, changing from creation of a software system to a definition of a process to migrate monolithic applications to microservice

architecture.

Table 3.1: Mapping of Activities Spaces from M3K to Essence

Areas of Concern	**M3K Activity Spaces**	**Essence Activity Spaces**
Costumer	Study Microservice Architecture	Understand Stakeholders Needs
	Analysis of the Current Application	Explore Possibilities
	Validation of the Microservice Architecture	Ensure Stakeholders Satisfaction
Solution	Design the new Architecture	Shape the System
	Refactoring	Implement the System
	Integrating Microservice	Deploy the System
	Evaluate the Microservice Architecture and Evolve the Architecture	Test the System
Endeavor	Coordinate Activity	Prepare to do the Work and Coordinate Activity
	Environment Designation and Review Architecture	Support Team
	Stop the Work	Stop the Work

3.2.2 Creating Activity Spaces

The definition of activities for defining a sufficiently generic process, such as a metaprocess, to migrate architectures is a complex task, especially when we consider that each application goes through different challenges. Therefore, the use of the definition presented by Essence, on Activity Spaces, proposes to facilitate the use of the metaprocess.

The activities proposed by Essence for the Activity Spaces are designed for development of an initial application, but when we analyze them from an architecture migration point of view, especially focusing on architectural views, some tasks are not satisfied, requiring a greater understanding of both that needs to be worked on, determined in the Customer stage; from the point of view of application refactoring and integration between the different microservices, in the Solution stage; and in how the work must be

defined and which roles each actor in the process must follow.

The activity spaces were defined based on the literature on the definition of migration processes from monolithic architectures to microservices. Figure 3.6 presents the methodology used for the definition of the Activities Spaces for M3K. The first step is the selection of activities presented in the literature for the migration process from monolithic applications to microservices-based ones. These activities are processed filtering the common activities in each process, resulting in a new set of activities. This new set pass throws a dual process, one to the generalization of activities, allowing that any monolithic context application can execute each activity, and a second one to define the competencies and alphas in each activity. This second task varies from how the migration process is conducted on the original work.

The process of selection, filter, and generalization is explained as follows, making a mapping with M3K Activity Spaces and the work of Ponce et al., 2019 and Nordli et al., 2023. Both articles are dedicated to presenting what microservices are, as well as monolithic architectures, so, based on this premise, the activity space "Study Microservice Architecture" is present in all of them. Another point that we consider as an activity space generated from the articles refers to the "Coordinate Activity", which the authors themselves are responsible for leading and guiding during the migration process, validating the existence of the activity space.

In Ponce et al., 2019, a rapid review is carried out to analyze and organize migration techniques from monolithic architectures to microservices proposed by the literature. Below we point out some of the activities proposed by the authors, as well as the mapping with the activities spaces of M3K.

The analysis of the architecture of the system to be migrated is one of the steps proposed by the authors, mainly because it is used as a decision factor of which are the possible candidates for microservices, thus corresponding to the activity space: Analysis

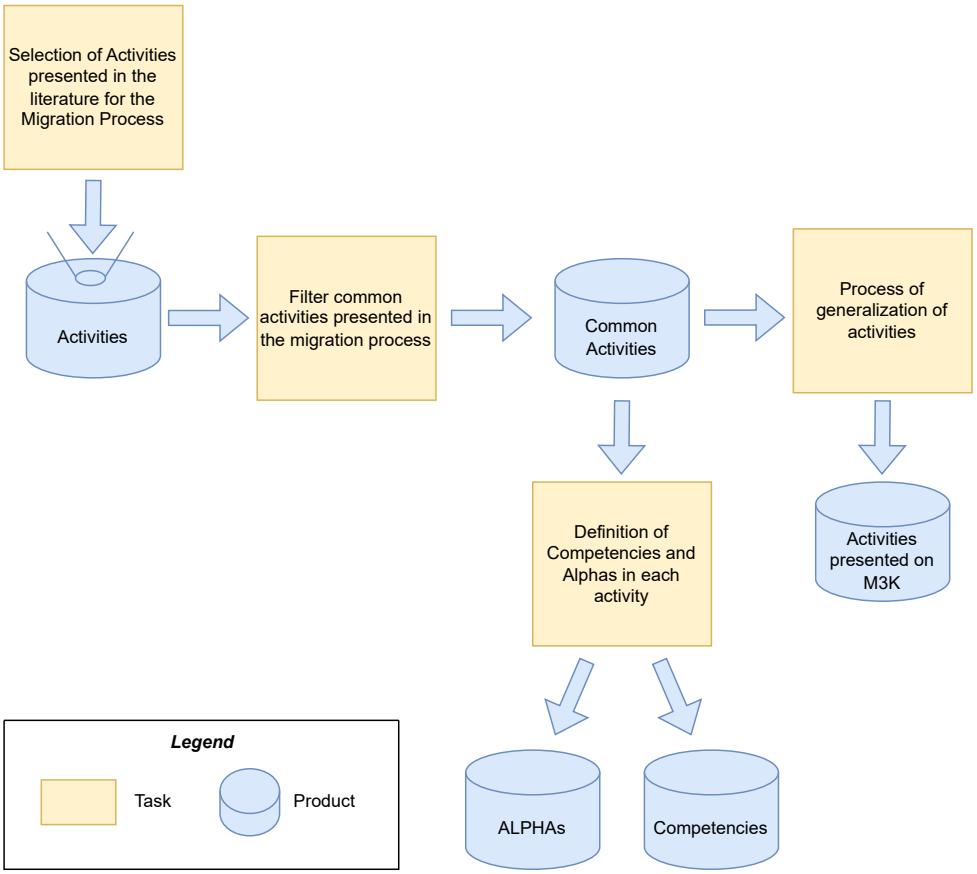

Figure 3.6: M3K Activity Definition

of the Current Application.

Ponce et al., 2019 details the different ways of choosing possible candidates for microservices. This detailing is mapped in M3K to “Design the new Architecture“, so that some works are analyzed work with static and evolutionary analyses, seeking the coupling between the classes of the system; and defining processes at runtime, through the grouping of similar functionalities. These candidates are extracted and organized to then begin the implementation process, this step can be mapped in the M3K to “Refactoring”, in which the microservices will inherit the logic of the original application adapting to the microservices architecture.

Nordli et al., 2023, in the project “Migrating monoliths to cloud-native mi-

croservices for customizable SaaS", explores approaches used in the industry for the migration of monolithic architectures to microservices, differentiating the approaches used for application transactions. The author uses an application example to base himself on the concept of the definition.

The author's migration is based on the use of the MVC design standard, using the Strangler approach, divided into 3 main phases: analyzed, transformation, and implementation. In the analysis stage, the application being migrated is verified, which can be mapped to the M3K defined in the activity space "Analysis of the Current Application", to see how the application works and how it can be separated into different microservices, mapping with "Design the new Architecture" on M3K. It is carried out by identifying and grouping the domains of each module focusing on specific areas of the application.

The second phase consists of transforming the existing infrastructure to support the new architecture, allowing the incorporation of new infrastructures when the transformation of the existing one is not possible, here the activity space "Environment "Designation" is directly defined.

The last part consists of the implementation of microservices, mapping in the "Refactoring" activity space, identified in the first phase, and connecting the microservices in the infrastructure added in the second phase, mapping to the activity "Integrating Microservices".

When presenting the migration example, the author focuses on presenting design decisions, such as the programming language used and how the final architecture should be, also defined in the "Design the new Architecture" activity space; the organization's modeling of how the architecture should meet and how the interfaces should communicate, defined in the "Design the new Architecture", "Refactoring" and "Integrating Microservices" activity space; the understanding of the function of the application and

the additions of new components, defined in the "Analysis of the Current Application" activity space and "Design the new Architecture" activity space, respectively.

The authors define architecture validation methods, validating whether the characteristics of the migrated application configure as a microservices architecture, in which we can perform a mapping for the activity space of "Evaluate the Microservice Architecture".

In the work presented by Ponce et al., 2019, some activities are not mapped in the metaprocess defined in this work because it is not a standard defined in all the migration works analyzed. Ponce et al., 2019 details database checking activities to identify consistency across candidates and microservices. Database restructuring steps are an optional activity given that migrations of applications that do not make database calls would not be possible to be included in the migration, and there are studies that detail that microservices do not require decoupling the database structure, being possible to carry out the sharing a database with several microservices Taibi et al., 2018.

Some extra considerations are presented below:

- Although Ponce et al., 2019 does not present a definition of how the microservices will be implemented, we consider that to validate the applications presented, the authors need a validation step from the point of view of executing the solution, corresponding to the steps of Evaluate the Microservice Architecture.
- Ponce et al., 2019 and Nordli et al., 2023 do not directly define a step corresponding to Stop the Work, but we consider that the results presented in the migration are automatically applied to the factor of finishing the work carried out.
- Nordli et al., 2023 presents activities of the migration process on how the migration should be conducted and carried out, mapping with the M3K the activity space "Coordinate Activity" is generated.
- Although neither of the two works scores "Validation of the Microservice Architec-

ture", which validates that the requirements imposed for the migration are satisfied, we consider that they indirectly bring this characteristic, mainly because each work defines microservices intending to solve a problem, which is achieved with these tasks.

- No literature paper, known by the author of this project, provides an overview of the Evolve the Architecture and Review Architecture activity spaces, which were defined based on the survey (section 5.2) presented by this work.

Activity spaces seek to assist in the development of activities, allowing migration authors to develop specific activities for the context in which they are necessary. Each one of them was elaborated to present a vision from Essence, changing the focus from the development of a new software system to the migration of architectures. Therefore, activities spaces seek to define activities that are generic enough for the metaprocess specification to be well defined, resulting in the elaboration of a quality process to migrate monolithic architectures to microservices.

3.3 Competencies

The skills needed to define migration processes from monolithic architectures to microservices-based architectures are divided again into the concern areas presented by Essence. The M3K defines six competencies distributed in the three concern areas, based on the competencies required by one of the areas, as shown in figure 3.7.

Analyzing the Consumer concern area, it is necessary to define the necessary competencies from the point of view of the business needs for the migration to be necessary and what the objectives are to be achieved. Based on this premise, the competence of Business Understanding is defined, as the ability to understand what the system proposes to execute and what points it seeks for improvement when performing the migration.

In the view of the Solution concern area, skills are those that are directly related

to the vision of the system, from its idealization to its completion. For this concern area, four competencies are defined, namely: Microservice Instruction, Architecting, Refactoring, and Evaluation. In the Microservice Instruction competency, one finds those competencies in understanding and being able to apply the microservices architecture. This competence is considered fundamental to be applied in the context of migration, especially when there is no clear idea of why to adopt microservices and if the architecture proposes to solve problems that the applications that are being migrated are going through. The Architecting competency proposes to be able to architect, through modeling and documentation, a new architecture, from an existing one, specifying how the migration should occur and what details the application must satisfy. The Refactoring competency details the need to be able to refactor an application, having the necessary knowledge of design standards and good programming practices to carry out the activities that require this competency. The Evaluation competency details the need to understand software testing, from the application point of view, as well as the application's architectural validations.

The concern area Endeavor has only one competency, Management, this competency is directly connected with all M3K activities and alphas. All actors in the processes must have management capacity, to deliver the planned microservices and to be able to manage all the other processes necessary to complete the migration.

3.4 Practices

In the context of M3K, a practice follows Essence's practice definition, referring to a concise method designed to guide teams in accomplishing specific tasks and producing corresponding work products.

Figure 3.8 depicts a practice concerning the motivation to elaborate a migration process to microservice. The practice is composed of two activities: Study Microservice

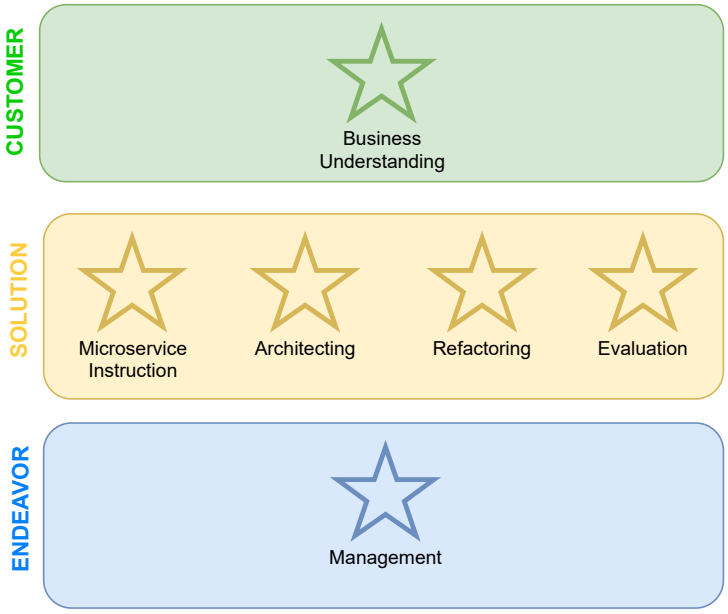

Figure 3.7: M3K Competencies

Architecture and Designing New Architecture. The Study Microservice Architecture activity requests Microservice Instruction competency, which is applied too in the Design of the new architecture activity, with Architecting competency. From the Design the new architecture activity, defines the Services and supports Architectural Changes, generating the new Architecture.

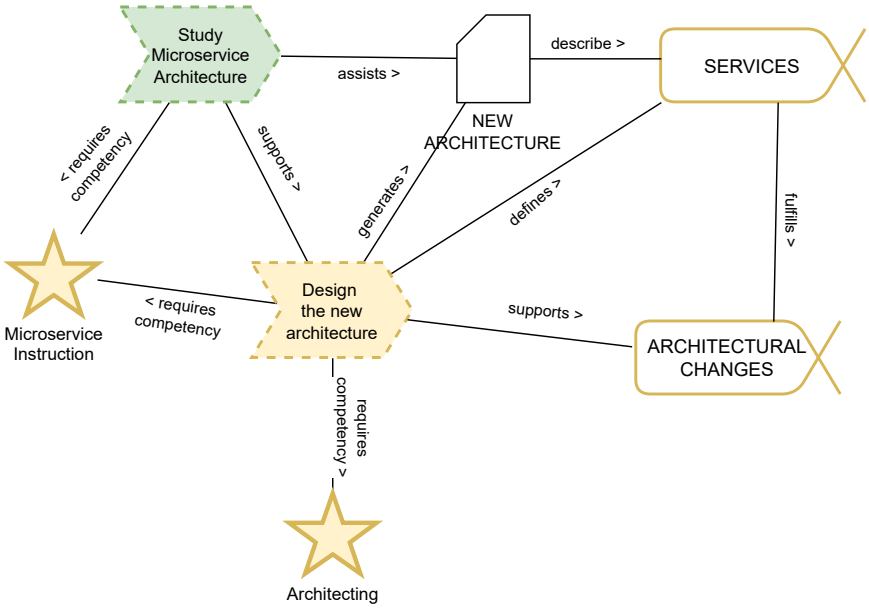

Figure 3.8: M3K Practice Example

4 Related work

Many works related to the migration process from monolithic architectures to microservices are defined in approaches for the (i) migration processes focused on specific contexts, (ii) demonstration of use cases of the migration process and (iii) examples of processes for migration of architecture. In the literature, no metaprocess is defined, based on a specification, that seeks to help develop a sufficiently generic process to carry out the migration, considering aspects such as the migration, the actors involved, and the work necessary to carry out the project.

In this chapter, we present the different approaches presented in the literature, both regarding the migration process and their application. This work is organized into four sections: (i) migration processes from monolithic applications to microservices (Section 4.1); (ii) migration of applications based on surveys (Section 4.2); (iii) definition of competencies to the migration (Section 4.3); and (iv) derivation of process approaches based in the literature (Section 4.4).

4.1 Migration processes

The description of the migration process seeks to define the necessary activities to be carried out in a project, based on a pre-established context. Nordli et al., 2023 explores the approaches adopted by the industry in the migration of legacy architectures, focusing on the migration of single to multitenant applications, using a distributed

system of a sports equipment store as a basis, dividing it into three steps: analysis, transformation (inclusion of components) and implementation, inserting a fourth step to symbolize the finalization of the migration. The activities of the process defined by the authors are standard activities when thinking about the migration process, being necessary to understand the needs of using microservices and define the roles of each participant in the process, issues not addressed by the authors, and which are foreseen in M3K. When analyzed from the point of view of activity definitions, the M3K makes it possible to define different practices for each workgroup, in order not only to focus on the work of each microservice that will need to be developed but to focus on the entire process that is being performed. As an example, a workgroup can already start the phase of defining the architecture of a microservice even before finishing all the analysis of the application, which cannot be carried out by the mentioned work. Another point to be detailed is that the authors focus on components that may not be necessary for other processes, which can lead to extra work to study and use such components, which does not happen in the M3K, in which each company can prepare its process according to your real needs.

In Mishra et al., 2018, an autonomous approach is proposed for migrating monolithic applications to microservices, using tools for usability analysis and data flow, to propose possible microservices and then performing the migration. The process consists of five steps, going through stages of analysis of the application to be migrated to the restructuring of data structures. The authors present a well-defined process of steps to be taken to carry out the migration process. The process begins by (i) analyzing the monolithic application, generating a representation of the monolithic application classes, (ii) performing a division of components, representing the microservices; then (iii) defining the communication interfaces between each component; (iv) proceeding to a stage of database restructuring; and finally, (v) the treatment of problems commonly

related to microservices, such as replication. When analyzed from the M3K point of view, the entire process can be incorporated into the Solution concern area, in which it has exactly the role described by the process. Analyzing from the point of view of the other areas of concern, it is not defined how the tasks should be distributed, much less how this analysis of the application should be carried out to then define the microservices, and steps that M3K seeks to encompass.

Kyryk et al., 2022 defines a process composed of nine sequential steps, with the main focus on restructuring the application data. The authors complement the work by including deployment methods that seek to help microservices and implementation management. Each stage proposed by the authors has a direct mapping with the M3K so that they are considered activities for the activity spaces. For example, if the migration process proposed by the authors were structured by M3K, the *Define the logic and data for new microservice* step would be defined in the activity space of *Design the new Architecture*, but task specification for Stakeholders and Developers could be performed to define how the tasks should be carried out and a more in-depth analysis of the objectives that are intended to be achieved with the migration, adding greater value to the work carried out. Although the authors restrict themselves to the concern area solution step, we can encapsulate each task for each activity space and develop practices to be structured with alphas and competencies, as analyzed in the example above.

In Volynsky et al., 2022, the migration of a management application called Artemis is presented. The authors define two main stages for the migration process, initially focusing on the actual migration of the architecture and moving on to another stage dedicated to the restructuring of the database, satisfying the database-per-microservice standard. The authors propose a 7-step migration process, namely: Repository per Bounded Context, Database per Bounded Context, Monolith Data Access Layer, Database Wrapping Service, Change Data Ownership, Data Synchronization, and

last one Tracer Write, respectively. The process, despite being focused on migrating the application to a microservice, reserves the right to detail the database migration process, to satisfy the condition of independent databases per microservice. The work proposes the use of the Saga pattern to assist in the management of distributed transactions involving several microservices, which can be implemented through events, in which a microservice executes a transaction, launching an event, in which the other microservices subscribe to the event and then execute your transactions. The second part proposed by the author is the use of a framework based on the Domain Specific Language (DSL) to carry out the deployment in the Kubernetes cluster. As the authors' focus is not on defining a process aimed at all the necessary aspects of a migration, but on restructuring the database for the migration, the entire set of ideas worked on can be incorporated into an activity within the M3K activity spaces, intended for database restructuring.

Wolfart et al., 2021 present a process to conduct the migration for modernizing monolithic legacy systems to microservices, detailing common activities and exemplifying with input and outputs of each activity. The work conducted a study relying on 62 papers in a systematic mapping, generating 8 activities covering initiation, planning, execution, and monitoring. The roadmap proposed in the work has not considered stakeholders and focuses only on the activities of the migration process, in which M3K presents, besides a set of activities essentially in the migration process, a set of alphas and competencies necessary to elaborate a migration process.

In Martínez Saucedo et al., 2025, a Systematic Mapping Study was conducted, selecting 114 studies from 1546 studies. The study focuses on the identification of microservices, factors influencing migration, tools used during migration, as well as the issues and benefits associated with migrating from monolithic to microservices architecture. The authors propose an e-commerce application to exemplify the migration from a monolithic to a microservices-based application. They identify two types of migration

proposals: *suggestions of a process and phases to be carried out* and *studies describing reproducible steps for migration*. A process comprising five phases is defined, which includes the following elements: planning, analysis, design, execution, and monitoring. Figure 4.1 illustrates the mapping of activities defined in (Martínez Saucedo et al., 2025) and the activity spaces outlined in this thesis.

The *Planning* activity focuses on reviewing and analyzing the factors driving an architectural migration. The *Study Microservice Architecture* activity space introduces the concept of understanding microservices, including their benefits and challenges. In the Analysis phase, the monolithic system is analyzed, represented in M3K by the *Analysis of the Current Application* activity space. The *Design* phase concentrates on decomposing the monolithic system and defining microservices, represented by *Design the New Architecture*, *Integrating Microservices*, *Refactoring*, and *Environment Designation*. The *Execution* phase involves testing and deploying the new microservices into production, corresponding to *Evaluate the Microservice Architecture* and *Validation of the Microservice Architecture* in M3K. Finally, the *Monitoring* phase involves overseeing the new architecture to ensure performance and quality of service. The *Coordinate Activity*, *Review Architecture*, and *Stop the Work* activity spaces are not defined in this study, as they are not mandatory according to the M3K framework.

The authors do not provide an analysis of the competencies necessary for the migration process and assume that no single technique is completely effective for migration. However, this is the main contribution of this work, as M3K is not a process, but a metaprocess designed to serve as a guideline for migrating from monolithic to microservices architecture.

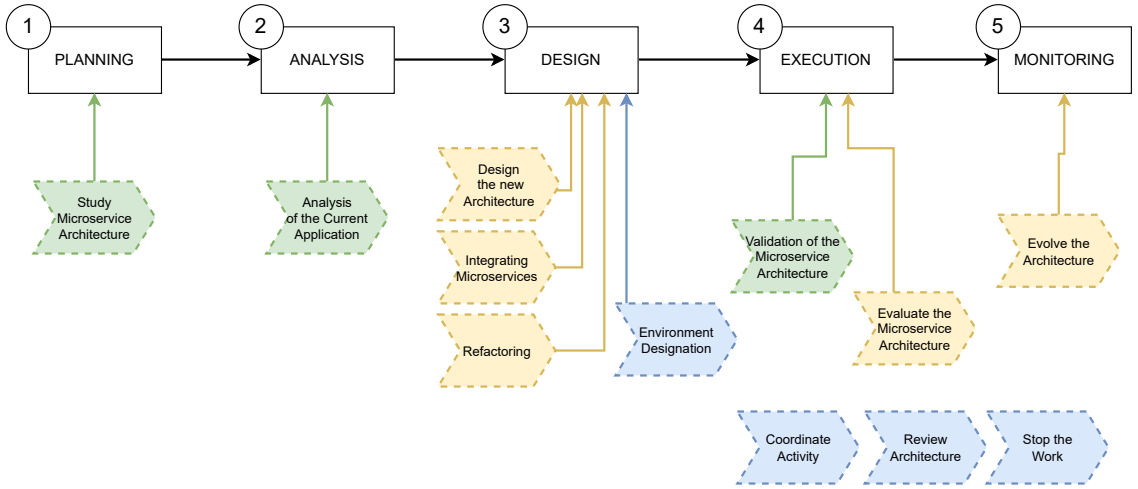

Figure 4.1: Mapping of competencies of M3K to Martínez Saucedo et al., 2025

4.2 Surveys on migration process

The decision to carry out migration is of paramount importance before starting any stage of the process, especially when we consider that several factors must be raised before starting the work. In Auer et al., 2021, interviews are conducted with professionals responsible for migrations from monolithic applications to microservices, to raise evidence of the need for migrations by companies, punctuating characteristics such as maintainability, deployability, and team organization. The author seeks to answer questions such as (i) why companies migrate to microservices, (ii) what information and metrics are collected before and after the migrations and (iii) which of them are considered useful by the participants. The process of identifying the need to carry out the migration in M3K is described in the Stakeholder and Motivation alpha, which can be incorporated into the practices of understanding these objectives, with the analysis activities of the current application.

Ayas et al., 2021 describe the decision-making process, identifying 22 decision points and interviewing 19 professionals with 7.5 years, on average, of experience on microservices projects. The work analyzes activities essential to the migration process

from monolithic applications to microservices-based architecture. These activities are exploring the potential of microservices in the organization, the values of the business for the company, deciding which approach for the migration to follow, what must be re-used from the system, which granularity the microservices must have, which microservices will be implemented in the system, define the order of migration, define new implementation norms and standards. In an organizational dimension is defined activities as how to drive the migration and maintain control, how to organize the organization for the migration, how the tasks will be performed, how to share knowledge, and how to define the operational model for the migration process. The paper presents common activities to a migration process, which allows these activities can be included using the Activity Spaces in the M3K.

Bucchiarone et al., 2018 presents the results obtained with the migration of a banking application, detailing the inclusion of components that are intended to help the coordination of microservices, such as containerization and task automation. Among the activities carried out, the authors detail (i) the understanding of the business functionalities, based on conversations with stakeholders, applying for a priority level order; (ii) the decision of which functionalities would become microservices, based on static analysis, if the functionality was isolated, too big, or shared with many other functionalities; and then, (iii) the microservices implementations. The authors define a set of architectural aspects and business processes to be realized, such as Containerization, Automation, Orchestration, and Integration. In addition to presenting solutions for common problems in the use of microservices, such as the use of a messenger for communication between microservices, it is not necessary to make direct calls between microservices, resulting in less coupling and the non-violation of the interfaces of each microservice. The approach based on a use case to exemplify the migration, allows the reader to have a clearer view of the objectives to be achieved but restricts the application

to a specific context, which does not occur with the M3K proposal, mainly because it focuses on other approaches such as task definitions and motivations for migration, in addition to migrating the actual solution.

Understanding microservices is of paramount importance for defining an actual migration process from a monolithic architecture to one based on microservices. In Garriga, 2018, a framework is defined based on the taxonomy of concepts from the microservice lifecycle. The author conducts a literature study on topics addressed during the microservice generation process and manages to establish essential definitions for their definition, such as Design, Implementation, Deployment, Runtime, Crosscutting Concerns, and Organizational Aspects. Despite not directly portraying terminologies for the migration process, these terms are fundamental for the generation of microservices, which are conceived during the migration process, so we can see a clear reference to them in the M3K. In Design, we have from the stages of understanding the application to the elaboration of how it should be designed and what the expected result will be, which can be incorporated into the study activities on microservices, the current application, the design of the new architecture and the coordination of tasks. The implementation step is defined in the application refactoring process and the environment designation. The Deployment step, Crosscutting Concerns and Organizational Aspects with the Environment Designation. And the Runtime one with the Validation of the Microservice Architecture and Evaluate the Microservice Architecture step. Despite the focus of the process being designated for the creation of microservices, these sets of activities can be restructured to obtain a migration process, which is proposed by M3K, so that the metaprocess proposed in this work can extend the ideas proposed by the authors. Moreover, this work proposes divergences and the raise of other important questions, such as the understanding of the objectives to be achieved and the definition of the activities of the executors of the process.

Ponce et al., 2019 performs a rapid review with the objective of (i) defining the techniques proposed in the literature, (ii) types of systems that the migrations occurred, (iii) the types of validation of these systems, and (iv) the challenges associated with the migration of a monolithic architecture to one based on microservices. Analyzing the process steps that were found in the literature, we have the Model-Driven approach, which seeks to understand the business logic and functionalities; static analysis, through the source code; and dynamic analysis, through functionalities that are triggered at runtime. The authors point out that these techniques, despite being used separately, can also be performed together, and the existence of an association between the use of static and dynamic analysis arises from the need for tools that support both processes. The author, despite performing a superficial analysis of the approaches adopted for the migration process, delimits the migration solution step, referring to the inclusion of processes in the M3K. Although the authors' focus is not on the definition of a process itself, it does not present how these approaches should be carried out or what activities result from these approaches, points that are addressed in the use of M3K, adding value to the architect's migration process.

Di Francesco et al., 2018 surveys with computing professionals to list the activities and challenges encountered in carrying out a migration process. In the research carried out by the author, pertinent questions are raised, such as the most common activities, pointed out by the interviewees, among them the (i) domain decomposition, (ii) the identification of services for the new system, (iii) the application of domain-driven design practices, (iv) system decomposition and (v) building proof-of-concept services to assess migration feasibility. Motivations, how the implementation is carried out, what type of migration is carried out, and what types of information are considered for migration are some other examples of questions raised by the authors, which encourage the definition of a migration process. Therefore, it is possible to carry out a direct analysis between the

search and the M3K, mainly because it considers activities that can be defined in the activity spaces, as well as the competencies and alphas. From the analysis of each point raised and discussions about which application the migration process will be applied, we can use the M3K to define a process aimed not only at the actual application but also at questions such as how to carry out the migration, the participants and the role of each in achieving the desired goals.

4.3 Competencies

There are not many papers concerning the competencies and roles required in a migration process, but Michael Ayas et al., 2024 presents a study about the competencies and roles of practitioners in microservice-based architectures. The authors analyze the technical aspects of the competencies. Each one of the roles can be associated with a direct competency defined by M3K, which can be attributed to one or more competencies.

The authors divide the competencies into a subset of competencies.

- Web Technologies: API Development and Service Integration, Fullstack Development, Front-end Development, and Security and Networking;
- DevOps: Version Control and Quality Assurance;
- Data Technologies: Data Analysis and Data Engineering, and Data Management; and
- Stand-alone: Programming, Data Structures and Algorithms, Mobile development, and Microsoft Cloud Services.

As defined by the authors, the competencies does not focus on generic competency to the migration, but competencies focusing on a specific environment and to the development overview.

The M3K defines competencies as a metaprocess of microservice migration,

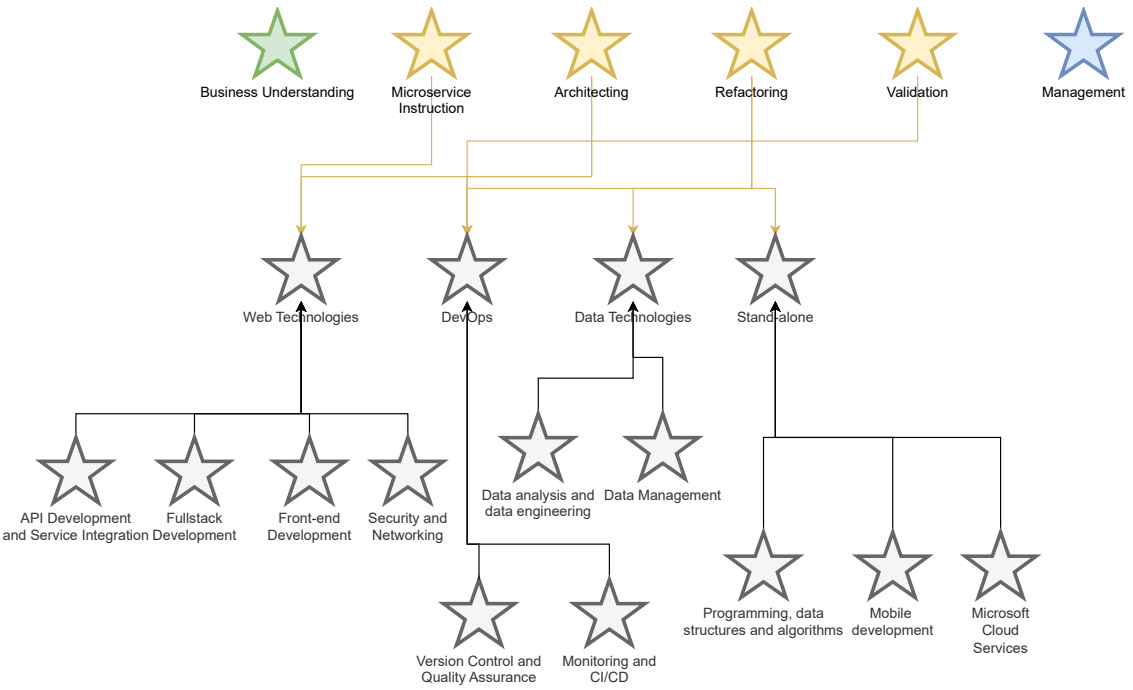

Figure 4.2: Mapping of competencies of M3K to the one presented in Michael Ayas et al., 2023

different from the one presented in the paper, which focuses directly on the development of microservice-based architectures. Figure 4.2 presents an assimilation between both competencies. Considering the competencies required for the migration, presented in the M3K, the authors can propose a subset of competencies, which involves not just the overall development, but the specific competencies of the developers. Another important detail is that all the competencies defined by the authors only are mapped to the competencies of the solution area of concern, which is the area focusing on the direct development and deployment of the application.

M3K proposes much more detail in the overall migration, not only in the development, which is just one part of the migration but not less important. The solution comprehends the development of the migration, creating and updating microservices, but customer and endeavor cannot be negligence. For detailed usage, the paper can be applied together with M3K, but only considering the development team and inside of

the competencies of the solution area, which can provide a good direction to managers and developers.

4.4 Proposal of Generic Process

Make an analysis of the migration process in the literature and define a process are examples of papers presented in Abgaz et al., 2023; Ahmad et al., 2014; Michael Ayas et al., 2023. In Abgaz et al., 2023 examines 35 papers from a defined Systematic Literature Review (SLR) protocol and snowbowling method. One of the main points is that the paper not only presents a composition framework, identifying phases and key elements of decomposition but also identifies metrics and datasets used to evaluate and validate monolith decomposition processes.

Considering the method defined by Abgaz et al., 2023, we can map the defined activities with the ones defined in the solution area of concern in activity spaces. Figure 4.3 provides an overview of the direct relation between both approaches, this indicates that the process defined can be instantiated from the M3K instance. The only difference between both approaches is that the Microservice Optimization is not applied in a specific activity to the M3K, because the M3K considers that during the Design the New Architecture phase, all necessary changes in the systems are already applied. The M3K provides some other activities that are defined by the paper but are not directly related to the Essence Framework, such as the activities designed for the endeavor and the customer area of concern.

The Input Collection analyzes the domain in which the migration process will be performed, which results in a study of a microservice architecture and an analysis of the application. The Monolith Analysis is a deep investigation of the monolith application, which results in the Analysis of the Current Application in the M3K. Microservice Identification is easily associated with the Design of the new Architecture, which will

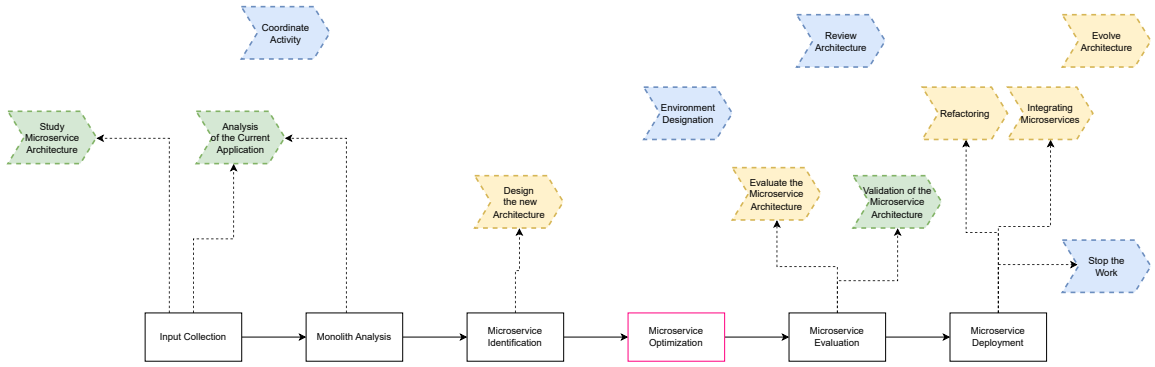

Figure 4.3: Association of activities proposed by Abgaz et al., 2023 and M3K

identify which functionalities will be transformed into microservices.

Microservice Evaluation can be divided into two types in the M3K, but results in analyzing the migration process and validating if the goal was achieved with the migration. In the end, Microservice Deployment resumes in turn the microservices for the stakeholders, resulting in the Refactoring, Integration of Microservices, and Stop the Work activities spaces in the M3K. The activities proposed by the authors do not provide a track to the Coordinate Activity and Environment Designation activities spaces provided by M3K. These two activity spaces must be considered taking into account how the process will be performed and which environment these applications should be considered, especially if it makes necessary a deep control of the resources of the machine.

Michael Ayas et al., 2023 present a full analysis of the processes used by the migration process, such as interviews with specialists and discussions over StackOverflow discussions. The Systematic Migration presented by the authors can be adapted to the activity spaces, or even serve as a base to define the states of the alphas. The authors go beyond specifying the necessary technical activities to be performed during the migration.

Figure 4.4 presents a mapping between the activities with the M3K activity spaces. The paper provides a example of the application of M3K, concerning not just the development and deployment, but the role of the Stakeholders, which is shortly reported in the paper, such as “define criteria for decomposition”. The authors define a specific

approach for the migration, but it makes the migration for a specific domain.

The authors divide the migration over two cycles, the *Systemic Migration* and another one defined by *Activities in the Technical Migration*. Each one of the cycles proposes a subset of activities divided into three sets, the *Migration planning*, *Executing migration*, and *Setting up Supporting Artifacts*. From these three sets, it is possible to make a direct relation with the area of concerns defined in the M3K, but with the main difference that it focuses only on the activities performed in the migration (planning, execution, and supporting artifacts.

Some activities like those focusing on the *Validation of the Microservice Architecture* and *Evaluate the Microservice Architecture* are not presented by the authors, but are pointed to future works. Concerning the Activity Space *Coordinate Activity*, there is no direct reference to the activities that should be performed by the authors, just activities concerning the direct implementation of the solution, e.g. creation of the microservice, and not designation and track of the activities that will be performed, such the ones provided by the M3K. There is no mention in the *Evolve the Architecture*, *Review Architecture*, and a direct reference to the end of work provided by the M3K, e.g. *Stop the Work*.

When we analyze all of the Activity Spaces presented by M3K and make the link with the ones provided in Michael Ayas et al., 2023, the activities are much more focused on the implementation of the microservice itself, and not on the overall required process for the migration, e.g. team coordination and the evolution scenario of the application.

In Ahmad et al., 2014 an approach similar to the one adopted for M3K is presented, applying the OMG Architecture Driven Modernization (ADM) framework. The proposed framework focuses on four activities: (i) architecture migration planning, (ii) architecture recovery and consistency, (iii) architecture transformation, and (iv) architecture-based development of cloud-enabled software. This framework is defined as

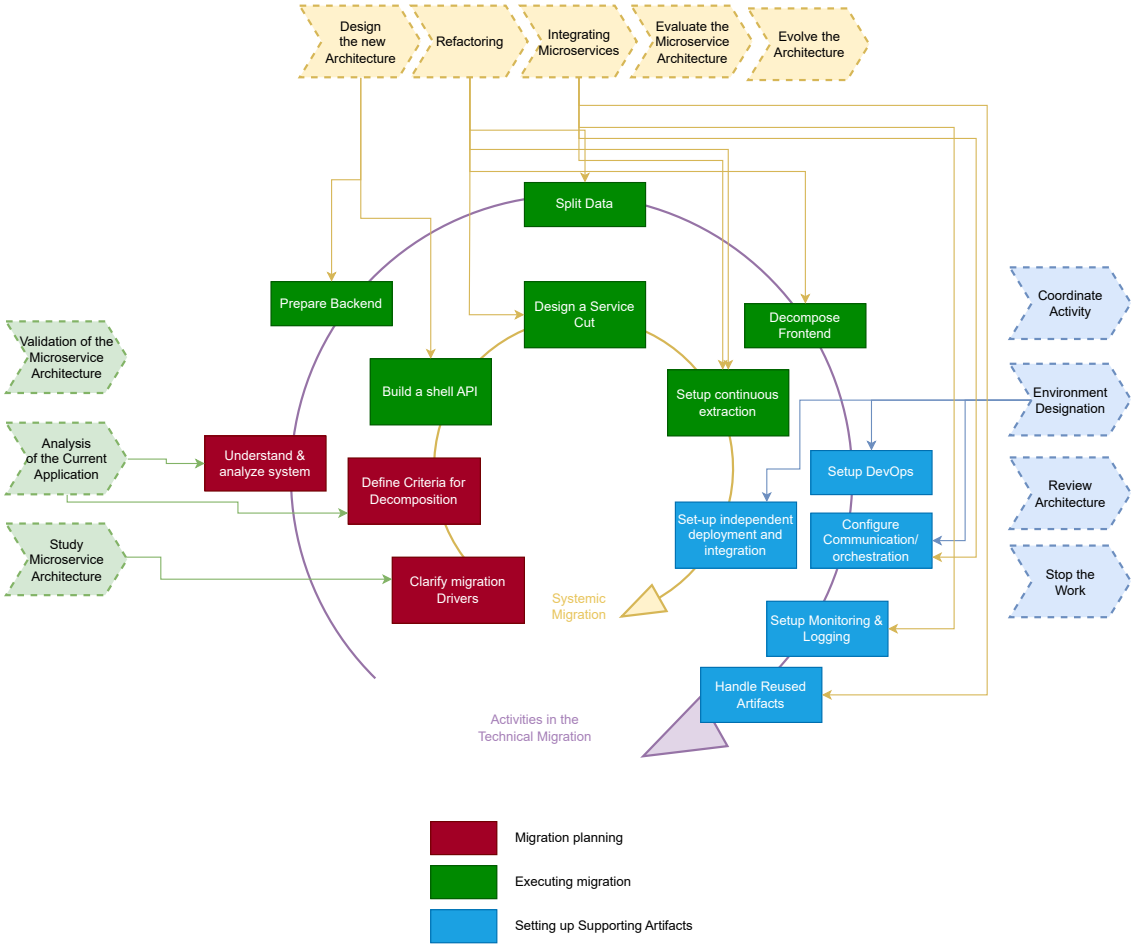

Figure 4.4: Association of activities proposed by Michael Ayas et al., 2023 and M3K

Legacy-to-Cloud Migration Horseshoe. Unlike the other papers.

The use of ADM by the authors is similar to the choice of Essence to M3K, the overview of different roles of the migration process, dividing it into business architecture, application and data architecture, and technical architecture. The authors divide the framework into four main processes composed of a set of subprocesses and activities. This division can be directly assimilated with the usage of activity spaces and states of alphas. Besides similar to M3K, the process is more descriptive to the solution environment, providing decisions focusing on the development and deployment of the application, different from M3K that focuses not only on the technical aspects but on the fundamental development aspects, such as the endeavor area of concern.

All three analyzed papers propose a general process applied to the migration of legacy applications to microservice-based architecture, based on the literature. Each activity can be considered as an instantiate of the M3K, because it can be assigned to an activity space of the M3K, based on its definition. This mapping shows how M3K can serve as an initial application of the metaprocess to define the migration process, having detailed activities.

4.5 Discussion

In this chapter, we categorize the existing literature on the migration from monolithic architectures to microservices into four sections: Section 4.1, Section 4.2, Section 3.3, and Section 4.

In Section 4.1 we present a set of processes for the migration of monolithic architecture to a microservice-based one. Table 4.1 summarizes key contributions from previous studies, outlining their goals, the steps they propose for migration, and the gaps identified in each approach.

The goal of this work is not to propose a single prescriptive process to address the

shortcomings of existing approaches. Rather, it aims to introduce M3K as a metaprocess framework that can be instantiated to create migration processes tailored to specific needs. M3K supports the generation of customized processes that replicate the strengths of the reviewed approaches and address their respective gaps, offering a more flexible and comprehensive foundation for architecture migration.

In the following section (Section 4.2), we present a review of surveys conducted by various researchers that emphasize the need for a metaprocess to guide the migration from monolithic architectures to microservices. These studies focus on identifying and monitoring the key attributes involved in the migration process. For example, Garriga, 2018 proposes a framework grounded in the taxonomy of microservice lifecycle concepts, highlighting the specific objectives that drive such migrations.

Bucchiarone et al., 2020, in turn, report findings from a completed migration in a real-world banking application, providing practical insights into the challenges encountered. Similarly, Di Francesco et al., 2018 present a survey of IT professionals that compiles common activities and challenges faced during migration efforts.

Furthermore, in the Competencies section (Section 4.3) of this work, we provide an overview of the essential skills required to develop microservice-based systems. These are then mapped to the competencies identified by M3K, illustrating how these competencies are critical for successfully executing a migration process.

Finally, Section 4.4 presents a set of generic processes for migrating from monolithic architectures to microservices. Table 4.2 summarizes the key gaps identified in each of the reviewed studies. Although only a limited number of works focus specifically on the development of generalized migration processes, M3K is designed not only to bridge these gaps but also to support companies and developers in improving the structure and effectiveness of their migration strategies.

Table 4.1: Summary of related work on monolith to microservices migration process

Source	**Goal**	**Proposed Steps**	**Identified Gaps**
Nordli et al., 2023	Migration process	Analysis, Transformation, Implementation, and Finalization	Lack of guidance on microservices adoption, undefined roles, insufficient support for parallel work, and no distinction between mandatory and optional components
Mishra et al., 2018	Structured five-step migration	Monolith analysis, component division, service communication, database restructuring, and handling migration issues (e.g., replication)	No definition of task distribution, execution guidance, or microservice identification criteria
Kyryk et al., 2022	Nine-step migration process	Focus on data restructuring	Emphasis on technical aspects only, neglecting stakeholder and developer involvement
Volynsky et al., 2022	Seven-step migration with DB focus	Focused on database migration	Inflexible regarding microservices architecture definition
Wolfart et al., 2021	Synthesis of 62 papers in a mapping study	Defined eight common migration activities	No reference to stakeholders or required competencies
Martínez Saucedo et al., 2025	Five-phase migration model	Planning, Analysis, Design, Execution, and Monitoring	No consideration for activity coordination or architectural evolution

Table 4.2: Summary of studies on generic migration processes and their limitations

Title	**Strengths**	**Limitations**
Abgaz et al., 2023	Proposes a well-founded migration process and identifies relevant metrics and datasets for evaluation and validation	Lacks focus on stakeholders and developers; no clear definition of environment setup or coordination of activities
Michael Ayas et al., 2024	Derives a migration process based on StackOverflow discussions, emphasizing technical aspects	Does not define activities formally; limited to application development without consideration for evolutionary activities
Ahmad et al., 2014	Defines four activities using the ADM framework to guide the migration process	Focuses exclusively on the solution environment, without specifying where or how activities should be executed

5 Validation

M3K validation is carried out based on a case study divided into two stages. The initial stage encompasses a comprehensive case study wherein a process is instantiated based on the metaprocess. This instantiation process entails the systematic execution of all phases delineated in the metaprocess, commencing with the initial steps and culminating in the successful migration, culminating in the delivery of the system in a microservices architecture. Subsequently, the second stage involves the solicitation of insights from developers that participated in migration of monolithic applications to microservices. The objective of this survey is to expound upon the metaprocess and scrutinize the efficacy of microservices in the realm of modernizing software architectures.

The subsequent section is structured as follows: Section 5.1 delves into the application of the metaprocess within a specific process, providing a detailed account of the instantiation procedure. In parallel, Section 5.2 elucidates the outcomes derived from the expert survey, shedding light on the perspectives and assessments of professionals engaged in the migration of monolithic applications to microservices.

5.1 Instantiation of the Metaprocess

M3K aims to help researchers and companies migrate from legacy applications to those based on microservices, thus applying a process of architecture modernization. Therefore, the case study will be based on the work described in (Batista et al., 2022),

as it is applied in an industrial context with an academic foundation, and inspired this work.

Eagle is a B2B product used by thousands of national and global companies, serving millions of daily requests. The application is a software module, developed by Synchro, which enables tax calculation services in the cloud as a SaaS. As a base scenario, let's simulate that the migration process to microservices has not been carried out and that Synchro wants to migrate its application using the metaprocess defined by M3K. Let's also consider that the migration team includes the project director, the project manager, a team of developers, and software architects. This initial definition of the team working on the migration is just a guide to the players involved in the migration, as well as being a common composition in software development companies.

The process definition is carried out from the instantiation of the M3K with experiences in the initial Eagle migration process, since the author of this work participated in the migration process and experiences in the scope of software development at the company.

This section is defined as an instantiation of the M3K and has the following organization: in subsection 5.1.1, the overview for each alpha is defined; in subsection 5.1.2, the activities present in the migration to the application are defined, based on the activity spaces; and in subsection 5.1.4, the practices are defined. As the process is an instance of M3K, the definitions of each alpha, activity space, and competencies are maintained.

5.1.1 Alphas overview

This section revisit the M3K alphas used in the migration, applying to the context of the application each state: Objectives, Stakeholders, Architectural Changes, Services, Services Environment, Architectural Team and Development Team. These

alphas are discussed in the context of their application within the migration process and how they will be utilized throughout its execution.

Objectives alpha

The **Objectives** alpha aims to discern the challenges inherent in the application, with its initial phase denoted as the *Operation* state, shown in figure 5.1. Synchro, a prominent cloud-based tax solution provider in Brazil, extends its services to a diverse clientele comprising both national and multinational corporations. The primary objective of Synchro is to streamline the intricate Brazilian tax calculation system, allowing companies to leverage this advantage and focus on their operational activities.

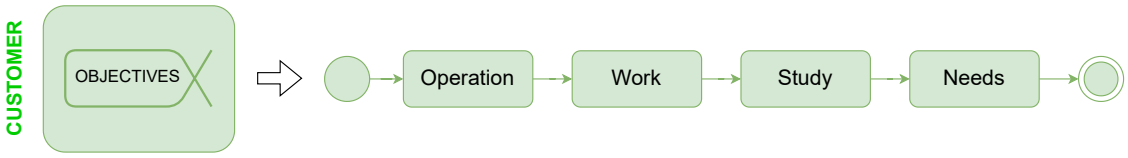

Figure 5.1: Eagle Objectives' states in M3K

The *Work* state of this study involves a comprehensive understanding of the application, with a specific emphasis on the Eagle application. Eagle is a specialized product tailored for corporations engaging in Brazilian tax calculations. Given the intricate nature of tax legislation in Brazil, considered one of the most complex globally, Synchro endeavors to simplify this process. Eagle is meticulously crafted through modular components within a singular monolithic application, facilitating the seamless integration of diverse modules to augment the overall tax calculation capabilities.

In the *Study* state, the application is employed, and Eagle is seamlessly integrated directly into business applications by invoking the legacy system through pre-configurations executed by Synchro's support team. Given the diverse taxation methods inherent to companies in disparate fields of activity due to varying tax rules, this phase is confined to presenting the system's behavioral attributes.

The subsequent stage termed the *Needs* stage, leverages insights derived from the Study state as ascertained from relevant literature. The work by (Batista et al., 2022) delineates a collection of characteristics emanating from the escalating size and complexity of the application. These encompass heightened coupling between modules, precluding the demarcation of clear boundaries between diverse implementations. The modules include elevated maintenance costs and a disincentive for the integration of new tools, which becomes progressively more time-intensive, thereby impinging on time-to-market considerations. Inefficient and costly deployments are evidenced, with minor modifications capable of inducing downtime in the overall system. The pervasive high coupling adversely influences the scalability of the system, necessitating the scaling up of the entire application when increased resources are required for a specific module.

Stakeholders alpha

Within the **Stakeholders** alpha phase in the figure 5.2, this study delineates the specification of the application earmarked for migration, coupled with a comprehensive comprehension of how each company within the context utilizes said application. The identification of stakeholders for the application is predicated on the analytical insights expounded in the paper presented at COMPSAC 2022. In the *Use* state, an in-depth analysis unfolds, adopting the perspective of a project manager vested with the responsibility of comprehending the functionalities of each module within the Eagle system, their interoperations, and communication protocols, notably with the database.

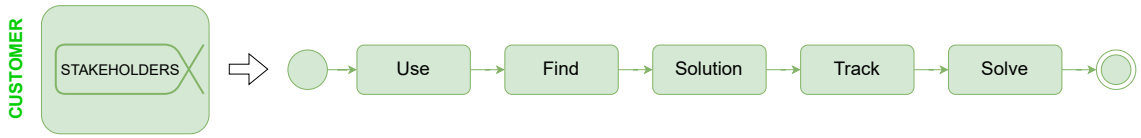

Figure 5.2: Eagle stakeholders' states in M3K

Transitioning to the *Find* state, a preliminary analysis is executed to discern the evolving needs of the Eagle system necessitating alterations. Leveraging the Stakeholder's

role in facilitating system analysis, this phase seamlessly integrates with the Needs state of Objectives. In the ensuing *Solution* state, the imperative to migrate to a microservices architecture is delineated, driven by the pursuit of scalability and performance enhancements across various application modules. This strategic shift is designed not only to afford the company internal benefits but also to extend novel services to customers and affiliated entities.

The Stakeholders assume a pivotal role in overseeing the ongoing migration progress, as encapsulated in the *Track* state, and ultimately culminating in the validation of the migration process during the *Solve* state. This meticulous progression ensures a comprehensive examination of the migration's success, affirming the alignment of the adapted system with the defined objectives and overarching organizational aspirations.

Architectural Changes alpha

In the **Architectural Changes** alpha, the components and the strategy for breaking down the monolithic version are defined, presented in figure 5.3. The *Agreement* states, that a meeting is held to understand the motivations of stakeholders in undertaking the migration, aiming to identify issues in the application. All involved parties must agree that these are indeed problems affecting the application and that a change in its architecture is genuinely required to address these issues.

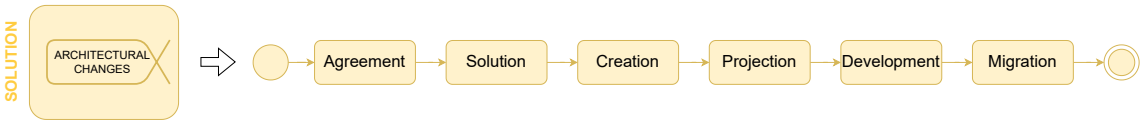

Figure 5.3: Eagle architectural changes' states in M3K

In the *Solution* state, stakeholders must comprehend what microservices are and how the migration can benefit the application through these changes, thereby proposing a solution to the identified problems. With the Synchro team's understanding of the benefits of microservices, a change in architecture is proposed.

The *Creation* state, the Synchro team seeks to structure Eagle to understand how modules communicate with each other and how they can be transformed into microservices. Through meetings, it is determined that the migration will be done incrementally, starting with one of Eagle's main modules, Simula. Using modeling tools, the microservice architecture and communication between the user and the microservice are defined.

Building upon the definitions made in the *Creation* stage, the *Projection* stage analyzes the microservice architecture and validates that the system's structure indeed satisfies microservices architecture, assigning a specific function to the application and defining a Gateway to handle communication between each user and each microservice.

Microservice development is initiated (*Development* state), with software developers working alongside software architects to guide the work process, resulting in cooperative efforts between both teams to achieve the final product. As application progress unfolds, new modules are reconsidered for microservices, initiating a parallel process related to Architectural Changes. In the end, the architecture is executed in the application, concluding the flow of the Architectural Changes alpha, resulting in the *Migration* state.

Services alpha

In the realm of **Services**, the alpha phase entails the creation of each component, presenting each state in the figure 5.4. Following the realization of the architecture, services are implemented according to the defined Architectural Changes. Initially, it is crucial to understand the desired outcomes of each microservice (*Definition* state): What roles does it play? What goals are expected to be achieved within the overall architecture? The first module, when defined, brought the immediate benefit of scalability, catering to increased user requests. As the system operates using multi-tenant technology, application scaling is possible through the number of replicas, benefiting the module without the

need to scale the entire application. The microservice is intended to be developed in JAVA, utilizing the Spring Framework development stack. Consequently, a development environment is provided for deploying the application.

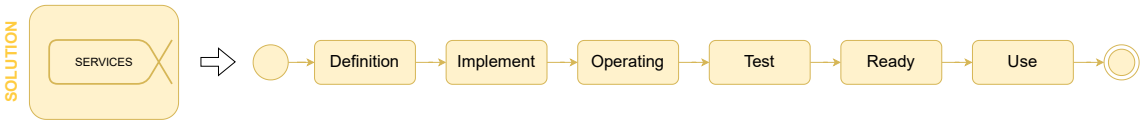

Figure 5.4: Eagle services' states in M3K

In the *Implement* state, developers embark on creating the microservice. A project decision dictates that the microservice should not utilize the logic of the monolithic application, necessitating the application of logic to perform calculations. Since the calculation is performed directly in the database, this step does not directly impact the calculation itself; however, connecting to the database and implementing the necessary logic to execute the calculation is essential, resulting in the *Operating* state. Following the operation state, a series of tests (*Test* state) are conducted to validate the accuracy of the calculation, as well as to test unexpected behaviors such as lack of user validation, negative values, and other inconsistencies.

In the *Ready* stage, all tests have been successfully validated, and the application can enter the homologation phase for use by companies. During this stage, the final environment is defined, and the application becomes available for usage. Finally, in the *Use* stage, users of the microservice can make API calls for utilization.

Services Environment alpha

The **Services Environment** alpha defines the environment in which each component will operate. The alpha is a standard for all environments, often reused, eliminating the need to restart the entire validation process for the alpha.

The initial state, presented in figure 5.5, is the *Define*, which represents the architecture team that determines in which the microservices will operate. As an option,

Kubernetes integrated with CI/CD is chosen as the environment, aiding in the administration of microservices and their configurations. The *Available* state validates that the environment is ready for use, allowing the development team to start working on microservices. The next state is the *Components* state, beginning to analyze the need for different components such as Message Broker, Tracing Server, Metrics UI, and APM Server.

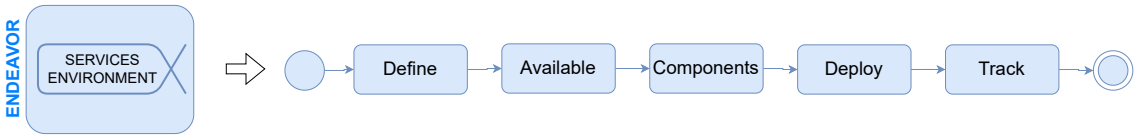

Figure 5.5: Eagle services environment' states in M3K

In the *Deploy* stage, the microservices are deployed in the Kubernetes environment, enabling the development team to test the application's behavior. Finally, the *Track* state analyzes the microservices' performance against the initial objectives of the application, conducting load tests and validations. In cases where a microservice exhibits different behavior than planned, a deeper analysis is required, returning to the Definition state of the Services Alpha.

Architectural Team alpha

These states, figure 5.6, define the roles of each team and outlines how the migration should be executed.

The team of software architects and developers, in collaboration with stakeholders, initiates a preliminary study on microservices to comprehend their advantages, disadvantages, and implementation methods, conducting the Study stage. Following this phase, the team evaluates the objectives intended to be achieved through the migration, scrutinizing all current architectural issues and considering potential system solutions. As the desired objectives align with microservices, migration emerges as the

most viable solution, marking the completion of the Goals stage. Tasks are then carried out in conjunction with microservices planning, with responsibilities divided between development teams and the infrastructure team, as the environment must be prepared for implementation. One task involves implementing a continuous deployment and delivery system, enabling developers to deploy onto the available infrastructure seamlessly.

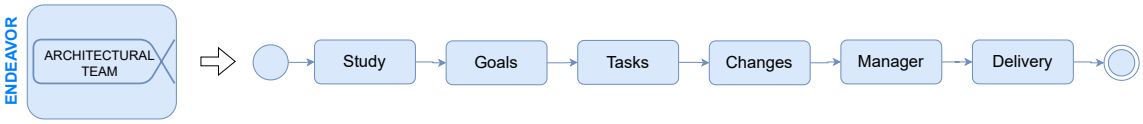

Figure 5.6: Eagle architectural team's states in M3K

During the Changes stage, architectural adjustments are made to accommodate the microservices architecture, creating necessary microservices from modules available in the legacy application. Throughout the microservices delivery project, the architecture team is tasked with monitoring the evolution and deliveries of microservices, as described in the Manager stage.

Finally, the Delivery stage is responsible for delivering the desired microservice, ensuring that the architecture team has validated all necessary changes. Consequently, users can utilize the developed microservice.

Development Team alpha

The **Development Team** Alpha defines the team as responsible for the implementation and the distribution of tasks to achieve the final product.

For the development team's alphas, in figure 5.7, it all begins in the *Understand* state, where all developers must comprehend the role of the microservice that will be implemented and its value to the company. In the *Division* state, tasks are divided, defining each team member's role in the final product delivery. During the *Implement* state, developers commence their function in system development, implementing the necessary logic and processes for the microservice.

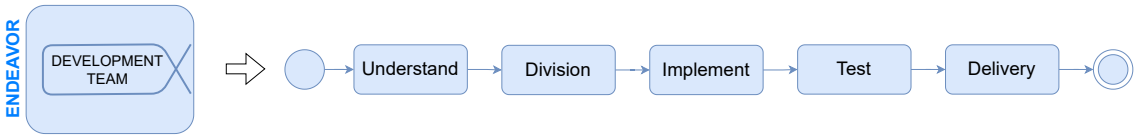

Figure 5.7: Eagle development team's states in M3K

After implementation, the *Test* state begins, validating implemented methods and the correctness of the microservice, ensuring that the final result meets expectations. Finally, the *Delivery* state involves the delivery of the tested microservice by the development team.

5.1.2 Activities

Using the M3K activity spaces, the activities defined in figure 5.8 are obtained. The activities are defined based on the objectives that the migration is intended to achieve in each of the concern areas.

Like the activity spaces, the activity process is not just a sequence of steps but serves as a flow indicator that can be followed. As per the design decision proposed by the original article, each microservice will be developed individually, so once the process has started and the basic concepts have been understood, it can focus solely on defining and implementing the microservices themselves, without having to revisit how a microservice architecture works every time the process is executed.

In the **Customer** concern area, four activities are generated, all focusing on activities related to the migration actors. The proposed activities are:

- *Understand the Microservice Architecture*: Study microservices, understanding how the architecture works and its main advantages and disadvantages.
- *Analysis of the Eagle Application*: Run the application and analyze which points could be improved in the application.
- *Detect Candidates for Microservice*: Choose one or more candidates for microservices.

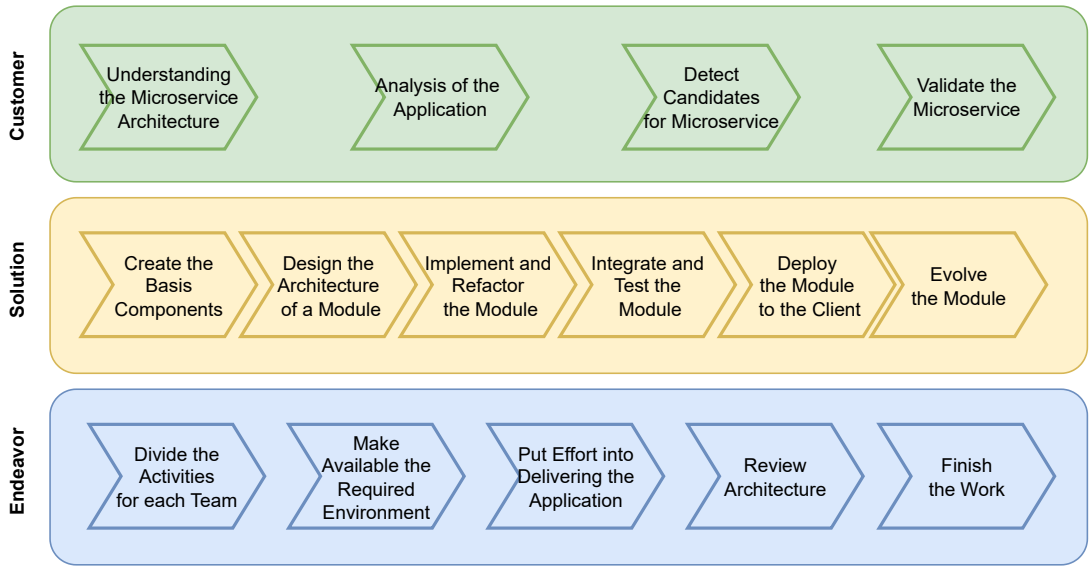

Figure 5.8: Activities proposed for the Eagle migration for microservices architecture.

As an initial proposal, the Simula module was the first to be chosen, so the module can move on to the migration process, and other modules can be started as soon as Simula is finished. If another development team is incorporated, another candidate module can be assigned to this new team, paralyzing development efforts.

- *Validate the Microservice*: After all the efforts have been made to develop the module, it is time to validate that the module meets the requirements and that it can now be assigned to another environment for approval, thus finalizing the entire process of migrating the module to a microservice.

In the context of the Customer concern area, the last activity does not characterize the end of the process, mainly because the modules are not all migrated at the same time. Therefore, after the Validate the Microservice activity, the process moves on again to the third activity, Detect candidates for microservice, and starts the migration process again from this point in this concern area. The migration completion activity is defined in the Endeavor concern area, which is presented later.

Solution is the concern area represented by the implementation of the archi-

tecture and microservices, which are named Services. Each of the activities is defined below:

- *Create the Basis Components*: Some components are necessary for the microservice architecture. This step is linked to Understand the Microservice Architecture in the Customer concern area. As a decision, components such as Gateway and Message Broker are incorporated to perform redirection for microservices, without having to expose all microservices to the external network and to make it possible to exchange messages between microservices and other components, respectively. Understand other components, such as application logs and tracing services.
- *Design the Architecture of the Module*: This activity begins the design of the architecture of a chosen candidate, module, to be migrated, so that it is clear to the development team what components are needed and what functionalities the microservice should have.
- *Implement and Refactor the Module*: The implementation and refactoring process puts into practice all the decisions related to the design of the module and creates the actual microservice, making it usable in a test environment.
- *Integrate and Test the Module*: Integration and testing aim to integrate the microservice with the components needed to run the solution externally, such as the gateway and log aggregator. The integration process also relates to integration into the overall architecture of the application, i.e. how the new microservice is located when the entire application is analyzed. It also includes software testing to guarantee the microservice's possible responses to various user behaviors.
- *Deploy the Module to the Client*: The deployment activity is dedicated to making the microservice available to the approval environment and finalizing the entire architecture migration stage.
- *Evolve the Module*: Perform maintenance and necessary updates as system updates

occur, such as changes in database and logical business.

The first activity in the Solution concern area, Create the Basis Components, is an optional step for each practice module. Since the basic components for a given microservice are already in place, such as the Gateway, for example, there is no need to create another component. However, in the case of specific components for a microservice, this activity can be included in the execution of the practice.

Finally, in the **Endeavor** concern area, five activities are presented: Divide the Activities for Each Team, Make Available the Required Environment, Put Effort into Delivering the Application, Review Architecture, and Finish the Work. Each one is detailed below.

- *Divide the Activities for Each Team*: Once the process of analyzing the legacy application has been completed and the understanding of microservices is clear, it is necessary to divide up the migration activities, either defining which people will be assigned to prepare the environments for implementing the solution, which will work on which module and who will be responsible for implementing the components external to each microservice. This stage is dedicated to assigning a role to each actor in the migration process.
- *Make Available the Required Environment*: Once the process of migrating a module has begun, the application's development environment must be functional. This activity is dedicated to making this environment usable for developers, simulating a homologation environment.
- *Put Effort into Delivering the Application*: All efforts to carry out the migration are necessary with this activity, the developers must dedicate themselves to delivering exactly as planned at the module architecture design stage.
- *Review Architecture*: Once an architecture has been defined at the beginning of the process, a new update to the architecture may be necessary, facilitating flexibility

during microservice development.

- *Finish the Work*: The Finish the Work activity is only achieved when all the modules have been migrated, i.e. until the entire architecture has been migrated to microservices, ideally, or when there are no other modules to be migrated.

In the Endeavor concern area, the process can go from the first activity directly to the third activity and then back to the first activity, starting the cycle all over again, since the second activity may not need to be carried out again.

The table 5.1 shows the association between activity spaces and activities, as defined above.

Table 5.1: Mapping of Activities of Eagle Migration to Activities Spaces from M3K

Areas of Concern	**M3K Activity Spaces**	**Eagle Activities**
Costumer	Study Microservice Architecture	Understanding the Microservice Architecture
	Analysis of the Current Application	Analysis of the Eagle Application and Detect Candidates for Microservice
	Validation of the Microservice Architecture	Validate the Microservice
Solution	Design the new Architecture	Create the Basis Components and Design the Architecture of a Module
	Refactoring	Implement and Refactor the Module
	Integrating Microservice	Integrate and Test the Module
	Evaluate the Microservice Architecture and Evolve the Module	Integrate and Test the Module and Deploy the Module to the Client
Endeavor	Coordinate Activity and Review Architecture	Divide the Activities for each Team
	Environment Designation	Make Available the Required Environment
	Stop the Work	Put Effort into Delivering the Application and Finish the Work

5.1.3 Competencies

From the definition presented of Competencies in the M3K, the following definitions can be applied to the context of tax calculation application and they are resumed in figure 5.9.

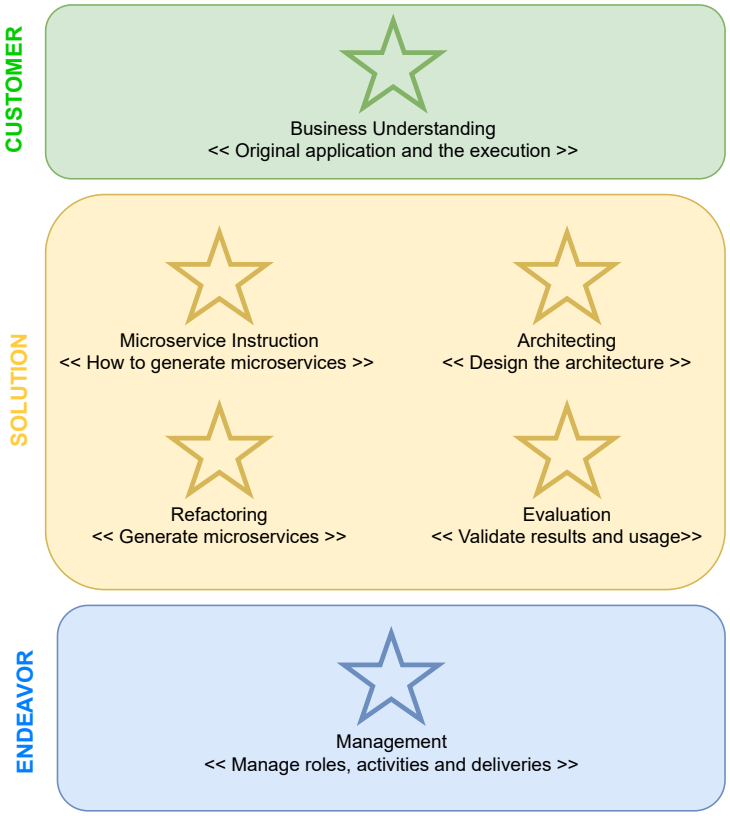

Figure 5.9: Competencies applied in the Eagle solution.

Business Understanding

Each participant in the migration process must comprehend the role of the module, Eagle, within the company and understand how the associated benefits will impact the system's usability, thereby attracting more clients.

Microservice Instruction

Understanding how microservices function and their benefits enables users to bring quality attributes to the system.

Architecting

In a migration process, the ability to architect the system allows for better interpretation and understanding by all those who need to use or modify the application. The main goal of the Architecting competency is to be able to design the architecture that will be applied in the migration.

Refactoring

Decisions regarding code reuse or the development of new code can be challenging, especially when a system has complex logic that is difficult to reuse. Architects and developers must understand the principles of code refactoring to make informed decisions, as well, be able to start the development of the microservices that will be worked, in this context, *Simula* functionality.

Evaluation

In addition to a well-implemented solution, developers must deliver a correct solution, anticipating potential errors that users may encounter and addressing such conditions. And at the end distribute the application to clients to be able to validate the deployment.

Management

Efficient management of all activities being performed and their proper execution is essential for the delivery of the final product. Therefore, application stakeholders need

to be capable of administering these actions.

5.1.4 Practices

Considering the definitions provided in the Alpha, Activities, and Competencies, it is possible to define the necessary practices for the development process. Through a system analysis, four main activities can be abstracted, many of which can be reused, and in some conditions, their complete execution may not be necessary.

Practices can be defined in different ways, either in a way that details each step of the migration or in a more general way, presenting a single development process for all microservices. Below are four example practices for the Eagle migration.

Figure 5.10 shows the practice for analyzing Eagle's legacy application. In this practice, the main objective is to understand how this needs analysis can be carried out. Considering the Eagle migration article in a multi-tenant environment, the interest comes from the Stakeholders, who have the domain of understanding the application. The activity then starts with the analysis of the original application, generating motivations, which generate a product, which is the necessary changes. Finally, these motivations served as the basis for producing the microservices (Services alpha).

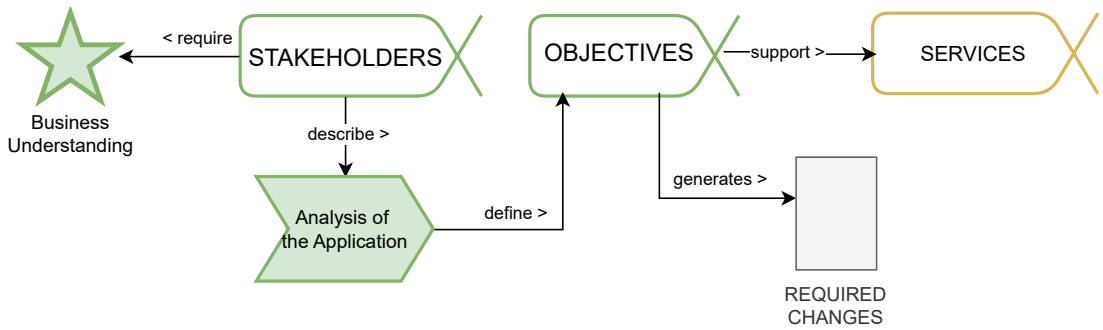

Figure 5.10: Practice for analysis of Eagle application.

From the practice in figure 5.10, you can generate the practice in figure 5.11. The practice shows the process of designing the architecture of a module. Based on

the Objectives, the Architectural Team is responsible for carrying out the design and defining the Architectural Changes of that module, as well as having Management and Architecting skills. This new design generates a product that is the documentation with the required changes and which provides the basis for generating the Services, which require Microservice Instruction skills to build the microservices.

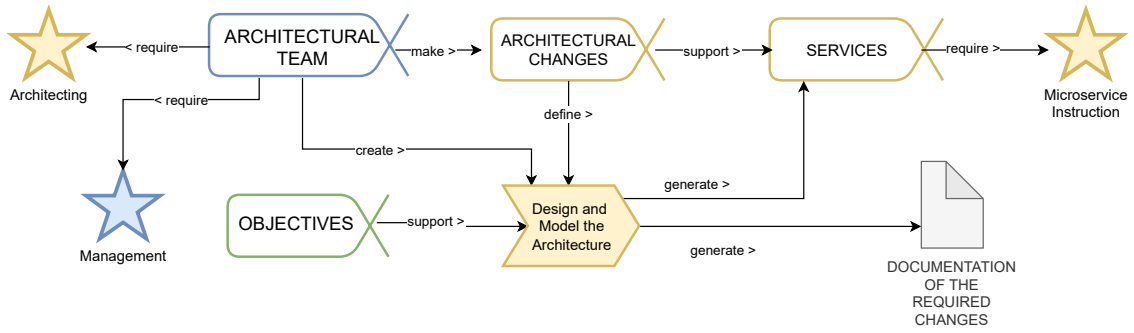

Figure 5.11: Practice for the design of the migration of Eagle application.

The next practice, figure 5.13, shows the configuration of the environment to make all the microservice implementations possible and thus validate the entire application. Once the environment is available, it is possible to move on to the Integrate and Test the Module activity, connecting the microservices with the entire structure proposed for the final application. The team responsible for making this environment available must have Management skills, making it possible to modify the environment when necessary and thus help the solution move forward.

Figure 5.12: Practice for the definition of environments of the microservices.

The last practice defined is in figure 5.12, which shows the microservice development process. Based on the divisions of activities assigned by the architectural team and the documentation of how the architecture of the module should be created, the Development Team receives the activities and starts to implement the microservice, thus

requiring Refactoring skills. Through its Management skills, the Development Team can monitor the progress of the solution and deliver a final product.

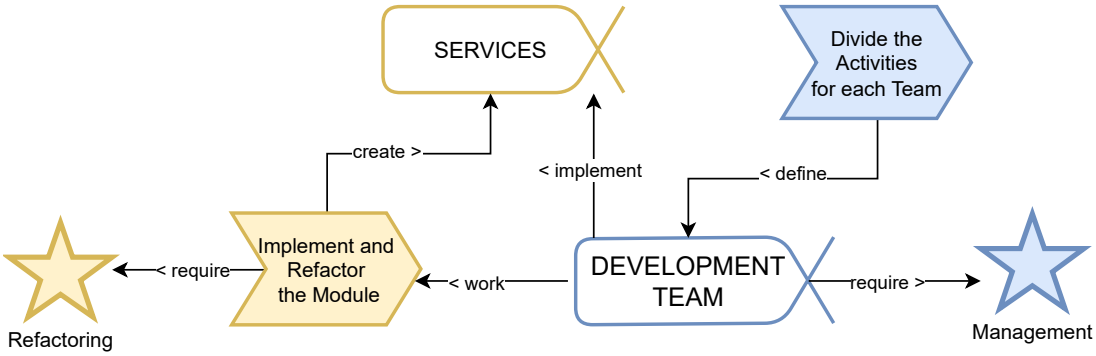

Figure 5.13: Practice for the creation of the microservices.

The practices shown here are an example of how the definition of activities can be carried out, from tasks detailing a larger overview of the microservice development process (figure 5.11), to practices focusing on a set of smaller activities, as exemplified in figure 5.13.

5.2 Survey

The M3K proposes to auxiliary the industry during monolithic migration to a microservice-based architecture. With this proposal, a survey is provided focusing on understanding the highlight points in M3K over a migration process.

The research was carried out in collaboration with the Eagle migration team at Synchro, which includes developers, a software architect, and a project lead. Since M3K was inspired by the migration process undertaken in the Eagle project, the team was invited to evaluate the instantiation process (see Section 5.1) and assess the effectiveness of M3K in supporting architectural migration efforts from a monolithic to a microservice architecture.

ID	Education	Industry experience	Role
R1	Graduate	1–3 years	Software developer
R2	PhD	> 10 years	Project leader
R3	Postgraduate	> 10	Software architect
R4	Graduate	1–3 years	Software developer
R5	PhD	5–10 years	Software developer
R6	PhD	1–3 years	Software developer

Table 5.2: Respondents' profile

5.2.1 Profile of the professionals

The survey is composed of 6 professionals in the area (table 5.2), 2 graduate, 3 PhD, and 1 postgraduate. With 3 of them working between 1 and 3 years, 1 between 5 and 10 years, and 2 more than 10 years. Over the distribution of the jobs, 4 are software developers, 1 is a software architect, and 1 project leader. The interviewees were Synchro employees and UFRN students who migrated from the original Eagle architecture to the Microservices architecture.

Over the knowledge and experience in microservices, 2 people defined them with a high level of knowledge, 1 with good knowledge, and the other, 3, have a basic knowledge of microservices, working or knowing what are microservices. All of the people had already worked in a migration process to a microservice-based architecture.

5.2.2 Migration process

Aiming to understand which are the main activities required to make a migration process, we requested them to specify which activities were performed in their experience. The most common activities were to understand the functionalities of the system and replicate them, these were a consensus among the participants, even those that joined the process after the start.

Considering those participants that have been performing the migration process since the beginning of the process, activities such as *plan the microservice* (considering

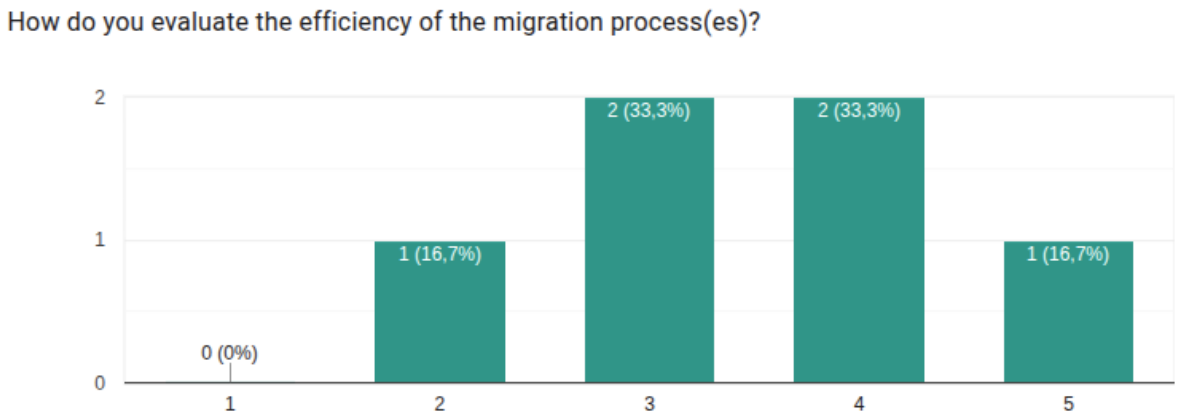

Figure 5.14: Efficiency of migration process(es) from 1 (not efficient) to 5 (very efficient)

goals, performance, multi-tenancy, etc.), implementing the microservice, evaluating the performance and adjusting to gather the desired results are some example of activities. However some pointed out that it is much more than just the division of the architecture into multiple services, it is an opportunity to change processes, and fluxes and make use of better technologies. The efficiency of the migration process adopted was considered on a scale from 1 (very inefficient) to 5 (very efficient), with 2 judged as 3 and 4, while 1 considered as very efficient and 1 judged with a 2, presented on figure 5.14.

Over the time that the processes were performed, there was no consensus on how much time was required, some took 3 to 4 months depending on the functionality, 6 months, 1 year, or even more than 3 years to get a stable version, but without any perspective to finish, considering the maintenance of the microservices over time.

The activities performed by the participants in the migration processes were mainly analysis and modeling of the architecture, developing the microservices, and applying CI/CD practices in the projects. These processes were performed in an ad-hoc manner, just analyzing the functionalities of the microservice, prioritizing the services defined by the company, and applying a Strangler Fig pattern.

The strengths of the experience were the focus on corrections of old implementations and, the division of the responsibilities in the architecture. The definition

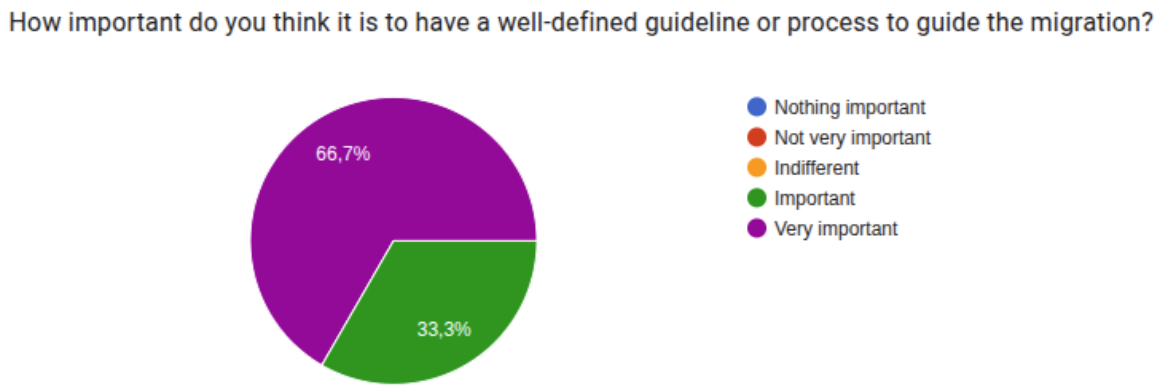

Figure 5.15: Importance of a well-defined guideline or process to guide the migration

of an architecture pattern to be followed during all the development, high iteration between all the participants of the migration, usage of more updated technologies, and the development of an application since the beginning complete the highlight points. The weakness is that some changes did not correct some problems of the application, the division of the monolithic in many microservices can create an overhead. The required changes during the migration process by the clients increase the time of the migration, which makes it more complex.

Concerning strengths and weakness, the migration process is affected by the replication of the monolithic functionality in the microservice, changes requested by clients during the migration process, standards that must be applied by the developers in the application, adoption of new technologies, and outdated databases and systems in the client side.

A consensus among all the participants is the importance of a guideline or well-defined process to guide the migration, presented in figure 5.15. This lack increases the complexity of the microservice, increasing costs and maintenance in the architecture. Difficult onboarding of new technical collaborators, a lot of rework, and delays in the migration are some other examples pointed out.

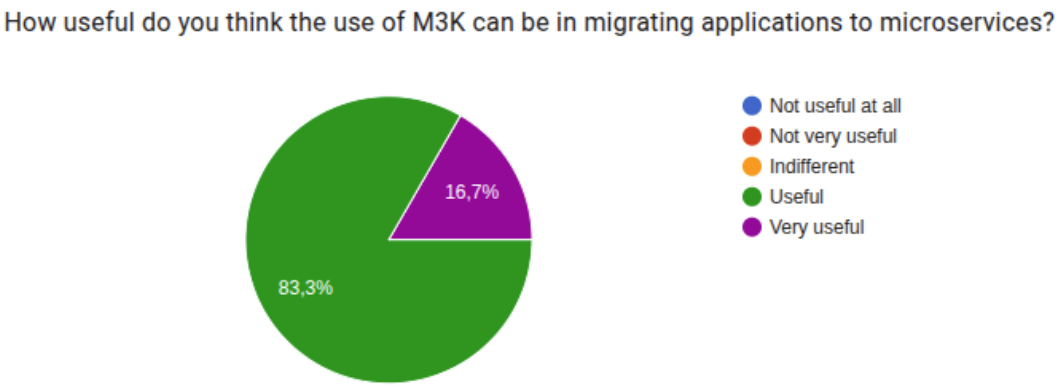

Figure 5.16: How useful it is the M3K to the migration of applications to microservice

5.2.3 M3K analysis

This section presents a short version of the application of M3K presented in section 5.1. We present an approach to apply M3K (Medeiros et al., 2023) to define a migration process and all its characteristics.

The roles and competencies defined by the M3K are considered as important to the participants, concerning that each one will know exactly which activities they must perform during the migration. Related to the activities proposed, they are defined as essentials to every migration process. This approach results in a useful tool to auxiliary in the migration process definition.

Concerning the M3K applicability, most participants would adopt M3K to assist in the migration process, figure 5.16. While concerned about the impact of the usage of M3K, all the participants define that the application of the metaprocess must provide a positive impact in the migration process. All the participants agree that *M3K helps migrate legacy applications to an architecture based on microservice.*

Over the strengths of M3K, the participants describe steps, roles, competencies, and well-defined activities, helping in the execution of the migration process. Division of responsibilities in the migration process, the definition of a structure to the architecture, and a guideline to be followed. The weakness is related to the lack of flexibility during the

migration to allow changes, such as changes in priorities of activities, context, restrictions in attributes, changes in team, and deadlines.

Some consideration of the M3K made by a participant focuses on the application of a *feedback loop*, in which each Activity Space could be able to calibrate the steps to be followed, monitoring the desired priorities, restrictions, and changes in the context. Add a new activity space concerning the evolution of the architecture or each one of the microservices, concerning essentially maintenance. And takes into account adjusting the metaprocess to support a more flexible environment, which changes are required during the migration process definition.

5.2.4 Discussion

The results obtained from this survey represent an initial exploration of developers' perspectives on the migration of monolithic applications to microservices. Among the limitations are the small number of participants and the fact that all interviewees were from the same company. As such, the findings cannot be generalized to broader contexts. However, they offer valuable preliminary insights into the objectives of this study and help establish a foundation for future research on software architecture migration.

6 Final Remarks

The migration process from a monolithic architecture to one based on microservices can be a challenge when considering the decisions to be taken so that the processes present in the literature are limited to presenting only one view of the application. M3K seeks to help developers and companies idealize a migration process, focusing the process on the real needs of the company. When analyzed from the point of view of implementation, the metaprocess proposed in this work presents several fronts to be addressed, in addition to refactoring and actual migration processes. Among the points addressed in addition to the Solution concern area, we defined a context for the system users' view, through the Customer concern area; and the vision of the means of work, through the Endeavor concern area.

M3K differs from several approaches presented in the literature by analyzing the entire process that involves the development of a migration process between monolithic architectures for microservices. From assigning roles to defining activities based on Activity Spaces and executing them, not concentrating all the efforts only on the last step.

Carrying out a well-documented migration and addressing all the needs for a quality migration allows the possible challenges to be encountered when performed in an ad-hoc manner, to be minimized, or even absorved. M3K proposes to deliver quality to the project, especially when the defined practices are structured to perform a product that delivers value to customers. For companies, the possibility of understanding

migration becomes a useful tool, mainly because after defining the process, practices can be restructured or even reused. For developers, the definition of practices helps in the execution of the necessary actions to be performed to obtain the final product.

A metaprocess for migrating monolithic architecture to microservices is pointed out by developers as important to make the migration, obtained by the survey presented in this work. Developers are open to test new methodologies to migrate monolithic architectures, being supported by a well-defined methodology. As result, we propose this methodology defining the M3K.

6.1 Reviewing the contribution

This work was guided by the investigation of three main Research Questions (RQs), each aiming to explore a distinct aspect of the migration from monolithic applications to microservice-based architectures:

RQ1: What activities are present in a migration process from monolithic applications to microservice-based architecture?

To address RQ1, a comprehensive literature review was conducted, examining both academic and industrial sources related to the migration process. From this review, a set of relevant activities was identified and cataloged, with an emphasis on understanding their context and how they were applied in practice.

However, the activities described in the literature were not always sufficiently detailed to clarify their specific objectives. Therefore, a deeper analysis was necessary to interpret how these activities were executed and how they connected to subsequent steps in the migration process. The methodology for identifying and organizing these activities is detailed in the

The activities identified through this investigation formed the foundation for answering the second research question.

RQ2: How to define a migration metaprocess from monolithic architecture to microservice-based architecture using the Essence standard?

The construction of the proposed metaprocess required a thorough generalization of the activities identified in RQ1. Each activity was abstracted to a level that allows for application across various domains, independent of specific technologies or business contexts.

Following this generalization, the activities were mapped onto the Essence framework. This began with the identification of relevant *Alphas*, followed by the definition of corresponding *Activity Spaces*, and finally, the *Competencies* required to carry out those activities.

This structured mapping enabled the development of the M3K metaprocess, providing a reusable framework for guiding migrations from monolithic to microservices architectures.

RQ3: How to validate the proposed migration metaprocess?

To validate M3K, it was applied in the context of a real-world software migration project involving a tax application. This case study demonstrates how the metaprocess can be instantiated to create a concrete, actionable migration plan tailored to a specific domain.

In addition to this practical application, validation was also carried out through a survey with developers who had prior experience in similar migration efforts. These insights helped assess the relevance, completeness, and utility of the proposed metaprocess, confirming its potential value in real migration scenarios.

6.2 Limitations of the work

Defining a generic metaprocess for migrating monolithic systems to microservices presents inherent challenges, primarily due to the variability in requirements across

different application domains. While this metaprocess provides a structured foundation, its practical application demands instantiation and adaptation to the specific context in which it is implemented.

One key limitation lies in the necessity to tailor the metaprocess to each domain. Certain steps outlined in this work may not be universally relevant, requiring modification or omission depending on the organization's specific needs. Thus, it becomes the responsibility of software architects to determine which elements of the metaprocess should be applied, adapted, or excluded to align with their domain's constraints and objectives.

Another significant limitation is related to time-to-market pressures. In many organizations, migration timelines are constrained by business demands for rapid delivery. The software migration process, particularly for legacy systems, can be time-intensive, as it requires validation of existing functionalities to ensure continuity. Although the metaprocess proposed here aims to streamline decision-making and migration planning, its comprehensive nature may be seen as a barrier when immediate results are prioritized.

Additionally, while the metaprocess was validated in a simulated scenario, broader empirical studies are still needed. This type of research demands not only willing organizations but also suitable applications that are in the pre-migration phase. For optimal effectiveness, the metaprocess should be applied before the start of the migration effort, making timely access to appropriate case studies a challenge for future evaluation.

Finally, although this work focuses on the technical and procedural aspects of migration, it is essential to emphasize that not all problems in monolithic systems are best addressed through architectural migration. The decision to transition to microservices should be made based on a thorough assessment of system needs and organizational goals. In some cases, retaining or refactoring the existing monolithic architecture may be a more effective solution.

6.3 Future Works

This section outlines potential directions for future research that aim to extend and enhance the contributions of this study:

Alignment of Alpha States and Activities with the migration process and continuous delivery: One promising area for future work is the formal alignment of Essence's Alpha States and Activities with continuous delivery practices. While Alpha States help track the progress of software development elements, Activities validate the progression between states. Integrating these components with continuous delivery pipelines could provide organizations with more effective tools for monitoring and controlling the migration process. This alignment would facilitate real-time feedback on which stages have been completed and what remains, thereby improving migration oversight and reducing process risk.

Automation of migration processes: With recent advances in software engineering powered by Large Language Models (LLMs), there is growing potential to automate aspects of the migration process. For instance, data extracted from databases and system documentation could be used to generate migration plans or code transformation suggestions. However, such automation raises concerns about data privacy and security, which could be addressed through techniques such as Federated Learning. Future work in this area would focus on developing an automated metaprocess that leverages machine learning to assist in architectural transitions.

Expanding the M3K: While M3K has been tailored specifically for migrations from monolithic systems to microservices, the underlying methodology can potentially be generalized for use in other architectural transformations. Future research could explore the expansion of M3K to support migrations between a broader range of architectural styles, such as service-oriented architectures (SOA), serverless architectures, or event-driven systems. Furthermore, incorporating architectural patterns and quality attributes

as first-class elements in the metaprocess would enhance its adaptability and ensure higher-quality outcomes across varied scenarios.

Bibliography

Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. (2023). Decomposition of monolith applications into microservices architectures: A systematic review. *IEEE Transactions on Software Engineering*, *49*(8), 4213–4242. https://doi.org/10.1109/TSE.2023.3287297

Ahmad, A., & Babar, M. A. (2014). A framework for architecture-driven migration of legacy systems to cloud-enabled software. *Proceedings of the WICSA 2014 Companion Volume*. https://doi.org/10.1145/2578128.2578232

Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From monolithic systems to microservices: An assessment framework. *Information and Software Technology*, *137*, 106600. https://doi.org/https://doi.org/10.1016/j.infsof.2021.106600

Ayas, H. M., Leitner, P., & Hebig, R. (2021). Facing the giant: A grounded theory study of decision-making in microservices migrations. *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. https://doi.org/10.1145/3475716.3475792

Batista, C., Proenca, B., Cavalcante, E., Batista, T., Morais, F., & Medeiros, H. (2022). Towards a multi-tenant microservice architecture: An industrial experience, 553–562. https://doi.org/10.1109/COMPSAC54236.2022.00100

Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V., & Sadovykh, A. (2020). *Microservices: Science and engineering* (1st). Springer Publishing Company, Incorporated.

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., & Mazzara, M. (2018). From monolithic to microservices: An experience report from the banking domain. *IEEE Software*, *35*(3), 50–55. https://doi.org/10.1109/MS.2018.2141026

Di Francesco, P., Lago, P., & Malavolta, I. (2018). Migrating towards microservice architectures: An industrial survey. *2018 IEEE International Conference on Software Architecture (ICSA)*, 29–2909. https://doi.org/10.1109/ICSA.2018.00012

Doležal, J., & Buchalcevová, A. (2022). Migration from monolithic to microservice architecture: Research of impacts on agility. *IDIMT-2022 Digitalization of society, business and management in a pandemic*.

Elvesæter, B., Benguria, G., & Ilieva, S. (2013). A comparison of the essence 1.0 and spem 2.0 specifications for software engineering methods. *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering*. https://doi.org/10.1145/2489833.2489835

Fachat, A. (2019). Challenges and benefits of the microservice architectural style, part 1. Available at: https://developer.ibm.com/articles/challenges-and-benefits-of-the-microservice-architectural-style-part-1/

Fonseca, I., Fritsch, L., Bernardino, M., & Basso, F. (2021). Bpmn, spem e essence no contexto da modelagem de processos de software: Uma revisão sistemática da literatura. *Anais da V Escola Regional de Engenharia de Software*, 119–128. https://doi.org/10.5753/eres.2021.18457

Garriga, M. (2018). Towards a taxonomy of microservices architectures. In A. Cerone & M. Roveri (Eds.), *Software engineering and formal methods* (pp. 203–218). Springer International Publishing.

Hassan, S., Bahsoon, R., & Kazman, R. (2020). Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience*, *50*(9), 1651–1681. https://doi.org/https://doi.org/10.1002/spe.2869

Jacobson, I., Lawson, H., Ng, P., McMahon, P., & Goedicke, M. (2019). *The essentials of modern software engineering: Free the practices from the method prisons!* Association for Computing Machinery; Morgan & Claypool Publishers. Available at: https://books.google.com.br/books?id=JOOkDwAAQBAJ

Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., & Lidma, S. (2012). The Essence of Software Engineering: The SEMAT Kernel. *Communications of the ACM*, *55*(12), 42–49. https://doi.org/10.1145/2380656.2380670

Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., & Lidma, S. (2013). *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley.

Kolny, M. (2023). Scaling up the prime video audio/video monitoring service and reducing costs by 90%. Available at: https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90

Koskinen, J., Ahonen, J., Sivula, H., Tilus, T., Lintinen, H., & Kankaanpaa, I. (2005). Software modernization decision criteria: An empirical study. *Ninth European Conference on Software Maintenance and Reengineering*, 324–331. https://doi.org/10.1109/CSMR.2005.50

Kyryk, M., Tymchenko, O., Pleskanka, N., & Pleskanka, M. (2022). Methods and process of service migration from monolithic architecture to microservices. *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 553–558. https://doi.org/10.1109/TCSET55632.2022.9767055

Martínez Saucedo, A., Rodríguez, G., Gomes Rocha, F., & dos Santos, R. P. (2025). Migration of monolithic systems to microservices: A systematic mapping study. *Information and Software Technology*, *177*, 107590. https://doi.org/https://doi.org/10.1016/j.infsof.2024.107590

Medeiros, H., Batista, T., & Cavalcante, E. (2023). On a metaprocess for microservice migration. *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, 116–121. https://doi.org/10.1145/3613372.3613382

Michael Ayas, H., Hebig, R., & Leitner, P. (2024). An empirical investigation on the competences and roles of practitioners in microservices-based architectures. *Journal of Systems and Software*, *213*, 112055. https://doi.org/https://doi.org/10.1016/j.jss.2024.112055

Michael Ayas, H., Leitner, P., & Hebig, R. (2023). An empirical study of the systemic and technical migration towards microservices. *Empirical Softw. Engg.*, *28*(4). https://doi.org/10.1007/s10664-023-10308-9

Mishra, M., Kunde, S., & Nambiar, M. (2018). Cracking the monolith: Challenges in data transitioning to cloud native architectures. *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. https://doi.org/10.1145/3241403.3241440

Newman, S. (2019). *Monolith to microservices: Evolutionary patterns to transform your monolith*. O'Reilly Media, Incorporated.

Newman, S. (2015). *Building microservices: Designing fine-grained systems* (1st). O'Reilly Media.

Ng, P.-W., & Huang, S. (2013). Essence: A framework to help bridge the gap between software engineering education and industry needs. *Software Engineering Education Conference, Proceedings*, 304–308. https://doi.org/10.1109/CSEET.2013.6595266

Nguyen, M., & Conradi, R. (1994). Classification of meta-processes and their models. *Proceedings of the Third International Conference on the Software Process. Applying the Software Process*, 167–175. https://doi.org/10.1109/SPCON.1994.344414

Nordli, E. T., Haugeland, S. G., Nguyen, P. H., Song, H., & Chauvel, F. (2023). Migrating monoliths to cloud-native microservices for customizable saas. *Information and Software Technology*, *160*. https://doi.org/10.1016/j.infsof.2023.107230

Object Management Group. (2018). *Essence – Kernel and Language for Software Engineering Methods, version 1.2*. OMG. USA. Available at: https://www.omg.org/spec/Essence/1.2/About-Essence

Ozkaya, M. (2023). Microservices architecture for enterprise large-scaled application (Medium.com, Ed.) [[Online; posted 17-February-2023]]. Available at: https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-for-enterprise-large-scaled-application-825436c9a78a

Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A rapid review. *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, 1–7. https://doi.org/10.1109/SCCC49216.2019.8966423

Richards, M. (2015). *Microservices vs. service-oriented architecture*. O'Reilly Media. Available at: https://books.google.com.br/books?id=Bd5mAQAACAAJ

Richardson, C. (2018). *Microservices patterns: With examples in java*. Manning. Available at: https://books.google.com.br/books?id=UeK1swEACAAJ

Rud, A. (2023). Why and how netflix, amazon, and uber migrated to microservices: Learn from their experience (H. Enterprise, Ed.) [[Online; posted 24-July-2019]]. Available at: https://www.hys-enterprise.com/blog/why-and-how-netflix-amazon-and-uber-migrated-to-microservices-learn-from-their-experience/

Sahay, A., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2023). Analyzing business process management capabilities of low-code development platforms. *Software: Practice and Experience*, *53*(4), 1036–1060. https://doi.org/https://doi.org/10.1002/spe.3177

Systems process engineering meta-model specification. (2008). Available at: https://www.omg.org/spec/SPEM/2.0/PDF

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, *4*(5), 22–32. https://doi.org/10.1109/MCC.2017.4250931

Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*.

Volynsky, E., Mehmed, M., & Krusche, S. (2022). Architect: A framework for the migration to microservices. *2022 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, 71–76. https://doi.org/10.1109/iCCECE55162.2022.9875096

Wagner, C. (2014). *Model-driven software migration: A methodology reengineering, recovery and modernization of legacy systems*. Springer Vieweg.

Wolfart, D., Assunção, W. K. G., da Silva, I. F., Domingos, D. C. P., Schmeing, E., Villaca, G. L. D., & Paza, D. d. N. (2021). Modernizing legacy systems with microservices: A roadmap. *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, 149–159. https://doi.org/10.1145/3463274.3463334