A Systematic Framework To Enhance Reusability In Microservice Architecture

Mehdi AIT SAID[1], Lahcen BELOUADDANE[1], Soukaina MIHI[1] and Abdellah EZZATI[1]

[1]*Hassan first University, Settat, Morocco*

Abstract: In the ever-evolving field of software engineering, the concepts of software reuse and Microservices Architecture (MSA) have emerged as fundamental pillars reshaping development methodologies and project outcomes. Recognizing the transformative potential of these paradigms, our research endeavors to delve deeper into their intersection, particularly focusing on enriching reusability practices within the context of MSA. To achieve this objective, we have identified and meticulously addressed five challenges that often impede optimal reusability in MSA environments: Code Duplication, Technology Heterogeneity, Service Boundaries, Versioning, and Decision-Making. Leveraging insights gleaned from practical experiences, we propose the Reusable Microservices Framework (RMF), a robust and comprehensive development process meticulously crafted to systematically tackle these challenges. Developed in close collaboration with MSA practitioners deeply invested in advancing reusability practices, the RMF embodies a synthesis of expert recommendations and industry best practices. Our validation process encompasses a multifaceted approach, ranging from a simulated environment to real-world implementation, including the adoption of the RMF within a software company. Through rigorous validation exercises, our findings unequivocally demonstrate the transformative potential of the RMF, showcasing significant enhancements in reusability metrics that exceed expectations by over threefold. By offering actionable insights and a practical framework honed through empirical validation, our study presents a compelling roadmap for harnessing the power of reusability to unlock the full potential of MSA.

Keywords: Microservice Architecture, RMF, DDD, MDD, Reusability

1. Introduction

Software reuse has been a central theme in software engineering, with continuous efforts to enhance its methods, techniques, and cost models. The evolution of reuse practices, often integrated into higher-level software engineering processes, has led to substantial research contributions over the years, Capilla et al. in 2019 [1] provided a comprehensive overview of the state of software reuse, highlighting emerging trends and opportunities in the face of an ever-changing technological field. The shift toward advanced programming techniques and the adoption of reuse methods have paved the way for new research areas and forms of reuse. Capilla et al.'s [1] exploration of the recent history of software reuse revealed the emergence of open data, feature models, and other novel opportunities beyond traditional software components. Importantly, their findings identified both opportunities and challenges in improving reusability, particularly in the context of modern software architectures.

Building upon the insights provided by Capilla et al., our work delves into the specific challenges and opportunities for improving reusability within the realm of Microservices Architecture (MSA). To achieve this, we conducted an industrial study involving 28 microservices practitioners, aiming to identify MSA reusability challenges and practices. In this paper, we present a framework developed based on the expertise of MSA practitioners and guided by the opportunities and challenges identified by Capilla et al.[1]

MSA has emerged as a transformative paradigm in software engineering [2], offering a modular and scalable approach to building complex applications [3]. The fundamental tenets of MSA involve breaking down monolithic applications into small, independent services that communicate seamlessly [4]. One of the key advantages touted by MSA enthusiasts is its inherent potential for enhancing software reusability [5]. In recent years, the software engineering community has witnessed a shift toward MSA due to its ability to facilitate agility, scalability, and ease of deployment[6]. The decentralized nature of microservices allows developers to create, deploy, and update individual services independently, fostering a culture of continuous integration and delivery [7]. As a result, MSA is often lauded for its potential to promote software reusability by enabling the creation of reusable and independently deployable microservices [8].

The primary objective of our research is to contribute to the ongoing discourse on software reuse, focusing on the unique challenges posed by MSA. By leveraging the insights from Capilla et al.'s work [1], we aim to not only validate their findings in a real-world setting but also to extend the understanding of reusability within the dynamic context of microservices.

By conducting an industrial inquiry through a series of interviews with 28 MSA practitioners, we identified five challenges: Code Duplication, Technology Heterogeneity, Service Boundaries, Versioning, and Decision-Making. Delving into the expert perspectives, we gained insights into how these challenges are navigated. Subsequently, leveraging these findings, we developed the Reusable Microservices Framework (RMF). To validate its efficacy, the RMF was implemented and adopted in a real-world software company over a span of two years. This practical validation served to affirm the framework's applicability and effectiveness in addressing the identified challenges within the dynamic context of MSA.

The remainder of this paper is organized as follows: Section 3 presents the background of microservices and reusability, followed by an exploration of related work in Section 4. Section 5 outlines our research methodology, and Section 6 presents challenges and best practices in microservice reusability. Section 7 details the proposed framework, and Section 8 covers framework validation in a real-world setting, presenting results with discussion. Finally, Section 9 concludes by summarizing key findings, contributions, and implications of the RMF framework, offering insights into future research and practical applications.

2. Background

MSA represents a transformative paradigm in software engineering [2], offering a modular and decentralized approach to designing and deploying applications [9] [10]. In contrast to traditional monolithic architectures, where applications are built as a single, tightly integrated unit [11], MSA decomposes applications into a collection of small, independent services [12], each microservice operates as a self-contained entity, with its own database and well-defined communication mechanisms with other services[13]. The shift from monolithic to MSA is driven by several factors. In a monolithic architecture, a change to one part of the application often necessitates rebuilding and redeploying the entire system [14]. This monolithic approach can hinder agility, scalability, and continuous integration [15]. MSA, on the other hand, promotes flexibility and agility by allowing developers to independently build, deploy, and update individual microservices without affecting the entire system [12] [5]. This modular structure enables organizations to embrace a DevOps culture, fostering faster development cycles and improved responsiveness to changing requirements. Comparing microservices with monolithic architectures unveils distinct advantages that contribute to the facilitation of software reusability:

Decentralization and Independence:

- Monolithic: In a monolithic architecture, components are tightly coupled, making it challenging to reuse specific functionalities independently [16].
- MSA: Microservices operate independently, allowing developers to create small, focused services with well-defined functionalities. This independence facilitates the creation of reusable microservices that can be employed across various applications [17].

Granularity of Services:

- Monolithic: Monolithic applications often consist of large and complex codebases, making it cumbersome to extract and reuse specific functionalities [18].
- MSA: The modular nature of microservices allows for the creation of fine-grained services. Developers can focus on creating small, specialized microservices that encapsulate specific features, promoting easier reuse in diverse contexts [19].

Technology Heterogeneity:

- Monolithic: Technology choices in a monolithic architecture are constrained by the overarching technology stack [12].
- MSA: Each microservice can be developed using different technologies, enabling teams to choose the most suitable technology for a specific service [20]. This flexibility contributes to the adaptability and reusability of microservices across diverse technological environments[21].

Microservices, with their decentralized and modular structure, inherently lend themselves to enhanced software reusability [5]. The encapsulation of specific functionalities within independent microservices allows for the creation of reusable components that can be seamlessly integrated into various applications. The agility provided by microservices, coupled with their independence, fosters a culture of continuous integration and delivery, further supporting the rapid deployment and reuse of software assets [22]. Remarkably, MSA aligns harmoniously with the 3C model [23], a pivotal reference model in the realm of software reuse. The 3C model, rooted in the history of software engineering's 30-year journey, delineates three fundamental dimensions: concept, content, and context (Figure 1). In the context of MSA, the alignment with the 3C model becomes apparent. Firstly, the concept in the 3C model, representing a reusable component's specification or "what it is," resonates with the individual microservices in MSA. Each microservice encapsulates a specific concept, defining its functionality and purpose. Secondly, the content dimension of the 3C model, emphasizing "how it works," finds resonance in the modular and independent nature of microservices. MSA allows developers to separate the specification from the implementation, akin to the 3C model's vision. For example, different microservices may implement similar concepts but utilize distinct technologies or implementation approaches, providing a versatile approach to content. Lastly, the context in the 3C model, indicating "how it is used," draws parallels with the decentralized nature of microservices.

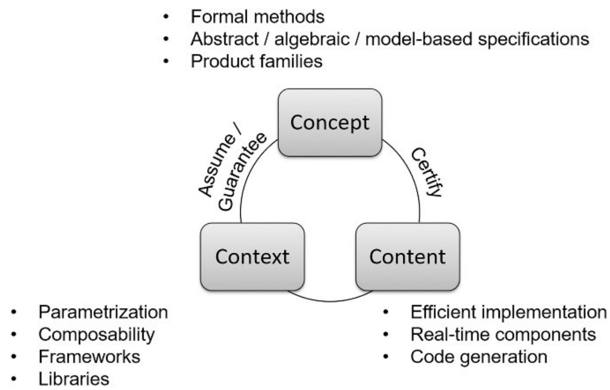

Figure 1. The 3C model contextualizing software reuse [1]

3. Related work

Within the MSA field, numerous studies have delved into distinct facets, tackling challenges and probing opportunities in reusability contexts. This section meticulously examines relevant works that provide key insights into the extraction, implementation, and transformation processes associated with enhancing reusability within the framework of microservices adoption.

In 2019, Carvalho et al.[24] conducted an exploratory study focusing on the extraction of reusable microservices from legacy systems, with a particular emphasis on systems featuring variability. The study revealed that variability is a key criterion in structuring microservices, and the simplicity of mechanisms used to implement variability plays a crucial role. This research not only contributes essential insights into the pragmatic facets of transitioning systems to an MSA but also sheds light on the role of reusability, emphasizing its centrality in the extraction process.

In 2019, Silva et al.[25] proposed Microservice4EHR, a cloud tool designed to dynamically generate reusable components from existing software artifacts in healthcare applications. The study demonstrated that employing the MSA enhances reusability in Health applications. Microservice4EHR offers a tangible solution for improving software reusability in the healthcare domain, aligning with the industry's unique requirements.

In 2023, Hamed explored the reusability of legacy software using a microservices-based architecture [26], focusing on the transformation of an online exam system. The study employed feature-driven microservice-specific transformation rules, prioritizing performance, maintainability, scalability, and testability. This work contributes insights into the efficient re-architecture and reengineering of legacy enterprise systems using microservices.

A. Comparison

These studies collectively enhance our understanding of microservices adoption, each addressing specific areas or domains such as migration, healthcare, legacy systems, and DevOps practices. A notable common thread among them is the emphasis on reusability as a pivotal aspect of microservices impact. In exploring diverse contexts, these studies underscore the significance of reusability in the microservices landscape. They illuminate how microservices, when implemented thoughtfully, become reusable components, contributing to heightened efficiency, code modularity, and scalability. Whether dealing with the intricacies of healthcare applications or navigating the challenges of legacy system migration, the studies consistently highlight the positive influence of microservices on fostering a culture of reusability.

In contrast to these domain-specific studies, our framework adopts a holistic perspective, prioritizing reusability as a central tenet. By offering a versatile development process, our framework transcends the limitations of context-specific solutions. It is designed to be universally applicable, aligning seamlessly with any agile development style. In doing so, our framework positions reusability as a core principle in microservices adoption, acknowledging its transformative impact on software development across diverse scenarios.

4. Methodology

Our research design aligns with the principles of Design Science (DS) proposed by Peffers et al.[27] and the Design Science Research Methodology (DSRM) proposed by March et al [28]. DS offers an approach to developing models, methods, and implementations with the intent of serving human purposes, and DSRM adapts this philosophy to the field of information systems research.

A. First phase: industrial inquiry

1) Approach

An industrial inquiry coupled with a qualitative research methodology was employed to gain profound insights into reusability challenges within MSA.

2) Objectives

- RQ1: Identify real-world challenges of reusability in an MSA-based environment.
- RQ2: Understand how experts in the field address these challenges.

3) Participants

A total of 28 practitioners, each possessing a minimum of 5 years of experience in MSA. Participants were selected from 24 software companies and 3 countries (Morocco, France, and USA), ranging from startups to industry giants such as Oracle and IBM. The selection criteria also considered the practitioners' explicit interest or existing initiatives related to reusing microservices within their respective organizations.

4) Data collection

Comprehensive semi-structured interviews were conducted over a period of two months, commencing in October 2020. The chosen qualitative approach, as outlined by

Brinkmann & Kvale [29], facilitated in-depth exploration of practitioners' experiences and perspectives. The interview questions can be found in Table A in the data repository, providing a detailed reference for the research methodology.

B. Second phase: Best Practices Formulation

Building on the insights garnered from the Industrial Inquiry, the study transitioned to articulating best practices tailored to address specific challenges. The outcome was a systematic framework equipped with stringent guidelines to augment reusability in MSA.

C. Third Phase: Framework Validation in a Real-World Setting

To validate and assess the effectiveness of our framework, we implemented it in a real-world software company over a span of two years. The evaluation included measuring two indicators, adoption cost and the percentage of code reused for each project, and comparing it with the previous time without our framework. The data collection process involved gathering quantitative indicators, supplemented by a series of interviews with stockholders and developers who actively engaged with our framework. These interviews aimed to elicit valuable insights and feedback, providing a nuanced understanding for ongoing refinements and improvements in our framework.

D. Data repository of tables

Due to the extensive volume of data accompanying our research paper, it has been made available for public access via the Zenodo[1] repository, accessible through the following link: https://zenodo.org/records/10849454. The dataset comprises three essential tables integral to our study. Table A presents the comprehensive set of interview questions employed to explore the challenges related to reusability in MSA. Table B outlines the microservices identified for the initial build of the Microservices Hub (Section 6.A.1). Table C encompasses the evaluation questions utilized to assess various aspects of the proposed framework, offering a structured approach to measure its effectiveness and user satisfaction. Access to this data repository ensures transparency and reproducibility, enabling researchers and practitioners alike to delve deeper into our study's methodology and findings.

5. Challenges

In our exploration of the practitioner's interviews, we've distilled insights from practitioners who identified and grappled with five prominent challenges. As a result of their experiences and reflections, the challenges of Code Duplication, Technology Heterogeneity, Service Boundaries, Versioning, and Decision-making emerged as focal points influencing the field of reusability within MSA. The frequency of mentions for each challenge by the practitioners is visually encapsulated in Figure 2, offering a snapshot of the prevalent concerns drawn directly from our interview results.

[1] https://zenodo.org

A. Code duplication

1) Challenge

Code duplication has been an ancient challenge in software engineering, having persisted over time as an enduring concern. This challenge refers to the repetition of code segments or functionalities within a software system. Code duplication can lead to various problems, including increased development and maintenance efforts, potential inconsistencies, and difficulties in ensuring updates and changes are applied uniformly. While code duplication has been recognized as a general challenge in software engineering, its manifestation within the context of MSA introduces unique considerations. In MSA, the challenge of code duplication takes on a distinctive character due to the decentralized and modular nature of microservices. MSA encourages the development of small, independent services, each responsible for a specific functionality. This autonomy granted to microservices can potentially result in unintended duplications across different microservices or even in separate projects developed by different teams. The distributed and independent nature of microservices in MSA introduces complexities that amplify the impact of code duplication, making it a more intricate challenge to address within this architectural paradigm. A concrete example illustrating the code duplication challenge was shared by a practitioner involved in our study: "In my daily routine, I used to check in with my developers to see what they were working on. Before start using MSA, their responses were clear and distinct, like "I'm working on the signup feature" or "I'm dealing with OAuth." These responses made sense to me because it seemed like they were tackling different tasks. However, with MSA in place, their answers changed to "I'm working on the user management system." It became evident that both developers were essentially working on the same thing. Previously, in a non-MSA setting, they might have been in different phases of the project timeline. For instance, the first developer might finish implementing OAuth after completing the signup feature, while the second developer might have already built the signup functionality. The shift to MSA blurred these distinctions, making it apparent that their efforts were converging on a shared goal within the broader user management system." Quote 1

2) Addressing the challenge

Practitioners advocate for a centralized microservices registry within MSA organizations to address code duplication. This registry serves as a shared catalog consolidating information on all microservices developed across teams. Its purpose is to provide a repository for developers to discover and access existing microservices, preventing unintentional code duplication. By facilitating awareness and collaboration, the registry promotes efficient reuse of services, fostering a culture of shared knowledge within the organization.

B. Technology Heterogeneity

1) Challenge

Technology Heterogeneity, while being a notable advantage of MSA, poses a unique challenge to reusability within this paradigm. The inherent flexibility of MSA allows developers to choose diverse technologies for implementing microservices based on project requirements, client preferences, or team expertise. However, this advantage becomes a double-edged sword when it comes to reusability. The challenge arises when developers create microservices with identical functionalities and interfaces but implement them using different technologies. This makes it challenging to seamlessly reuse microservices in projects with specific technological constraints, hindering the straightforward interchangeability of components. Consequently, when a development team seeks to reuse a particular microservice, the decision is complicated by the need to align with the client's preferred technology or adhere to existing project frameworks. One software engineer from the practitioners mentioned: "... despite having a handy catalog of microservices, there were instances where I found myself having to rebuild a microservice from scratch. You know how it goes – sometimes, the client has their preferences (like a strong preference for Java, for example), and that means starting anew. It can be a bit frustrating, but hey, client satisfaction comes first..." (Quote 2)

2) Addressing the challenge

Addressing the challenge of technology heterogeneity in MSA has led practitioners to two insightful solutions:

1) **Model-Driven Engineering (MDE)**: The majority of our practitioners, numbering 25, have embraced MDE as a powerful tool to navigate the complexities of technology heterogeneity. They've created code templates and generators for each microservice, essentially establishing a blueprint or skeleton [30]. This approach enables them to swiftly generate the required microservice with the preferred technology, streamlining the development process.
2) **Evolutionary Shift in Perspective or Single Technology**: A smaller group of 3 practitioners has taken a different route. They've chosen to overlook the nuances of technology heterogeneity, considering it a relic of the past when the battle between technologies was more pronounced. Today, with technological equality prevailing, they find that the initial advantages associated with different technologies have largely dissipated. As one practitioner highlighted: "Back in 2017 and earlier, NodeJs was our go-to for real-time data microservices due to its ease with WebSockets and Socket.IO[2]. However, now, all technologies have packages that handle real-time data efficiently, eliminating the need for diverse languages" (Quote 3). All practitioners employing this solution share a common trait – they operate within domain-specialized companies. These organizations focus exclusively on specific domains such as Healthcare, E-commerce, or Education. This specialization allows them to tailor their technology choices and development strategies according to the unique needs and intricacies of their respective domains. In doing so, they can strategically align their technology stack with the specific requirements of their industry, minimizing the impact of technology heterogeneity on their microservices landscape.

[2]https://socket.io

C. Service boundaries

1) Challenge

The service boundaries challenge stems from the absence of a standardized methodology or clear guidelines for identifying and extracting microservices within a project. This results in each team adopting different methods, often relying on experiential knowledge rather than strict guidance.Ait Said et all.[5] and Zhou et al. [31] highlighted a similar issue in their exploration of MSA practices in real-world companies, noting that practitioners tend to rely on experiences from similar projects for microservice identification. The lack of a standardized approach has significant repercussions on reusability in MSA, potentially leading to the development of similar features across different microservices. It also poses challenges in identifying whether a specific microservice already exists in the catalog. A practitioner's real-world case further illustrates this challenge: "... so when I first jumped into MSA for an e-commerce platform, I crafted two separate microservices – one looking after the blogs and another for products. Now, both these microservices had some kind of user feedback going on. The blogs featured comments with reactions, and the products had comments with ratings. Guess what happened next? As it turns out, these functionalities, although developed independently as different features, were essentially the same. So, to tidy things up, we created a brand new microservice called 'feedback.' This nifty microservice now takes care of all comments, reactions, and ratings, making it this an independent module that we can use across different types of content... This scenario is repeated especially between different teams, only in the last years, we noted that the payment microservice was built 8 times by different teams and 13 times payment functionalities were implemented as part of other microservices..." (Quote 4).

2) Addressing the challenge

Addressing the service boundaries challenge involves the adoption of two prominent methodologies of microservices identification: Domain-Driven Design (DDD)[32] and Functional Decomposition (FD)[33].

- **DDD**: This approach gained favor among 24 practitioners who leveraged the power of Domain-specific Language (DSL) generated through DDD. With DDD, practitioners found a robust solution to precisely define each microservice and facilitate the handling of similar functionalities. The DSL played a pivotal role in the microservices' clarity and served as a valuable tool for practitioners to compare new microservices with existing ones in the catalog. The inherent structuring and clarity afforded by DDD proved instrumental in overcoming the challenges associated with defining and extracting microservices [34].

- **FD**: While not as widely adopted as DDD, FD found preference among 4 practitioners. This approach involves breaking down a system into its functional components, offering a distinctive perspective on microservices design. Practitioners employing FD focused on identifying and defining microservices based on the discrete functions they perform within a system. Despite being less prevalent, FD provided an alternative framework for practitioners who found value in a functional-centric approach to microservices.

D. Versioning

1) Challenge

Versioning poses a significant challenge in MSA, especially when various versions of a microservice coexist simultaneously or multiple updates are pushed to the catalog concurrently with distinct functionalities. The crux of the problem lies in the potential mismanagement of interfaces between these versions, making it challenging to guarantee consistent reuse of microservices across the organization. The simultaneous usage of different microservice versions or the concurrent introduction of updates to the catalog can create complexities in maintaining a standardized approach to versioning. This challenge becomes more pronounced when the interfaces between different versions are not effectively managed. When versioning issues arise, ensuring seamless and consistent reuse of microservices becomes a formidable task. Without well-defined interfaces, compatibility issues may surface, hindering the smooth integration of microservices into diverse projects.

2) Addressing the challenge

Practitioners underscored key aspects of versioning management in MSA. Semantic Versioning was highlighted 14 times for its structured approach, aiding clear communication of changes. API gateways with robust version control were deemed essential by all 28 practitioners, centralizing version management and enhancing visibility. Additionally, rollout strategies like feature toggles and canary releases were emphasized by 9 practitioners, enabling controlled and iterative introduction of new microservice versions.

E. Decision-making

1) Challenge

Decision-making particularly regarding MDE adoption for enhancing reusability, presents a complex dilemma. While MDE offers significant advantages like generating microservice templates, its implementation entails considerable time and resource investment. Organizations must carefully weigh the benefits of reusability against the costs of MDE development. Factors such as microservice nature, functionalities, reuse frequency, development timelines, and long-term reusability impact must be meticulously evaluated in the decision-making process.

2) Addressing the challenge

The practitioners who encounter the decision-making challenge unanimously agree that relying on experience becomes a crucial factor in navigating this complexity. Drawing parallels with past projects, understanding the nature of microservices, and assessing the potential benefits against the costs of MDE implementation are common practices. This experience-driven decision-making approach offers a pragmatic way to balance the desire for reusability with the practical constraints of resource allocation and project timelines.

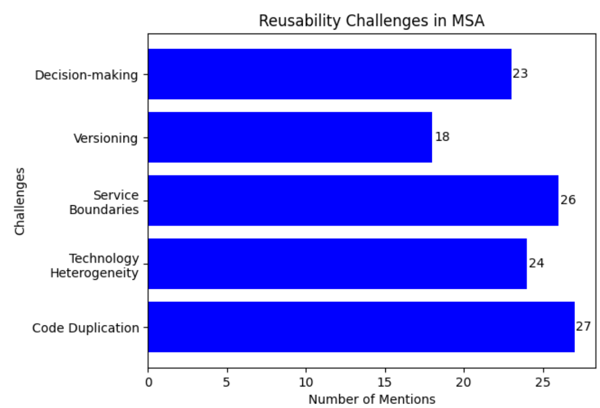

Figure 2. Reusability Challenges in MSA

6. Proposed Framework

A. Base concepts

In this section, we provide a foundational introduction to the base concepts underlying our proposed framework. By elucidating the core concepts driving our framework's design and development, we aim to lay a solid groundwork for the subsequent discussion and evaluation.

1) Microservices Hub

The Microservices Hub (MH) serves as the backbone of our proposed framework, playing a pivotal role in enhancing reusability within MSA. This catalog serves as a comprehensive repository housing all previously developed microservices, carefully curated for potential future reuse. Every microservice enlisted in the MH undergoes meticulous documentation, offering a detailed insight into its functionality, purpose, and utilization. This documentation is achieved through a clear DSL and a thorough description of use cases, accompanied by a historical account of the teams involved in its development. The inclusion of a change log further enhances transparency, providing a dynamic record of modifications and updates.

Microservices within the MH can manifest in two distinct forms, adapting to the organizational preference. For organizations prioritizing MDE to address the Technology Heterogeneity challenge, microservices take the form of Models. These Models encapsulate code templates and generators, streamlining the process of microservice development in diverse technological landscapes. Alternatively, organizations that do not require Technology Heterogeneity can opt for the single code source format.

2) Tech Masters Team

The TMT, a distinctive component in our proposed framework, introduces a paradigm shift in project dynamics within tech organizations. Traditionally as presented in the top side of Figure 3, projects are led by a combination of a product manager, tech leader, software architect, and developers. Our framework introduces an addition to this structure, positioning the TMT at the nexus between the product manager and development teams as shown in the bottom side of Figure 3, with a focus on three key roles:

1) **Firstly**, the TMT takes on the crucial task of identifying microservices in new projects. Even with methodologies like DDD, the process of microservices identification often relies on experiential knowledge [5], lacking strict guidelines. The TMT bridges this gap by centralizing viewpoints and expertise, making microservices identification more efficient and consistent. Leveraging their experience and insight, the TMT play a pivotal role in aligning project requirements with potential microservices, thereby optimizing the identification process.
2) **Secondly**, the TMT assumes control over the MH. This team holds the exclusive authority to assign development teams to specific microservices and oversees the introduction of new microservices or features into the MH. After the identification of microservices, the TMT strategically matches them with the most suitable development teams. Any addition to the MH, whether it be a new microservice or feature, undergoes rigorous validation by the TMT. This centralized control ensures a cohesive and standardized approach to microservices development, enhancing overall reusability.
3) **The third role** of the TMT involves workflow control. Positioned between the product manager and development teams. As presented in Figure 4, the TMT act as facilitators, directing project/feature requirements and specifications to the appropriate teams. In our framework, we aim to minimize coupling between projects and microservices earmarked for reusability. The development teams, functioning as a supply chain for features and new microservices, may not necessarily be aware of the project they are contributing to. This strategic decoupling allows teams to focus on building and upgrading specific microservices over an extended period, fostering specialization and expertise. The TMT orchestrates this workflow, promoting efficiency and skill development within the development teams.

3) Classification Framework

The Classification Framework , a cornerstone of our proposed methodology, addresses the nuanced challenge of deciding which microservices should be earmarked for reusability, considering the potential costs associated with methodologies like MDE because they add a layer on top of the native code source. While the aim is to enhance reusability, it's crucial to allocate resources judiciously and focus efforts on microservices that offer the most significant impact.

Our framework introduces a Classification Framework that operates across three distinct levels of reusability: Cross-Domain (CD), In-Domain (ID), and Single Project (SP):

1) **CD Microservices**: These microservices exhibit a lack of specific domain affiliation, making them versatile and applicable across various contexts. Examples include Payment Service and Users Management Service. CD microservices boast high Reusability Potential (RP) and are inherently designed for broad reuse. Additionally, this classification encompasses technical capability microservices like Messaging Service and File Storage and Management Service, further contributing to their applicability across domains.
2) **ID Microservices**: In this classification, microservices are characterized by their ability to be reused within specific domains. Examples include Fleet Management Service and Point of Sale Management Service. While ID microservices may not possess the same level of versatility as CD microservices, they still exhibit a moderate RP. These microservices are well-suited for reuse within the defined boundaries of specific domains, offering a balance between broad applicability and contextual relevance. This type of microservices can be considered as CD microservices in organizations specializing in a single domain such as Healthcare, Education, E-commerce...
3) **SP Microservices**: This category includes microservices closely aligned with specific business requirements or use cases of a system, rather than a broader domain. Examples could be custom microservices tailored to unique project needs. SP microservices have a lower RP due to their limited applicability beyond their specific context. While their reuse may be constrained, they play a crucial role in addressing specific project requirements.

B. Reusable Microservices Framework (RMF)

The whole RMF process is presented in Figure 5, as shown the process starts with the standard practice of defining and validating project specifications and requirements by the product manager. Subsequently, the TMT takes

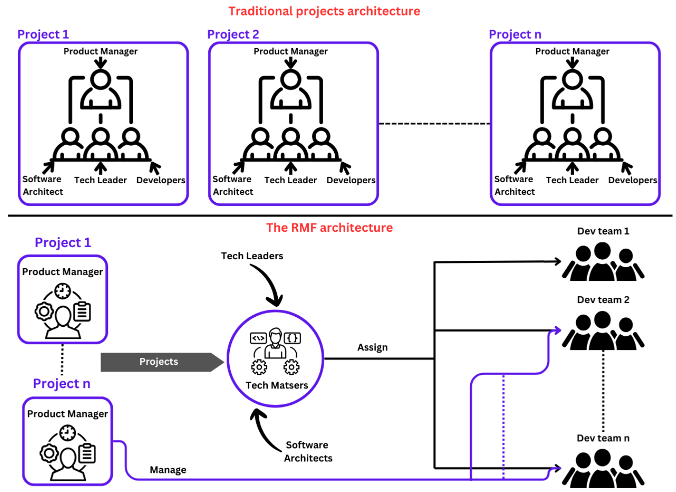

Figure 3. Traditional project architecture VS RMF architecture

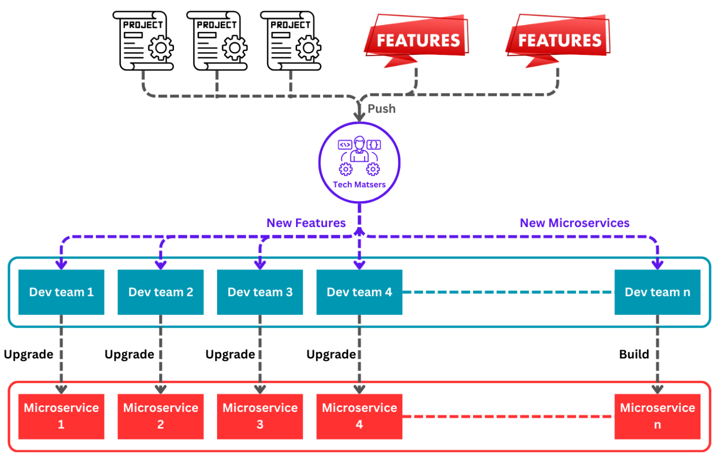

Figure 4. The high-level abstraction of RMF

center stage in the RMF process. The primary responsibility of the TMT is to analyze the provided specifications and requirements and apply DDD decomposition methodology to extract a collection of microservices (MsSet = MS1, MS2, ...MSi, … MSn). DDD proves instrumental in this step by enabling the use of DSLs, thereby enhancing the quality of documentation within specific domains. The DSLs play a pivotal role in locating Relevant Pre-Built Microservices (RPBMs) within the MH. The RMF unfolds in two key cases for each MSi in the MsSet:

1) **Case 1 (Presence of RPBM)**: If an RPBM is found in the MH for the new variant of microservice (MSi), the TMT identifies a suitable Dev Team. Ideally, this could be the team that originally built the RPBM or previously worked on it. In the absence of the original team, the TMT decides on a new development team. Subsequently, the chosen Dev Team pulls the RPBM from the MH, conducts a thorough comparison with the MSi, and determines whether additional features are present. If affirmative, the team develops the new features following the MDE process and pushes a new version to the MH. If not, the MDE process is invoked to generate the code source. Customizations, such as UI/UX enhancements, are implemented as necessary.
2) **Case 2 (Absence of RPBM)**: If no RPBM is found in the MH, the TMT assesses whether the new microservice should be considered for potential reusability based on the established Classification Framework. A Dev Team is then assigned accordingly. If the microservice is deemed reusable, the team follows the MDE approach to build and push it into the MH. Conversely, if the microservice does not align with reusability criteria, traditional development methodologies are employed.

The final steps of the RMF involve the standard procedures of testing and deploying the system, ensuring that the implemented microservices meet the specified requirements.

When new features are slated for integration into existing projects, the TMT assesses the features to determine the most suitable team for implementation, aligning with the standard RMF process. In projects where microservices are already identified and developed, the Dev Team starts from the step of checking if these new features are already developed in the RPMS and continue the process normally.

The RMF offers a comprehensive and structured methodology to enhance reusability within MSA, optimizing the allocation of resources and fostering a culture of efficient microservices development and reuse. Notably, as presented in Figures 3 and 4, there is no direct interaction between the Product Manager and Dev Teams. However, it's important to highlight that the Product Manager retains the capacity to assist and validate the development process if necessary. This collaborative approach ensures that the development aligns seamlessly with the project specifications and meets the desired outcomes.

In Case 1, when the suitable Dev Team is already engaged with the same microservice, the TMT is required to push the new variant of microservice or features exclusively to this team. To enhance development efficiency further, we propose the monitoring and tracking the frequency of adding new features to each microservice within the MH. This information becomes invaluable when assigning dev teams. By strategically analyzing the upgrade frequency of a microservice in the MH, a decisive step can be taken to establish a dedicated Dev Team for that particular microservice. This dedicated team can adeptly handle the continuous influx of features from various projects. If the demand for adding features to this microservice is substantial, scalability becomes a viable option. This Single Responsibility Principle (SRP) approach transforms the development process into a software supply chain (Figure 4) controlled by the TMT, where the developer might be working on a microservice without explicit knowledge of the associated project. This agile and scalable methodology aligns with the concept of managing microservices as independent units, promoting efficiency and adaptability within the development workflow.

The collaboration between DDD and DSL significantly boosts development efficiency. DDD offers a structured methodology for precise microservices definition within domains, while DSL, generated through DDD, acts as a common language for clear communication. This synergy aids in efficient microservice identification, utilizing DDD's decomposition of project specifications and DSL as a navigational tool in the MH. DSL enables quick feature comparison, helping Dev Teams identify additional features for new microservice versions. Standardized language reduces extensive communication between stakeholders, fostering seamless understanding. By treating microservices development as a supply chain, DDD and DSL enhance team independence, promoting autonomy and reducing dependencies.

In organizations favoring a Single Technology Approach, MDE may not be deemed necessary for microservices development. However, the Classification Framework remains crucial. While MDE streamlines reusable microservices creation, its absence doesn't diminish complexities in building such components. Without MDE, developers may face challenges ensuring microservices reusability across projects. The Classification Framework evaluates reusability potential and domain specificity, aiding in identifying reuse opportunities and streamlining development. Thus, even without MDE prioritization, the Classification Framework optimizes reusable microservices creation and deployment.

7. Validation and evaluation

The RMF underwent validation and evaluation through a two-year simulation and adoption in a real-world Moroccan software company (size of 50 to 100), complemented by interviews to gather feedback. Augmenting this process, insightful interviews were conducted to solicit feedback from the teams actively engaged in RMF adoption. This company, with a pre-existing commitment to reusability, adheres to the traditional Shared Catalog Approach (SCA), encompassing code snippets, assets, and web components.. This established focus on reusability sets a robust backdrop for validating and substantiating the efficacy of the RMF within an environment already attuned to systematic reuse.

Our validation approach was guided by a dual focus: firstly, understanding the adoption cost incurred during the integration of RMF into the existing development ecosystem, and secondly, quantifying the percentage of code reusability achieved through the framework and comparing it with another approach. These two pivotal insights form the bedrock of our assessment, offering valuable perspectives on the practicality and effectiveness of the RMF in real-world scenarios.

A. Environment preparation for RMF adoption

The adoption of the RMF was a strategic initiative undertaken by four development teams, each comprising 3 to 5 software engineers, totaling 16 experienced professionals with a minimum of 2 years of expertise in MSA. The pivotal TMT, consisting of a software architect and tech leader, both boasting over a decade of experience, anchored the adoption process with their profound understanding of MSA and MDE. The adoption process followed in four stages:

1) **Training in MDE (1 week)**: The journey began with an intensive week-long training on MDE for all four development teams by the TMT.
2) **In-Depth analysis of old projects (1 week)**: Collaborating with the TMT, the development teams delved into a comprehensive analysis of projects spanning up to five years, irrespective of their architectural paradigm (MSA or monolithic). The objective was to identify recurrent patterns and common functionalities. This meticulous exploration resulted in the identification and extraction of 23 microservices (B

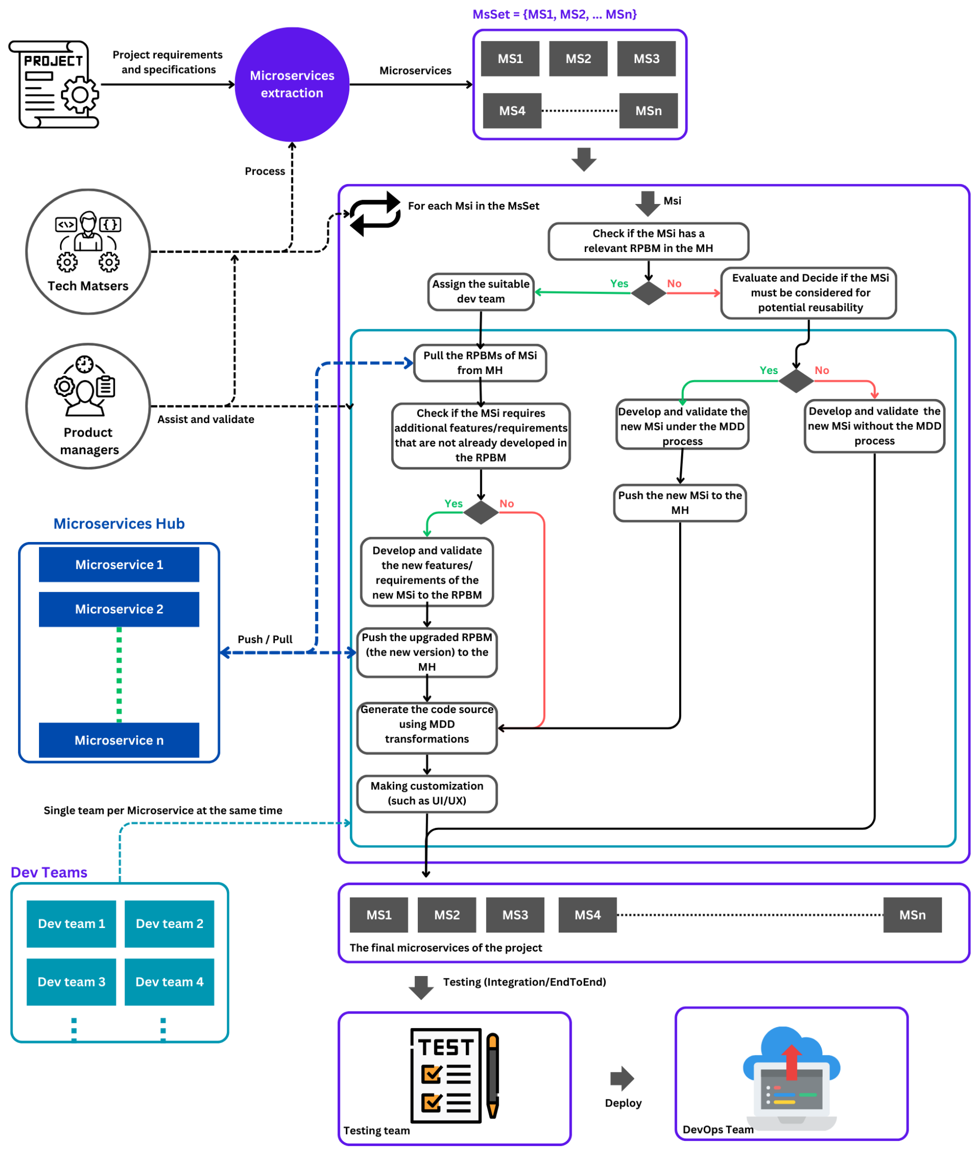

Figure 5. Reusable Microservices Framework

in the data repository), consolidating features that served identical contexts across various projects. In the process, we consider only the CD and ID microservices for potential reusability.

3) **Global MDE generator development (in the same week of step 2)**: The subsequent step involved the creation of a universal MDE generator capable of producing code compatible with Java, C#, and Python. This global generator such as Forms, and Tables ..., not bound to any specific context, would serve as a foundational tool for subsequent microservices development.

4) **Development marathon for MH building (8 weeks)**: The teams embarked on an 8-week development marathon to construct the MH. Employing MDE for microservices development, the teams harnessed the global generator to streamline the process. The output was an MH housing 23 pre-built, well-documented microservices. The MH itself developed as a web platform built on GitLab, featuring an interface comprising a microservice list with brief descriptions and a search bar with tags. Each microservice has a dedicated page offering detailed information, including historical usage in projects, feature pushes, comments, and a comprehensive documentation page.

The compiled list of extracted microservices is documented in Table B in the data repository. These microservices, derived from the in-depth analysis of past projects, are made available for use by any company embarking on the adoption of the RMF. This resource is particularly beneficial for new companies without pre-existing projects for analysis, providing a ready-made foundation to jumpstart their implementation of the RMF

B. *Cost Implications*

The adoption and environment preparation phase encompassed 10 working weeks, involving the concerted efforts of 16 software engineers and the expertise of the TMT. After an initial 3-week intensive engagement, the role of the TMT evolved to accommodate external tasks. The cost incurred during this phase is foundational, serving as a bedrock for subsequent microservices development. It's noteworthy that the adoption cost is an upfront investment, with additional costs incurred when new features or microservices are considered for future reusability. The implementation of MDE, with its abstraction layer of code generators, introduces efficiency benefits, as attested by the development teams who reported increased efficiency after an initial adaptation period of 2 to 3 weeks, turning MDE into a valuable asset, particularly when constructing common components like Tables. Subsequent interviews with the development teams will shed light on the evolving dynamics and the impact of this adoption on their workflow.

C. *Real-world adoption*

Throughout the real-world adoption of the RMF in a Moroccan software company, spanning approximately two years from September 6, 2021, to the end of 2023, the validation process unfolded. Over this period, a total of 9 projects were developed under the RMF framework, adhering strictly to the RMF environment's guidelines. It's noteworthy that only new features and maintenance tasks from these RMF-adopted projects were integrated into the MH, fostering its growth organically. We intentionally focused on building a catalog of microservices exclusively from RMF-adopted projects during this validation phase.

The development teams and the TMT occasionally engaged in external tasks when no RMF-related tasks were in progress. Throughout this period, the entire adoption process was closely monitored by the four researchers involved in this work. For each of the 9 projects developed under RMF, we calculated the reusability percentage by comparing the total number of functionalities (TF) with the number of functionalities generated (GF) from the MH.

Concurrently, we observed the development of 31 projects by other teams within the company during the same timeframe who adopted the SCA. The reusability percentage for these projects was assessed using a similar method, incorporating components from the company's existing catalog, such as code snippets and web components.

Under the RMF framework, the reusability of functionalities saw a substantial improvement, ranging from an initial 24% to a peak of 75%, with an average reusability percentage of 52.22%. Figure 6 illustrates the reusability percentage for each project, ordered by their completion date. Notably, the reusability percentage consistently increased with the introduction of each new project and the addition of new features, showcasing the positive impact of the RMF framework.

In contrast, Figure 7 blue line presents the reusability percentage for the observed 31 SCA projects outside of the RMF environment, organized by their completion dates. The reusability percentages in this case exhibited variation and a slower increase over time, with an average of 23.68%. This average is notably lower than the average reusability percentage observed under the RMF framework. This contrast highlights the effectiveness of the RMF methodology in systematically enhancing reusability when compared to conventional development practices within the organization.

D. *Simulation*

In the real-world adoption, it became evident that the RMF significantly enhances reusability compared to the traditional global SCA, albeit with a slight gap. A direct comparison between the 9 projects under RMF and other SCA projects, which varied in specifications and requirements, revealed this distinction.

To address this gap, a comprehensive simulation was conducted during the adoption period. This simulation involved replicating the development of the 31 projects that were originally executed with SCA, maintaining the same chronological order of project arrivals and feature implementations. The simulation aimed to emulate the RMF process for these projects, utilizing the same MH constructed at the beginning of the adoption period. The four researchers, acting as both the TMT and development teams, analyzed each project, extracted microservices, and simulated the RMF process by documenting the microservices and features without actual development. The documentation served as a skeletal representation of each microservice's features, with additions made for new features.

The reusability percentage, calculated using the same method applied in the real-world adoption, demonstrated a noteworthy outcome. As presented in Figure 7 green line, over each simulated project and subsequent feature additions, the reusability percentage consistently increased. Starting at 21%, it reached a peak of 87% by the end, with an average reusability percentage of 54.10% and a max of 91%.

In the comparative analysis between the RMF and the SCA, the results showcase the superior performance of the RMF in enhancing reusability. The reusability percentages across the simulated projects consistently demonstrate higher values when using the RMF compared to the SCA. This trend is especially noticeable in the average figures, with the RMF demonstrating 54.10%, reaching a maximum of 91%, while the SCA lags significantly with an average of 23.68% and a maximum of 42%, and in the P28, the RMF successfully triples the reusability rate. This pronounced difference underscores the efficacy of the RMF in methodically fostering reusability across a spectrum of diverse projects.

In the evaluation of both our RMF and the SCA, it was observed that when receiving projects from new domains such as P11, P16, P17, and P21, there was a notable reduction in reusability. However, a distinct advantage emerged for RMF, as it consistently maintained reusability percentages above 20%. This resilience can be attributed to the foundational principle of RMF, which relies on pre-built microservices designed for cross-domain applicability. Noteworthy examples include microservices like User Management and Payment Processing, which possess versatile functionalities applicable across diverse projects. Consequently, each new project phase introduces a higher probability of incorporating one of these pre-built microservices, contributing to the sustained reusability of the RMF framework. In contrast, the SCA experienced more substantial reductions, underlining the efficacy of RMF's design in enhancing reusability across varying project domains.

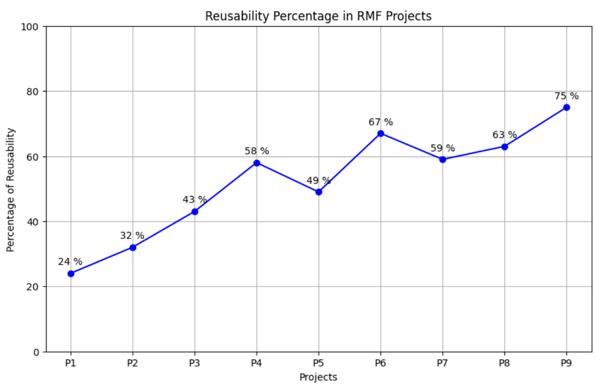

Figure 6. Reusability Percentage in RMF Projects

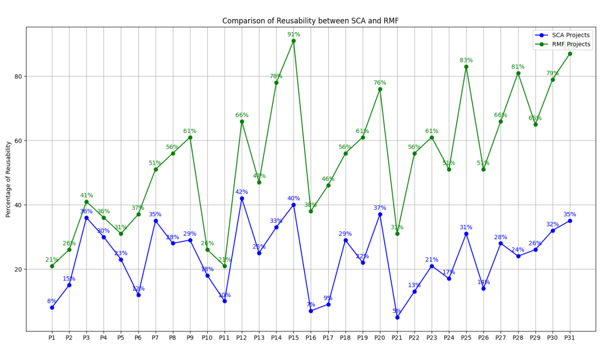

Figure 7. Comparison of Reusability between SCA and RMF

E. Evaluating RMF Adoption and Impact

1) Methodology

In this section, we present the evaluation of the RMF, employing a qualitative methodology to delve into its multifaceted impact on software development practices and scrutinize the adoption of RMF comprehensively, and determine its overall worthiness as a framework for microservices development and reuse within the company's software engineering landscape. The evaluation is driven by six primary objectives, including assessing user experience (Ob1), evaluating productivity (Ob2), measuring satisfaction levels from stakeholders (Ob3), identifying challenges and limitations (Ob4), gathering recommendations (Ob5), and assessing cost-effectiveness with a focus on measuring return on investment (ROI) (Ob6).

To achieve these objectives, we conducted interviews with key stakeholders involved in the adoption of RMF. The interviewees included the 16 software engineers from the development teams, the 2 experts from the TMT, and two Managers with decision-making roles within the company. In total, we engaged with 20 stakeholders.

For the interview process, we devised 15 questions presented in Table C in the data repository in the appendix. The first set of questions (Q1 to Q10) were directed towards the 16 software engineers and the 2 TMT experts, focusing on aspects related to user experience, productivity, drawbacks, and recommendations. The second set of questions (Q11 to Q15) was specifically tailored for the two Managers, delving into aspects related to Satisfaction and cost-effectiveness. This comprehensive approach ensured a holistic understanding of the RMF's impact from both the development and managerial perspectives.

2) Results and Discussion

Ob1: Assessing User Experience The assessment of user experience highlighted several positive impacts of the RMF on daily work and workflow efficiency. Enhanced collaboration among team members, accelerated development cycles, improved task management, and reduced repetitive tasks were notable strengths identified. However, an increase in stress levels, attributed to the responsibility of developing reusable code and components, was also acknowledged. This finding emphasizes the need for supportive measures to mitigate potential stressors and maintain a healthy work environment. Moreover, the examination of overall user experience revealed five key instances demonstrating the RMF's influence. These instances include workflow streamlining, quality improvement, communication facilitation through DSLs, customization flexibility, and effectiveness of provided documentation. While the MH user interface was recognized for its value, respondents expressed a desire for improvements, particularly in enhancing feature discoverability. The proactive approach of the TMT in addressing this challenge through the development of a AI-powered extension signifies a commitment to enhancing user-friendliness and continuous improvement within the

MH interface.

Ob2: Evaluating Productivity The evaluation of productivity enhancements resulting from the adoption of the RMF illuminated significant improvements across key agile characteristics. Rapid adaptation to the MDE approach within 2 to 3 weeks was observed among the majority of engineers, indicating the accessibility and user-friendliness of the RMF. The adoption of the RMF yielded notable improvements in automating repetitive processes, task management, resource utilization, and development cycle acceleration, aligning well with agile principles emphasizing efficiency and continuous improvement. Additionally, the adoption of the SRP within the RMF, where development teams were dedicated to specific microservices, facilitated streamlined communication, collaboration, and enhanced code quality. The SRP approach also fostered knowledge sharing and cross-functional collaboration, contributing to a cohesive understanding of the microservices ecosystem. Furthermore, respondents highlighted specific task and process improvements, including efficiency gains in feature development, streamlined workflows, accelerated bug identification and resolution, enhanced code review processes, and improved cross-team coordination. These findings underscore the multifaceted impact of the RMF on productivity, aligning with agile principles and demonstrating notable gains across various aspects of the development process.

Ob3: Measuring Satisfaction Levels from Managers The two managers expressed a high level of satisfaction with the RMF, citing its positive impact on project timelines and collaboration. While the RMF facilitated timely deliveries and successful outcomes, challenges arose in aligning unique or complex project requirements with the platform. Despite these challenges, the overall influence on timelines and collaboration remained significant, with the potential for further adjustments and improvements. The RMF was acknowledged for enhancing communication channels and fostering transparent, collaborative environments, although some complexities affected the clarity of project-related information. Stakeholders appreciated instances where feedback was integrated effectively, leading to improved project outcomes.

Ob4: Identifying Challenges and Limitations Teams encountered challenges during the adoption and implementation of the RMF, including learning curve hurdles associated with the MDE approach and integration complexities with existing tools and workflows. These challenges highlighted the importance of comprehensive training and dedicated efforts to align the RMF seamlessly with established processes. Additionally, teams faced limitations and drawbacks such as extensive documentation overhead, which slowed initial development speed, and challenges in managing dependencies between microservices, necessitating meticulous coordination to mitigate disruptions. Striking a balance between documentation and streamlined processes is crucial for maintaining efficiency, while resolving dependency management issues is essential for ensuring a cohesive development process.

Ob5: Recommendations for RMF Enhancement from Dev Teams and TMT vision In response to the experiences with the RMF, several key recommendations emerged. Teams expressed a need for enhanced customization features, calling for greater flexibility in tailoring RMF tools to meet specific project requirements. Improved integration capabilities, particularly with widely used development tools like Jira[3] and Notion[4], were highlighted as essential for creating a seamless project ecosystem. The significance of comprehensive training and ongoing support during the adoption process was underscored, with recommendations urging investment in extensive training resources. Security measures were a central concern, prompting recommendations to strengthen protocols and ensure robust data privacy features within the RMF. These recommendations collectively aim to cultivate a platform that is not only user-friendly and adaptable but also prioritizes security and efficient collaboration.

Ob6: Impact on Cost-Effectiveness and Return on Investment The perspectives of the two managers on the RMF's influence on cost-effectiveness and ROI were notably positive. Despite acknowledging potential upfront implementation costs, both managers expressed a strong belief in the long-term value and benefits of the RMF. They highlighted its positive impact on the cost-effectiveness of development projects, emphasizing how the RMF streamlined workflows, reduced redundant tasks, and enhanced collaboration, ultimately leading to improved project outcomes. In terms of ROI, insights shared by the managers focused on the considerable time and resource savings facilitated by the RMF. They noted that the initial investment in adopting the RMF resulted in tangible benefits for their teams and the organization as a whole. The emphasis on long-term gains and efficiency gains underlines the strategic and forward-thinking approach that organizations can adopt when implementing the RMF, aligning with broader goals of productivity and value delivery.

8. Conclusion

This work has shed light on the critical role of software reuse within the context of MSA and its profound implications for modern software engineering practices. By identifying and addressing five key challenges—Code Duplication, Technology Heterogeneity, Service Boundaries, Versioning, and Decision-Making—we have proposed the RMF as a systematic approach to optimizing reusability in MSA environments. Developed in collaboration with MSA practitioners and grounded in industry insights, the RMF offers a comprehensive solution to enhance reusability practices. Through simulations and real-world implementations, including adoption in a software company setting, we have demonstrated the significant improvements in

[3]https://www.atlassian.com/software/jira
[4]https://www.notion.so/

reusability achieved with the RMF, exceeding threefold in observed cases. In future work, we plan to address remaining challenges like tool integrations within the RMF. Additionally, we aim to create a public platform for the MH, enabling shared access to microservices across companies. This initiative will foster an RMF community, promoting the exchange of best practices and facilitating widespread adoption of reusability within MSA.

References

[1] R. Capilla, B. Gallina, C. Cetina, and J. Favaro, "Opportunities for software reuse in an uncertain world: From past to emerging trends," *Journal of Software: Evolution and Process*, vol. 31, no. 8, 8 2019. [Online]. Available: https://doi.org/10.1002/smr.2217

[2] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, 4 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.01.001

[3] V. Raj and R. Sadam, "Performance and complexity comparison of service oriented architecture and microservices architecture," *International Journal of Communication Networks and Distributed Systems*, vol. 27, no. 1, p. 100, 1 2021. [Online]. Available: https://doi.org/10.1504/ijcnds.2021.116463

[4] J. Lewis and M. Fowler, "Microservices," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[5] M. AIT SAID, A. EZZATI, S. MIHI, and L. BELOUADDANE, "Microservices Adoption: An Industrial Inquiry into Factors Influencing Decisions and Implementation Strategies," *International Journal of Computing and Digital Systems*, vol. 15, pp. 1417–1432, 3 2024. [Online]. Available: http://dx.doi.org/10.12785/ijcds/1501100

[6] A. Razzaq and S. A. K. Ghayyur, "A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges," *Computer Applications in Engineering Education*, vol. 31, no. 2, pp. 421–451, 11 2022. [Online]. Available: https://doi.org/10.1002/cae.22586

[7] M. Ait Said, A. Ezzati, and S. Arezki, "Microservices, a Step from the Low-Code to the No-Code," pp. 779–788, 11 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-20601-6_64

[8] ——, "Microservice-Specific language, a step to the Low-Code platforms," pp. 817–828, 1 2023. [Online]. Available: https://doi.org/10.1007/978-3-031-26384-2_72

[9] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2 2015, iSBN: 1491950358, URL: https://www.oreilly.com/library/view/building-microservices/9781491950340.

[10] ——, *Monolith to microservices*. O'Reilly Media, 9 2019, iSBN: 9781492047841, URL: https://www.oreilly.com/library/view/monolith-to-microservices/9781492047834.

[11] J. Doležal and A. Buchalcevová, "Migration from monolithic to microservice architecture : Research of Impacts on Agility," *IDIMT-2022 : digitalization of society, business and management in a pandemic : 30th Interdisciplinary Information Management Talks*, pp. 401–, 2022. [Online]. Available: https://doi.org/10.35011/IDIMT-2022-401

[12] G. J. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," *IEEE Access*, vol. 10, pp. 20 357–20 374, 1 2022. [Online]. Available: https://doi.org/10.1109/access.2022.3152803

[13] Z. Li, C. Shang, J. Wu, and Y. Li, "Microservice extraction based on knowledge graph from monolithic applications," *Information Software Technology*, vol. 150, p. 106992, 10 2022. [Online]. Available: https://doi.org/10.1016/j.infsof.2022.106992

[14] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to Microservices: An assessment framework," *Information Software Technology*, vol. 137, p. 106600, 9 2021. [Online]. Available: https://doi.org/10.1016/j.infsof.2021.106600

[15] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, and Z. Shan, "A dataflow-driven approach to identifying microservices from monolithic applications," *Journal of Systems and Software*, vol. 157, p. 110380, 11 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.07.008

[16] K. Gos and W. Zabierowski, "The Comparison of Microservice and Monolithic Architecture," *IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, 4 2020. [Online]. Available: https://doi.org/10.1109/memstech49584.2020.9109514

[17] H. Chawla and H. Kathuria, "Evolution of microservices architecture," pp. 1–20, 1 2019. [Online]. Available: https://doi.org/10.1007/978-1-4842-4828-7_1

[18] V. Velepucha and P. Flores, "Monoliths to microservices - Migration Problems and Challenges: A SMS," *Second International Conference on Information Systems and Software Technologies (ICI2ST)*, 3 2021. [Online]. Available: https://doi.org/10.1109/ici2st51859.2021.00027

[19] L. Cao and C. Zhang, "Implementation of Domain-oriented Microservices Decomposition based on Node-attributed Network," *11th International Conference on Software and Computer Applications*, 2 2022. [Online]. Available: https://doi.org/10.1145/3524304.3524325

[20] N. Santos and A. R. Silva, "A Complexity Metric for Microservices Architecture Migration," *IEEE International Conference on Software Architecture (ICSA)*, 3 2020. [Online]. Available: https://doi.org/10.1109/icsa47634.2020.00024

[21] Y. M. Abgaz, A. McCarren, P. Eklund, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yılmaz, J. Buckley, and P. M. Clarke, "Decomposition of Monolith Applications into Microservices Architectures: A Systematic review," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4213–4242, 8 2023. [Online]. Available: https://doi.org/10.1109/tse.2023.3287297

[22] V. Velepucha and P. Flores, "A survey on Microservices Architecture: Principles, Patterns and migration challenges," *IEEE Access*, vol. 11, pp. 88 339–88 358, 1 2023. [Online]. Available: https://doi.org/10.1109/access.2023.3305687

[23] LatourLarry, WheelerTom, and FrakesBill, "Descriptive and predictive aspects of the 3Cs model," *Ada letters*, vol. XI, no. 3, pp. 9–17, 4 1991. [Online]. Available: https://doi.org/10.1145/112630.112632

[24] L. Carvalho, A. Garcia, W. K. G. Assunção, R. Bonifácio, L. P. Tizzei, and T. E. Colanzi, "Extraction of Configurable and Reusable

Microservices from Legacy Systems," *Proceedings of the 23rd International Systems and Software Product Line Conference*, 9 2019. [Online]. Available: https://doi.org/10.1145/3336294.3336319

[25] M. A. P. Da Silva, V. C. Times, A. Araújo, and P. C. Da Silva, "A Microservice-Based Approach for Increasing Software Reusability in Health Applications," *IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, 11 2019. [Online]. Available: https://doi.org/10.1109/aiccsa47632.2019.9035229

[26] S. H. A. Hamed, "Reusability of legacy software using microservices: An online exam system example," *Al-Qadisiyah Journal for Computer Science and Mathematics*, vol. 15, no. 3, 9 2023. [Online]. Available: https://doi.org/10.29304/jqcm.2023.15.3.1263

[27] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science research Methodology for Information Systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 12 2007. [Online]. Available: https://doi.org/10.2753/mis0742-1222240302

[28] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decision Support Systems*, vol. 15, no. 4, pp. 251–266, 12 1995. [Online]. Available: https://doi.org/10.1016/0167-9236(94)00041-2

[29] S. Brinkmann and S. Kvale, *Doing interviews*, 1 2018, iSBN: 9781529716665, DOI: https://doi.org/10.4135/9781529716665, URL: https://www.sagepub.com/hi/nam/doing-interviews/book259212.

[30] J. Sorgalla, F. Rademacher, S. Sachweh, and A. Zündorf, "Model-driven Development of Microservice Architecture: An Experiment on the Quality in Use of a UML- and a DSL-based Approach," *Software Engineering*, 4 2020. [Online]. Available: https://kobra.uni-kassel.de/handle/123456789/11912

[31] X. Zhou, S. Li, L. Cao, H. Zhang, Z. Jia, C. Zhong, Z. Shan, and M. A. Babar, "Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry," *Journal of Systems and Software*, vol. 195, p. 111521, 1 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2022.111521

[32] P. Merson and J. W. Yoder, "Modeling Microservices with DDD," *EEE International Conference on Software Architecture Companion (ICSA-C)*, 3 2020. [Online]. Available: https://doi.org/10.1109/icsa-c50368.2020.00010

[33] S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, "Identifying microservices using functional decomposition," pp. 50–65, 1 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-99933-3_4

[34] R. A. Schmidt and M. Thiry, "Microservices identification strategies : A review focused on Model-Driven Engineering and Domain Driven Design approaches," *15th Iberian Conference on Information Systems and Technologies (CISTI)*, 6 2020. [Online]. Available: https://doi.org/10.23919/cisti49556.2020.9141150