UNIVERSITY OF SÃO PAULO - INSTITUTE OF MATHEMATICS AND STATISTICS
MAC 499 - Capstone Project

From Modular Monolith to Microservices: A Guide to Facilitating Scalable Software Development

Gabriel Arrais

Abstract

This topic has gained increasing relevance due to the prevalence of monolithic systems in the industry and the growing need for independent deployment and scalability of system components. The primary objective of this study is to provide valuable strategies and practical guidance for developers and software architects who intend to either design modular monoliths or migrate them to microservices. The methodology will involve the design and implementation of a fictional system following modular monolith principles, followed by its decomposition into microservices. Each phase of the process highlights the main architectural concepts, the challenges encountered, and strategies to overcome them. The expected outcome is the development of a reference framework that supports both the modular monolith design and the migration process, offering useful insights for each stage.

Keywords: *DDD; Monolith; Microservice*

---

1. INTRODUCTION

Modern software development increasingly demands scalable, maintainable, and evolvable systems. However, many organizations still operate legacy monolithic applications that were not designed with modularity or distributed architectures in mind. Migrating these systems to a microservice architecture presents numerous technical challenges, particularly due to tightly coupled components and a lack of clear service boundaries. This scenario creates a significant barrier for companies seeking to adopt modern architectural paradigms.

To overcome this barrier, adopting well-established literature at the early stages of system design can greatly contribute to the development of complex, yet maintainable systems. Important concepts such as Domain-Driven Design by Eric Evans offer strategic approaches for modeling business domains aligned with real-world processes, fostering a shared understanding between developers and domain experts. Complementary to this, architectural paradigms like Clean Architecture advocate for clear separation of concerns and independence of frameworks, which are essential when aiming for scalable and adaptable systems. Furthermore, the use of patterns such as Event Sourcing can enhance system resilience and traceability by persisting changes as a sequence of immutable events. These practices, when applied cohesively, not only facilitate long-term maintainability but also lay a solid foundation for a smoother transition to distributed architectures like microservices.

Given this context, the modular monolith emerges as a promising alternative. It enables teams to incrementally evolve monolithic systems while enforcing modular boundaries, serving as a strategic stepping stone toward microservices. This project is motivated by the need to explore and demonstrate how modular monolith principles can support maintainability and future scalability without requiring a full system redevelopment.

2. LITERATURE REVIEW

This section presents the fundamental concepts and topics required to understand the research project. After reviewing this section, the reader will be better prepared to follow the rest of the proposal and comprehend its underlying ideas.

2.1. Domain-Driven Design

The Domain-Driven Design(DDD) is a software development approach that emphasizes a deep connection between the software model and the business domain it represents. The main goal is to build systems that reflect real-world complexity through well-defined models, improving both communication and code clarity. DDD is especially useful for complex systems where business rules are central to the application, so it's highly recommended to design the domain in close collaboration with domain experts[1].

2.2. Monolith

A monolithic architecture refers to a traditional model where the entire application is designed as a single, unified unit. In this approach, all components are

tightly coupled into a single codebase. This results in a single executable file, which can be easier to develop initially but presents challenges as the system grows. One of the primary downsides of monolithic architectures is their lack of flexibility. As a system becomes larger, it becomes harder to maintain and scale because any change or enhancement often requires modifications to the entire application.

2.3. Microservice

The microservices architecture involves developing a single application as a collection of small, independent services. Each service runs in its own process and communicates with others through various mechanisms, often using HTTP-based resource APIs[2]. These services can maintain their own databases and may be implemented in different programming languages. This separation enables services to be deployed and scaled independently. Additionally, each service maintains its own codebase, allowing different development teams to work on individual services autonomously.

2.4. Modular Monolith

The Modular Monolith emerged to improve adaptability, facilitate system evolution, and even allow separate deployment of modules within a single monolithic system. It is crucial that it is built with clearly defined module responsibilities and communication strategies between them. This architectural style resembles a "middle ground" between monoliths and microservices and can address challenges of both models[3].

3. PROPOSAL

3.1. Goals

This research aims to offer valuable insights for developers and system architects seeking to design applications with modular monolith approach, while anticipating future scalability requirements and a potential transition to a microservices architecture. Furthermore, the research intends to contribute with a framework to assist practitioners when migrating from a modular monolith to microservices. This study will be guided by the following set of Research Questions (RQ):

RQ1: What are the trade-offs involving the key characteristics when building a modular monolith?

RQ2: What are the critical points in the migration process from a modular monolith to a microservices architecture?

RQ3: What architectural indicators can developers use to compare the modular monolith and the microservices models?

RQ4: What can alleviate the difficulties when migrating from a modular monolith to microservices?

3.2. Methodology

The methodological approach adopted for the development of this work is depicted in Figure 1, which outlines the main stages.

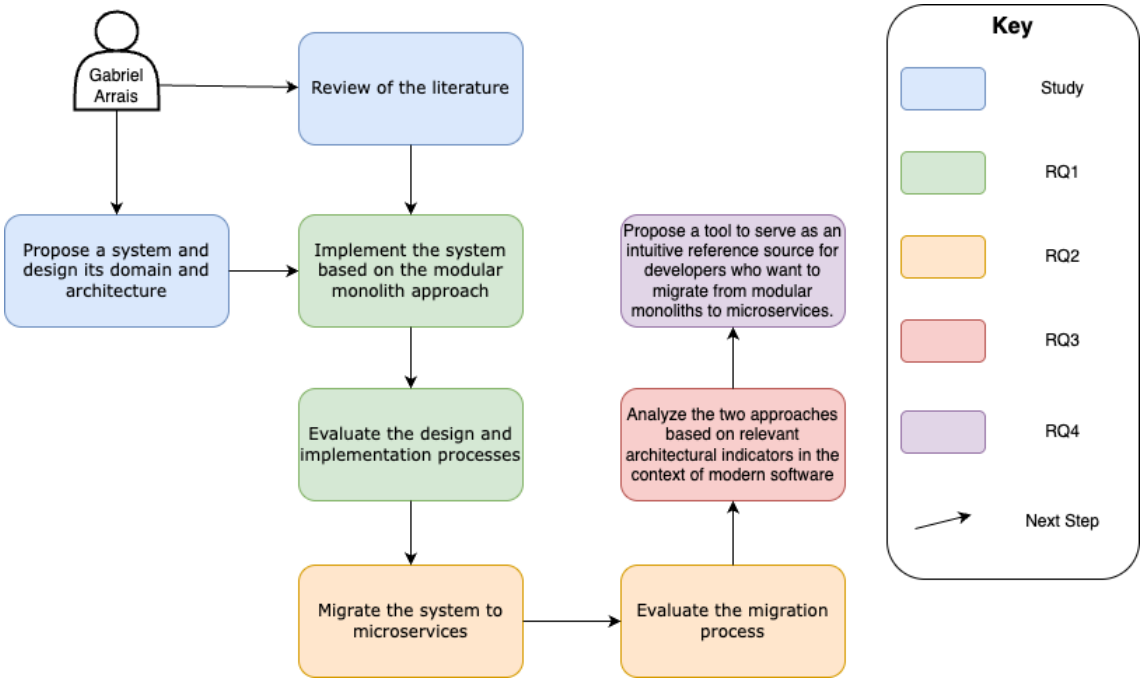

Figure 1. Work plan flow diagram

Initially, a review of the literature on Domain-Driven Design, Clean Architecture, Modular Monoliths, and Microservices will be conducted, serving as the theoretical foundation for the development of this research. This step is essential to understand established approaches and identify opportunities for further investigation.

Concurrently, the design phase of the system's domain and architecture will be carried out, using the concepts learned in the first step's literature. This stage is crucial, as a well-defined model with low coupling, clear interfaces, and well-established boundaries is essential to achieving the proposed objectives.

Once the domain has been defined, the implementation phase based on the modular monolith approach will begin. During development, the characteristics of the programming languages and frameworks used will be examined, with particular attention to their support for domain modeling, bounded contexts, and event handling. At the end of this stage, the implementation will be evaluated to identify the advantages, limitations, and challenges encountered. Together with the results of the previous step, this stage will provide the basis for answering RQ1.

After that, the system will undergo a migration to a microservices-based architecture. This phase will involve extracting the bounded contexts from the modular

monolith and adapting them into independently deployable microservices. Throughout this process, the critical points and challenges observed will be documented, with the goal of answering RQ2.

With both systems implemented, a comparative evaluation will be conducted using software architecture metrics such as coupling, cohesion, scalability, maintainability, and complexity. This analysis will contribute to answering RQ3 and provide a solid foundation to select appropriate architectural approaches in future projects.

Finally, a support tool will be developed to serve as a practical reference for developers seeking to migrate modular monolith systems to a microservices architecture. This tool will encapsulate the best practices, common challenges, and architectural decisions identified throughout the research, providing actionable guidance, contributing to the resolution of RQ4.

4. WORK PLAN

4.1. Timeline

1. Review of the literature on domain-driven design and modular monoliths.
2. Propose a system and design its domain and architecture.
3. Implement the system based on the modular monolith approach.
4. Evaluate the implementation.
5. Migrate the system to microservices.
6. Evaluate the migration process.
7. Analyze the two approaches based on relevant architectural indicators in the context of modern software.
8. Propose a tool to serve as an intuitive reference source for developers who want to migrate from modular monoliths to microservices.

Activities	Apr	May	Jun	Jul	Aug	Sep	Oct
1	•	•					
2		•					
3			•				
4				•			
5				•	•		
6					•		
7						•	
8							•

Table 1. Work plan's schedule

4.2. Expected Results

We expect to provide clear and well-founded answers to the questions raised in the proposal. The expected results of this study include:

- The discussion of the principal philosophical and architectural challenges encountered during the design of a modular monolith system, along with strategies to address them effectively.
- A documented approach to migrate from a modular monolith to a microservices architecture, emphasizing the advantages of a well-structured design.
- A comparative evaluation of modular monolith and microservice architectures, using relevant architectural metrics to assess qualities such as modularity, coupling, scalability, and maintainability.
- The development of a supportive tool aimed at assisting developers in migrating from modular monoliths to microservices.

References

[1] E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, Boston, 2003.

[2] M. Fowler and J. Lewis. Microservices, 2014. URL `https://martinfowler.com/articles/microservices.html`. Accessed: 2025-04-19.

[3] R. SU and X. LI. Modular monolith: Is this the trend in software architecture? In *Proceedings of the 2024 IEEE/ACM International Workshop on New Trends in Software Architecture (SATrends)*. IEEE, 2024.