# Review of Methods For Migrating Software Systems to Microservices Architecture

Paper Received: 03.10.2021.
Paper Accepted: 21.11.2021.

**Keywords:** Microservices; Microservices architecture; Software architecture; Reengineering; Migration methods.

# Contents

- Basic concepts

-Monolith Systems

-Microservices Architecture

-Reengineering

- Microservices vs Monolith Systems
- Why should migration be preferred?
- Research Methods
- Migration Algorithms
- Migrating Steps

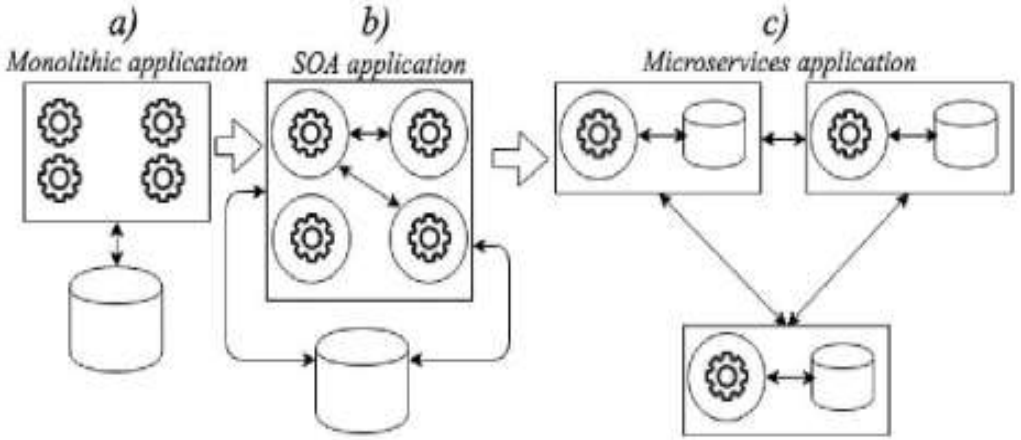
# Basic Concepts

**Monolith Systems**

- All functionality in one system
- Not standalone modules
- Good for startup projects but just first times
- Difficult deployment process

## Microservices Architecture

- Standalone modules
- Availability for DevOps and CI/CD process
- Like advanced SOA but not same
- When there is a problem in the system, only the relevar part is affected
- Communicate between network/HTTP API



- Loose coupling and hifg cohesion
- Independent services + different technologies
- SOA: share-as-much-as-possible

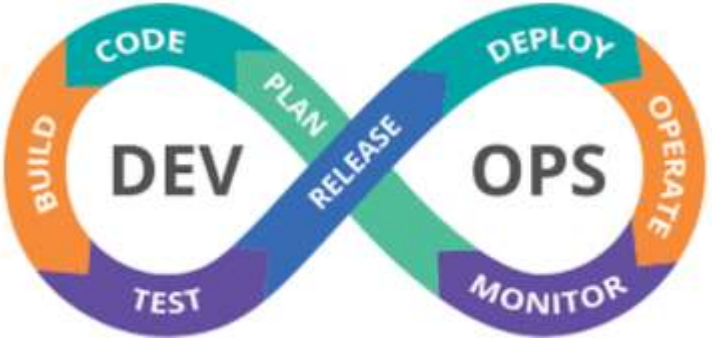
Microservices: share-as-little-as-possible

- **SOA**

An approach in which the functions of more than one application are designed to be used by other applications

- **DevOps**

An approach to systematize the software lifecycle




| Company | Deploy Frequency | Deploy Lead Time | Reliability | Customer Responsiveness |
|---|---|---|---|---|
| Amazon | 23,000 / day | minutes | high | high |
| Google | 5,500 / day | minutes | high | high |
| Netflix | 500 / day | minutes | high | high |
| Facebook | 1 / day | hours | high | high |
| Twitter[2] | 3 / week | hours | high | high |
| typical enterprise | once every 9 months | months or quarters | low/medium | low/medium |

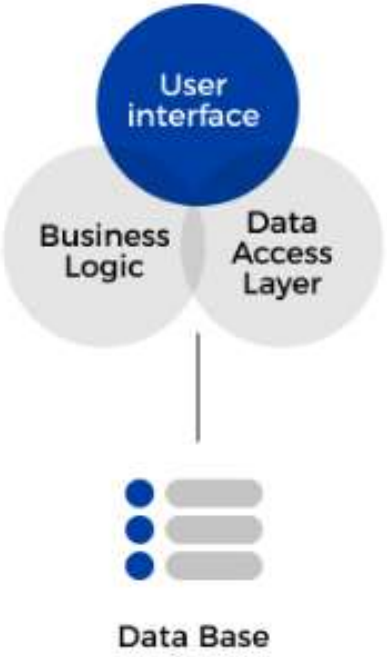
## Reengineering

- Same functionality but transformed old system to new architecural model
- Should start when system becomes too complex when upgraded new features
- Main task is determination of system functions

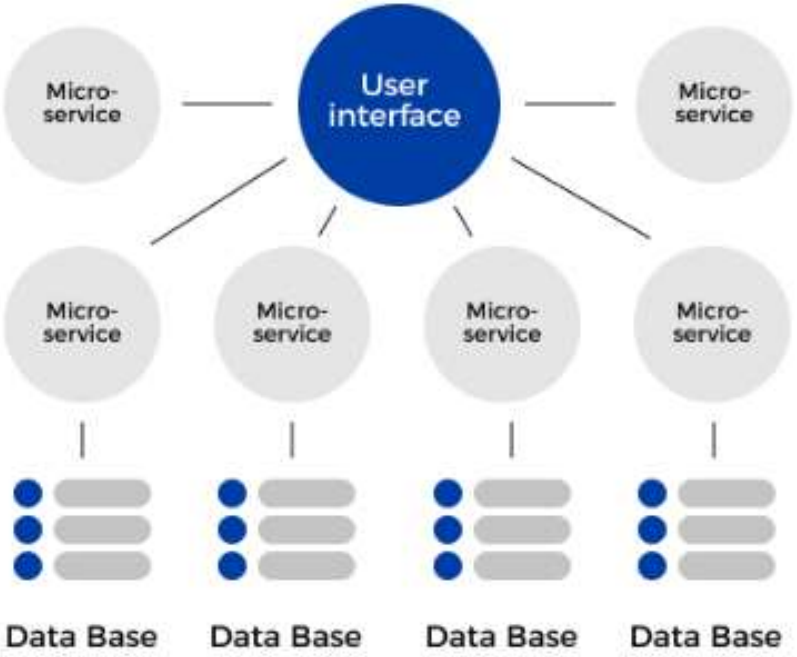
# Microservices vs Monolith Systems

| Category | Monolith | Microservices |
|---|---|---|
| Time to market | Fast in beginning, slow in growing | Slower in beginning, faster later |
| Refactoring | Hard, change affect multiple places | Easier change just inside service |
| Deployment | Deploy all system | Deploy just one service |
| Scaling | Deploy whole monolith | Done per service |
| Implementation | Written one language | All service can difference languages for performance |

## MONOLITHIC ARCHITECTURE



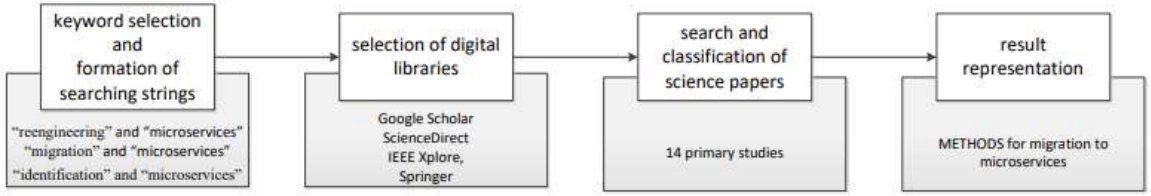

## MICROSERVICE ARCHITECTURE



# Why should migration be preferred?

- Easier Maintanence/Change System
- High Scalability/Availability
- Easier Team Management
- Multiple and fast deployments
- Reducing programmatic development risks

# Research Methods




- **Purpose of paper:** To collect, organize and analyze migration techniques in the literature.

-What are the migration algorithms used in the literature?

-Are there difficulties in transitioning from monolithic architecture to microservice architecture?

| Domain of use | Software type |
|---|---|
| Banking system | monolithic system |
| Cargo Tracking System | |
| industrial application | |
| renting bikes | |
| Learning System | monolith mobile application |
| web store | monolithic web application |
| software quality measurement and visualization tool | monolithic enterprise systems |
| web crawling project | |
| business system | legacy enterprise system |
| ferry booking system | |
| web-based monitoring and visualization tool | legacy system |
| IT company | legacy web system |
| Spring Boot Pet Clinic | open-source application |
| Forum | messaging boards application |
| Blog | blogging website |

# Migration Algorithms

- **Decomposition:** OpenAPI is used. Allows to separate functionality with similar concept into a group.

OpenAPI: Used to define RESTful services. Allows to extract the functionality of the program.

- **Semantic Assestment:** Analyzes the processes in the software system.
- **Fast Community Graph Clustering:** Software system is represented as a graph. Identifies microservice candidates.

- **NSGA II. Nondeminated Sorting Generic Algorithm:** Sorts and compares all solutions for detecting microservices.
- **SarF, Software Clustering Algorithm:** Aggregates software in clusters without human interaction.
- **Determination of microservices boundary:** Creates graphs according to functions. Classification is made by creating a matrix according to the application behavior.

- **AMI, Automated Microservices Identification:** Discovers key objects and splits them into microservices by extracting their connections to classes.
- **Colloborative Clustering:** Allows partitioning into microservices based on storage dependencies.
- **Affinity Propagation:** Based on the measurement of data similarity.

- Although the inputs in the algorithms differ, their purposes are basically the same.
- Divided into automatic, semi-automatic and manual.
- Common to software is that they use a monolithic system.

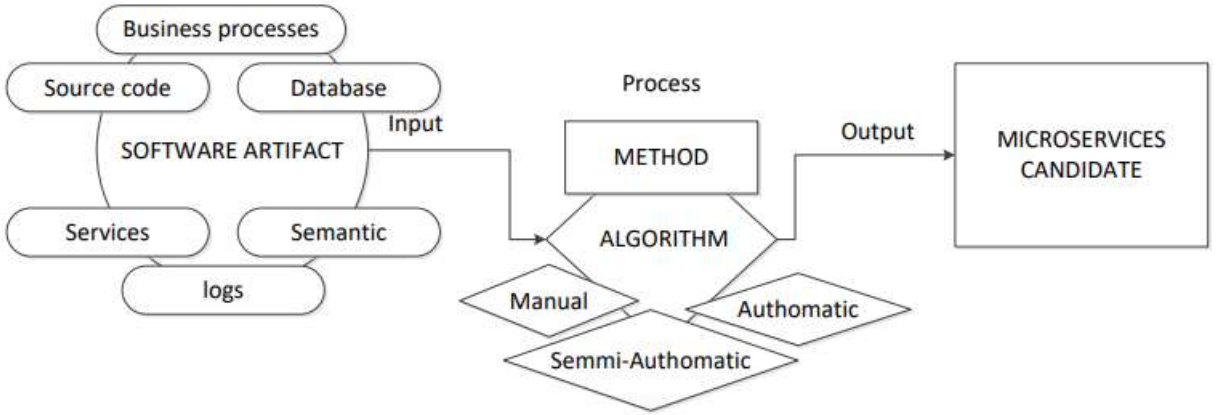



*Figure 2: General migration process*

## Challenges in migrating from monolithic to microservice architecture

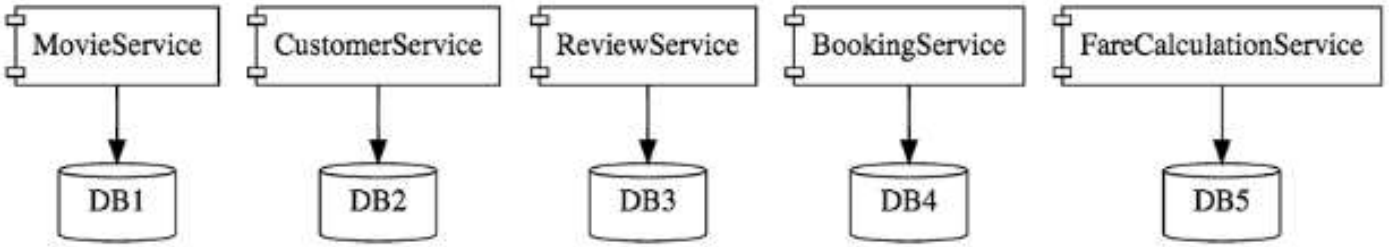
- Seperating business logic into sub-business logic
- Need proffesional developers
- Resource Management
- Adaptation of software teams

- **Database Migration**

-According to the survey, more than half of the participants stated that the existing data during the migration was not migrated.

-Not moving the database prevents services from developing independently.

-Database migrations are still a problem, according to research.



# Migrating Steps

1. Identify logical components
2. Flatten and refactor components
3. Identify component dependencies
4. Identify component groups
5. Create an API for remote user interface
6. Migrate component groups to macroservices (move component groups to separate projects and make separate deployments).
7. Migrate macroservices to microservices.
8. Repeat steps 6-7 until complete.

**1. Identify logical components**

Modules should be categorized for use in the next steps. Operations with data objects can be determined more easily.

**2. Flatten and refactor components**

In the final system, there should be only one microservice that performs any specific function.

**3. Identify component dependencies**

The source code is analyzed and visualized in a hierarchical order.

**4. Identify component groups**

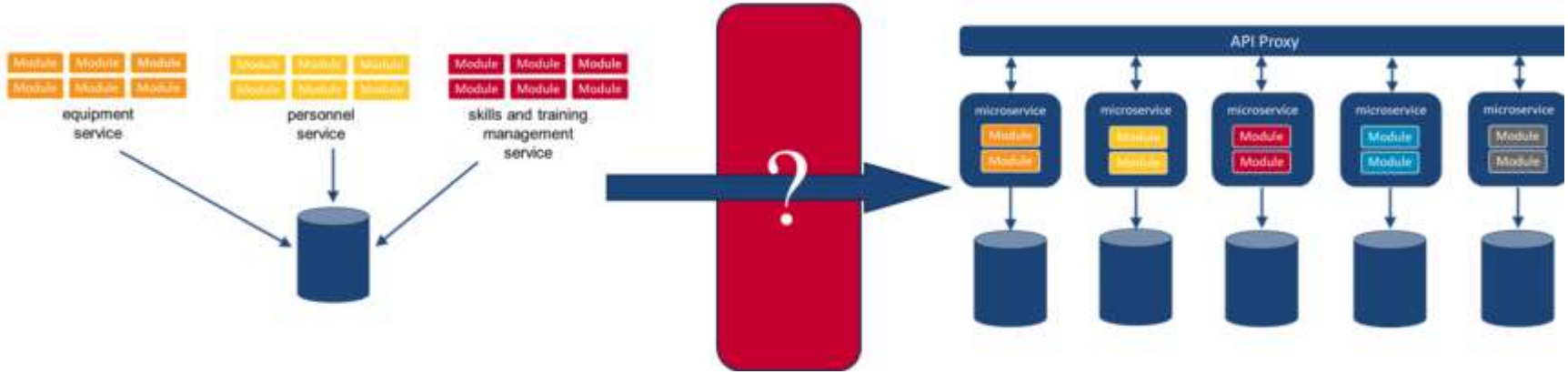
The goal is to identify a small set of objects that should be logically separated in the final system and their key actions.

**5. Create an API for remote user interface**

The remote user interface is designed as the only mode of communication between the system, its components and system users.Services use API to process data and to communicate

**6.Migrate component groups to macroservices**

The key goal at this step is to move component groups into separate projects and make separate deployments.


Module
equipment service
personnel service
skills and training management service
?
API Proxy
microservice

**7.Migrate macroservices to microservices.**

Each microservice maintains its own datastore and performs only a small set of actions on the data objects within that datastore.

**8.Deployment and Testing**

User interfaces should be tested to test migration accuracy. It is checked that there is no relationship with the old architecture.

# Reference


- https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/
- https://blog.javan.co.id/micro-services-versus-monolithic-architecture-what-are-they-e17ddc8d3910
- https://pure.ulster.ac.uk/ws/portalfiles/portal/78494479/Paper_39_Matrix_Clustering_Based_Migration_of_System_Application.pdf

*Teşekkürler*