Automated Microservice Identification: an Approach to Decomposition into a Modular Monolith Architecture

University of Turku
Department of Computing
Master of Science (Tech) Thesis
March 2024
Florian Dejonckheere

UNIVERSITY OF TURKU
Department of Computing

Florian Dejonckheere: Automated Microservice Identification: an Approach to Decomposition into a Modular Monolith Architecture

Master of Science (Tech) Thesis, 18 p., 4 app. p.
Department of Computing
March 2024

The modular monolith architecture emerged in recent years as the harmonization of the monolithic and microservices architectures. The paradigm offers a compromise between modularity, flexibility, and scalability. Many monolithic applications are being migrated to modular monoliths or microservices entirely, to satisfy increasingly complex and volatile business requirements. This process is labour-intensive, slow, and may take months to years for larger codebases. Modularization of a codebase typically requires the developer to have an intimate knowledge of both the application code and domain.

In this thesis, we investigate the modular monolith software architecture, and how modules are typically determined as part of the modularization efforts. We propose an automated solution based on dependency analysis and machine learning algorithms to aid in the identification of module boundaries, and evaluate its effectiveness using a case study. We discuss the results and draw conclusions about the propsed solution.

Keywords: software architecture, monolith, microservices, modular monolith

Contents

List of Figures

List of Tables

List of Acronyms

DSRM Design Science Research Methodology

DSRP Design Science Research Process

SDLC Software Development Life Cycle

1. Introduction

1.1. Scope and goal

This research is centered around three research questions:

Research Question 1: What are the challenges and opportunities of the modular monolith architecture compared to traditional monolithic and microservices architectures?

Research Question 2: What are the existing approaches and tools for automated microservice candidate identification in monolith codebases?

Research Question 3: How can static analysis of source code effectively identify optimal module boundaries in a modular monolith architecture?

To answer the first research question, we will first define the modular monolith architecture, and examine what sets it apart from monolithic and microservices architectures. Then, we will proceed to investigate the merits and drawbacks of the software architecture when applied to an existing codebase.

For the second research question, we will enumerate the existing technologies to aid modularization of monolithic codebases, and choose one automated technology for further examination. *(Automated technology)* will then be implemented for a given use case, and compared to manual modularization efforts in terms of accuracy, efficiency, development velocity. This comparison will help us to answer the third research question.

The goal of this research can be summarized as follows:

1. Investigate the merits and drawbacks of the modular monolith architecture
2. Investigate the use of automated technologies to modularize a monolithic architecture

The proposed solution will add value to the field of software engineering, and will be able to be used as a base for future improvements regarding automated modularization of monolith codebases.

1.2. Motivation

1.3. Methodology

A literature review is conducted to answer the first and second research question. For the first research question, the study aims to find a definition of the modular monolith architecture, and to list the advantages and disadvantages of the architecture based on existing literature. For the second

research question, the state of the art in automated modularization technologies is reviewed and summarized.

The third research question is answered by choosing the most appropriate automated technology, and implementing it for a given use case. The implementation is then evaluated based on quantitative and qualitative metrics, and compared to manual modularization efforts.

Finally, the findings are summarized, and an outlook on future work is given.

For the case study, a Design Science Research Methodology (DSRM) is adopted, which is a research paradigm for information systems research focused at creating and evaluating artifacts. In particular, the research and design of the proposed solution follows the six-step Design Science Research Process (DSRP) model [1]. Their model is based on prior research and is designed to guide researchers through the process of analysis, creation, and evaluation of artifacts.

The six steps of the process are:

1. **Problem identification and motivation**: Research problem statement and justification for existence of a solution.
2. **Objectives of a solution**: Definition of the objectives, derived from the problem statement.
3. **Design and development**: Creation of the artifact.
4. **Demonstration**: Usage of the artifact to demonstrate its effectiveness in solving the problem.
5. **Evaluation**: Observation and measurement of how well the artifact supports a solution to the problem.
6. **Communication**: Transfer of knowledge about the artifact and the problem solution to the relevant audience.

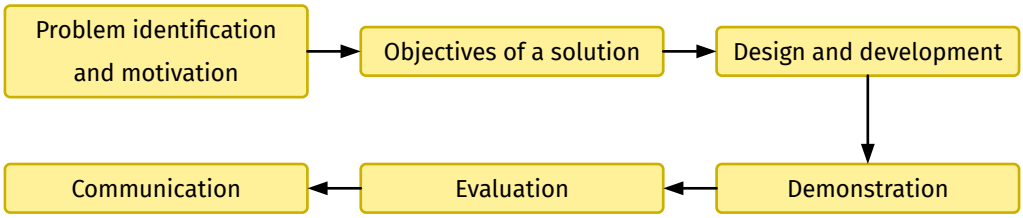

Figure 1: Design Science Research Process (DSRP)

The process is structured sequentially, however the authors suggests that researchers may proceed in a non-linear fashion, and start or stop at any step, depending on the context and requirements of the research.

In this thesis specifically, the DSRP is used to guide the design and development of the automated modularization technology, with a particular focus on the design and development, demonstration, and evaluation steps.

1.4. Outline

The thesis is divided into three parts.

The first part comprises the background and related work. In Chapter 1, the scope and goal of the research is defined, and the research questions are formulated. The stakeholders are identified, and the methodology is explained. Chapter 2 introduces the reader to the research background and necessary concepts. In Chapter 3, the existing literature is reviewed, and the state of the art is presented.

The second part of the thesis, starting with Chapter 4, is dedicated to the first research question. The modular monolith architecture is defined, and its merits and drawbacks are discussed.

The third part aims to solve the second and third research question. Chapter 5 gives an introduction into the automated modularization of monolith codebases, listing the existing technologies. It then continues to focus on one automated technology, *(automated technology)*, and explains its implementation. Chapter 7 applies *(automated technology)* on a given case study, and compares it to manual modularization efforts.

Finally, Chapter 8 summarizes the findings, and gives an outlook on future work.

2. Background

2.1. Monolith architecture

2.2. Modular programming

2.3. Microservice architecture

3. Related work

4. Modular monolith architecture

4.1. Background

4.2. Challenges and opportunities

4.3. Modularization

5. Automated modularization

In this chapter, we investigate the state of the art in automated technologies for modularization of monolith codebases. Using a systematic literature review, we identified and categorized existing literature regarding automated modularization of monolith codebases. We also provided a brief overview of the most relevant approaches and tools.

A systematic literature review is used to identify, evaluate and interpret research literature for a given topic area, or research question [2]. The systematic nature of systematic literature reviews reduces bias through a well-defined sequence of steps to identify and categorize existing literature, although publication bias still has to be considered. Studies directly researching the topic area are called *primary* studies, systematic studies aggregating and summarizing primary studies are called *secondary* studies. *Tertiary* studies are systematic studies aggregating and summarizing secondary studies.

The literature review was conducted using a three-step protocol as defined by Kitchenham & Charters [2]:

	Step	**Activity**
1	Plan	Identify the need for the review, specifying the research questions, and developing a review protocol
2	Conduct	Identification and selection of literature, data extraction and synthesis
3	Report	Evaluation and reporting of the results

Table 1: Systematic literature review process

5.1. Plan

Using the systematic literature review, we answered the following research question:

Research Question 2: What are the existing approaches and tools for automated microservice candidate identification in monolith codebases?

The motivation for the research question is discussed in Chapter 1.

In current literature, several systematic mapping studies related to microservices architecture have been conducted [3], [4], as well as systematic literature reviews related to microservice decomposition . However, in these studies the techniques described are mainly used as an aid for the software architect when identifying microservice candidates. Therefore, we believe that there is a need for

a systematic literature review aimed at summarizing existing literature regarding fully automated techniques for modularization of monolith codebases.

As a search strategy, the following platforms were queried for relevant publications:

1. IEEE Xplore
2. arXiv

Based on a list of relevant topics, we used a combination of related keywords to formulate the search query. The search query was adapted to the specific search syntax of the platform. We refrained from using more generic keywords, such as “architecture” or “design”, as they would yield too many irrelevant results. The topics relevant for the search query are:

- *Architecture*: the architectural styles being discussed in the publications.
  Keywords: microservice, monolith, modular monolith
- *Modularization*: the process of identifying and decomposing modules in a monolith architecture.
  Keywords: service identification, microservice decomposition, monolith modularization
- *Technology*: the technologies, algorithms, or methods for modularization.
  Keywords: automated tool, machine learning, static analysis, dynamic analysis, hybrid analysis

In addition to search queries on the selected platforms, we used snowballing to identify additional relevant publications. Snowballing is a research technique used to find additional publications of interest by following the references of the selected publications.

Based the inclusion/exclusion criteria in Table 2, the results were filtered, and the relevant studies were selected for inclusion in the systematic literature review.

	Criteria
Inclusion	• Title, abstract or keywords include the search terms • Conference papers, research articles, blog posts, or other publications • Publications addressing (semi-)automated technologies, algorithms, or methods
Exclusion	• Publications in languages other than English • Publications not available in full text • Publications using the term "microservice", but not referring to the architectural style • Publications aimed at greenfield[1] or brownfield[2] development of microservice-based systems • Publications published before 2014, as the definition of "microservices" as an architectural style is inconsistent before [4] • Publications addressing manual technologies, algorithms, or methods • Surveys, opinion pieces, or other non-technical publications

Table 2: Inclusion and exclusion criteria

As a final step, the publications were subjected to a validation scan to ensure relevance and quality. To assess the quality, we mainly focused on the technical soundness of the approach or technique described in the publication.

The quality of the publication was assessed based on the following criteria:

- The publication is peer-reviewed or published in a respectable journal
- The publication thoroughly describes the technical aspects of the approach or technique
- The publication includes a validation phase or case study demonstrating the effectiveness of the approach or technique

This step is necessary to ensure that the selected publications are relevant to the research question and that the results are not biased by low-quality publications.

Once a final selection of publications was made, the resulting publications were qualitatively reviewed and categorized based on the type of approach or technique they describe.

[1]Development of new software systems lacking constraints imposed by prior work [5]
[2]Development of new software systems in the presence of legacy software systems [5]

5.2. Conduct

Using the search strategy outlined in the previous section, we queried the selected platforms and found a total of publications.

Platform	**Search results**	**Selected publications**
IEEE Xplore	337	48
Snowballing		
Total	337	48

Table 3: Summary of search results

After applying the inclusion/exclusion criteria, we selected publications for inclusion in the systematic literature review. Of these publications, are primary studies, and are secondary studies. The secondary studies were used to categorize the selected primary studies (if any), and as a starting point for the snowballing process, which resulted in additional publications being included in the systematic literature review. Table 4 lists the selected primary publications, and Table 5 the selected secondary publications.

	Type	Publication
1	Proceedings	[6] M. Saidi, A. Tissaoui, and S. Faiz, *A DDD Approach Towards Automatic Migration To Microservices*. 2023.
2	Proceedings	[7] Z. Yang, S. Wu, and C. Zhang, *A Microservices Identification Approach Based on Problem Frames*. 2022.
3	Proceedings	[8] J. Kazanavičius and D. Mažeika, *Migrating Legacy Software to Microservices Architecture*. 2019.
4	Proceedings	[9] T. Kinoshita and H. Kanuka, *Automated Microservice Decomposition Method as Multi-Objective Optimization*. 2022.
5	Article	[10] X. Zhou and J. Xiong, "Automated Microservice Identification from Design Model". 2022.
6	Proceedings	[11] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, *Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics*. 2020.
7	Article	[12] G. Quattrocchi, D. Cocco, S. Staffa, A. Margara, and G. Cugola, "Cromlech: Semi-Automated Monolith Decomposition Into Microservices". 2024.
8	Proceedings	[13] G. Mazlami, J. Cito, and P. Leitner, *Extraction of Microservices from Monolithic Software Architectures*. 2017.

Table 4: Selected publications (primary studies)

	Type	Publication

Table 5: Selected publications (secondary studies)

5.3. Report

The publications selected for inclusion in the systematic literature review were qualitatively reviewed and categorized in three dimensions.

To begin with, we categorized the publications based on the SDLC artifact they use as input for the microservice candidate identification algorithm. Each artifact category has an associated collection type, either static, dynamic, or hybrid. [14]. Static collection describes a SDLC artifact that was collected without executing the software, while dynamic collection describes a SDLC artifact that

was collected after or during execution of the software. Some publications describe algorithms or techniques that use a combination of SDLC artifacts, which we categorized as hybrid.

Thereafter we categorized the publications based on the algorithm used for microservice candidate identification. The algorithms were subdivided into several classes based on the technique.

Ultimately, the publications were also categorized by the metrics discussed.

5.3.1. SDLC artifact

The identified SDLC artifact categories used as input for the microservice candidate identification algorithm are described in Table 6. The categories are based on Bajaj et al. [14].

Artifact	Type	Publications
Requirements documents and models	Static	[6]–[8]
Design documents	Static	[8], [10], [12]
Codebase	Static	[8], [9], [13]
Execution data	Dynamic	[11]

Table 6: SDLC artifact categories

In software engineering, requirements documents and models are used to formally describe the requirements of a software system following the specification of the business or stakeholder requirements [15]. They include functional and non-functional requirements, use cases, user stories, and business process models. Approaches using requirements documents and models as input for the microservice candidate identification algorithm often times need to pre-process the data to extract the relevant information, as the documents are not intended to be directly read by a machine. In many cases, requirements documents and models are no longer available for legacy systems, which makes this approach less suitable for automated microservice identification.

Saidi et al. [6] use domain-driven design techniques to extract functional dependencies from the software design as starting point in microservice identification. Yang et al. [7] tackle requirements engineering using problem frames. Zhou & Xiong [10] use schematic design documents in XML format as input for the algorithm.

Design documents created by software architects are machine-readable representations of the software system. They are deliverables produced during the design phase of the software development life cycle. Design documents include API specifications, UML diagrams (such as class diagrams and sequence diagrams), and entity-relationship diagrams. Techniques using design documents either

use a domain-driven approach, or a data-driven approach. Domain-driven approaches use domain-specific knowledge to identify microservice candidates, while data-driven approaches use knowledge about data storage and data flow to identify microservice candidates. Similar to requirements documents and models, design documents are often not available for legacy systems, although some design documents can be reconstructed from the software system (e.g., reverse engineering entity-relationship diagrams from the database schema).

Kazanavičius & Mažeika [8] enumerate three decomposition methods depending on the requirements of the business: (1) code based, (2) business domain based, and (3) data storage based. The authors suggest a code based approach as suitable for automatic decomposition of monoliths.

Quattrocchi et al. [12] takes a different approach to the problem, using a data-driven approach combined with a domain-driven approach. Software architects describe the software system using a custom architecture description language, and the tool developed by the authors is able to identify microservice candidates. The tool can be persuaded to generate different, more efficient decompositions when given domain-driven requirements.

A third category of SDLC artifacts is the executable codebase of the software system. (Static) analysis on the source code of the application can be used to identify microservice candidates. Furthermore, also indirect information, such as revision history, can be used to complement the decomposition approach.

For example, Kinoshita & Kanuka [9] consider object-oriented classes as units of business capability, as do Mazlami et al. [13]. The authors of the latter paper remark that increasing the granularity from classes to methods and functions has the potential to improve the quality of the decomposition.

The decomposition process described by Mazlami et al. [13] also uses the revision history of the software system to identify suitable microservice candidates.

Finally, data can also be collected during the runtime of the software system. Execution data includes log files, execution traces, and performance metrics.

An example of an approach using execution traces as input is described by Zhang et al. [11]. Using software probes inserted into the bytecode of software systems, the authors are able to monitor execution paths. Additionally, various metrics such as processor time and memory usage are collected.

5.3.2. Algorithm

The identified classes of microservice candidate identification algorithms are described in Table 7.

Type	Algorithms	Publications
Clustering	K-Means, Kruskal	[13]
Evolutionary algorithms	NSGA-II	[9]–[11]
Search-based		[12]

Table 7: Microservice identification algorithm

Publications using a clustering approach to identify microservice candidates typically collect static information from the software system, and represent it as a directed graph. The graph exposes the relationship between the classes, modules, or components. The nodes of the graph represent the classes, modules, or components, and the edges the function or method calls between them. Often the edges are weighted, representing the frequency or cost of the calls. Using this information, the graph is then divided into several clusters, each indicating a microservice candidate. Typical clustering algorithms used for this purpose are K-Means, Kruskal's algorithm [16], and agglomerative clustering.

Mazlami et al. [13] use Kruskal's algorithm [16] to isolate microservice candidates.

The second class of algorithms identified in the literature is evolutionary algorithms. Evolutionary algorithms, and in particular genetic algorithms, are algorithms aimed at solving optimization problems by borrowing techniques from natural selection and genetics. Such algorithms typically operate iteratively, selecting the best solutions from a population at each iteration (called a generation), and then combining the selected solutions to create new combinations for the next generation. The process is then repeated until certain criteria are met, for example a maximum number of generations, convergence of the population, or a quality indicator.

Examples of publications using Non-Dominated Sorting Algorithm II (NGSA-II) as multi-objective optimization algorithm to identify microservice candidates are Zhou & Xiong [10], Kinoshita & Kanuka [9], and Zhang et al. [11]

The authors of Quattrocchi et al. [12] incorporated a linear optimization solver in their tool to identify microservice candidates.

5.3.3. Metrics

The quality metrics used to determine the performance and efficacy of the algorithm are described in Table 8.

Metric	Publications
Cohesion	
Coupling	

Table 8: Algorithm metrics

6. Proposed solution

7. Case study

7.1. Background

7.2. Experimental setup

7.3. Evaluation and results

7.4. Discussion

8. Conclusion

8.1. Future work

References

[1] K. Peffers, M. Rothenberger, T. Tuunanen, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research”, vol. 24, no. 3. 2007.

[2] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering”. 2007.

[3] N. Alshuqayran, N. Ali, and R. Evans, *A Systematic Mapping Study in Microservice Architecture*. 2016.

[4] C. Pahl and P. Jamshidi, *Microservices: A Systematic Mapping Study*. 2016.

[5] R. M. Gupta, *Project Management*. Prentice-Hall of India Pvt.Limited, 2011.

[6] M. Saidi, A. Tissaoui, and S. Faiz, *A DDD Approach Towards Automatic Migration To Microservices*. 2023.

[7] Z. Yang, S. Wu, and C. Zhang, *A Microservices Identification Approach Based on Problem Frames*. 2022.

[8] J. Kazanavičius and D. Mažeika, *Migrating Legacy Software to Microservices Architecture*. 2019.

[9] T. Kinoshita and H. Kanuka, *Automated Microservice Decomposition Method as Multi-Objective Optimization*. 2022.

[10] X. Zhou and J. Xiong, “Automated Microservice Identification from Design Model”. 2022.

[11] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, *Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics*. 2020.

[12] G. Quattrocchi, D. Cocco, S. Staffa, A. Margara, and G. Cugola, “Cromlech: Semi-Automated Monolith Decomposition Into Microservices”. 2024.

[13] G. Mazlami, J. Cito, and P. Leitner, *Extraction of Microservices from Monolithic Software Architectures*. 2017.

[14] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, “A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts”. 2021.

[15] “IEEE Guide for Software Requirements Specifications”. 1984.

[16] J. Kleinberg and É. Tardos, *Algorithm Design*. Pearson Education, 2006.

[17] B. Andrade, S. Santos, and A. R. Silva, *A Comparison of Static and Dynamic Analysis to Identify Microservices in Monolith Systems*. 2023.

A Examples

Examples First page

```ruby
class Example
  def initialize
    @text = "Hello world"
  end

  def say_hello
    puts @text
  end
end

example = Example.new

example.say_hello
# => Hello world
```

Listing 1: Ruby code example

Examples Second page metadata

B Figures

Figures First page

Figures Second page metadata