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

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, 24 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

AST	Abstract Syntax Tree
BPMN	Business Process Model and Notation
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

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. 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

The resulting search query can be expressed as follows:

```
1    (('microservices' IN title OR abstract) OR
2     ('monolith' IN title OR abstract))
3  AND
4    (('decomposition' IN title OR abstract) OR
5     ('identification' IN title OR abstract))
6  AND
7    ('automated' IN title OR abstract)
```

The search query was adapted to the specific search syntax of the platform.

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 337 publications.

Platform	Search results	Selected publications
IEEE Xplore	337	34
arXiv		
Snowballing		
Total	337	34

Table 3: Summary of search results

After applying the inclusion/exclusion criteria, we selected 34 publications for inclusion in the systematic literature review. Of these publications, 29 are primary studies, and 5 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] D. Escobar *et al.*, "Towards the Understanding and Evolution of Monolithic Applications as Microservices", in *2016 XLII Latin American Computing Conference (CLEI)*, Valparaíso, Chile: IEEE, 2016. doi: 10.1109/CLEI.2016.7833410.
2	Proceedings	[7] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures", in *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA: IEEE, 2017. doi: 10.1109/ICWS.2017.61.
3	Proceedings	[8] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting Candidates of Microservices from Monolithic Application Code", in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Nara, Japan: IEEE, 2018. doi: 10.1109/APSEC.2018.00072.

4	Proceedings	[9] M. J. Amiri, "Object-Aware Identification of Microservices", in *2018 IEEE International Conference on Services Computing (SCC)*, San Francisco, CA, USA: IEEE, 2018. doi: 10.1109/SCC.2018.00042.
5	Proceedings	[10] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics", in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil: IEEE, 2020. doi: 10.1109/ICSA47634.2020.00021.
6	Proceedings	[11] O. Al-Debagy and P. Martinek, "Extracting Microservices Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach", in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, Budapest, Hungary: IEEE, 2020. doi: 10.1109/SoSE50414.2020.9130466.
7	Proceedings	[12] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From Monolithic Architecture Style to Microservice One Based on a Semi-Automatic Approach", in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil: IEEE, 2020. doi: 10.1109/ICSA47634.2020.00023.
8	Proceedings	[13] L. Carvalho *et al.*, "On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices", in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Adelaide, Australia: IEEE, 2020. doi: 10.1109/ICSME46990.2020.00060.
9	Proceedings	[14] F.-D. Eyitemi and S. Reiff-Marganiec, "System Decomposition to Optimize Functionality Distribution in Microservices with Rule Based Approach", in *2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE)*, Oxford, United Kingdom: IEEE, 2020. doi: 10.1109/SOSE49046.2020.00015.
10	Proceedings	[15] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki, "Towards an Automatic Identification of Microservices from Business Processes", in *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative*

		Enterprises (WETICE), Bayonne, France: IEEE, 2020. doi: 10.1109/ WETICE49692.2020.00017.
11	Proceedings	[16] C. Bandara and I. Perera, "Transforming Monolithic Systems to Microservices - An Analysis Toolkit for Legacy Code Evaluation", in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, Colombo, Sri Lanka: IEEE, 2020. doi: 10.1109/ ICTer51097.2020.9325443.
12	Proceedings	[17] G. Filippone, M. Autili, F. Rossi, and M. Tivoli, "Migration of Monoliths through the Synthesis of Microservices Using Combinatorial Optimization", in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Wuhan, China: IEEE, 2021. doi: 10.1109/ISSREW53611.2021.00056.
13	Proceedings	[18] S. Agarwal *et al.*, "Monolith to Microservice Candidates Using Business Functionality Inference". 2021.
14	Article	[19] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems Based on Execution Traces", *IEEE Transactions on Software Engineering*, vol. 47, no. 5, 2021, doi: 10.1109/TSE.2019.2910531.
15	Proceedings	[20] Z. Yang, S. Wu, and C. Zhang, "A Microservices Identification Approach Based on Problem Frames", in *2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI)*, Xiamen, China: IEEE, 2022. doi: 10.1109/SEAI55746.2022.9832106.
16	Proceedings	[21] T. Kinoshita and H. Kanuka, "Automated Microservice Decomposition Method as Multi-Objective Optimization", in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA: IEEE, 2022. doi: 10.1109/ICSA-C54293.2022.00028.
17	Article	[22] X. Zhou and J. Xiong, "Automated Microservice Identification from Design Model", 2022.
18	Proceedings	[23] S. Wu and C. Zhang, "Identification of Microservices through Processed Dynamic Traces and Static Calls", in *2022 3rd International*

		Conference on Computer Science and Management Technology (ICCSMT), Shanghai, China: IEEE, 2022. doi: 10.1109/ICCSMT58129.2022.00071.
19	Proceedings	[24] P. Zaragoza, A.-D. Seriai, A. Seriai, A. Shatnawi, and M. Derras, "Leveraging the Layered Architecture for Microservice Recovery", in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, Honolulu, HI, USA: IEEE, 2022. doi: 10.1109/ICSA53651.2022.00021.
20	Article	[25] S. Santos and A. R. Silva, "Microservices Identification in Monolith Systems: Functionality Redesign Complexity and Evaluation of Similarity Measures", *Journal of Web Engineering*, 2022, doi: 10.13052/jwe1540-9589.2158.
21	Proceedings	[26] S.-P. Ma, T.-W. Lu, and C.-C. Li, "Migrating Monoliths to Microservices Based on the Analysis of Database Access Requests", in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Newark, CA, USA: IEEE, 2022. doi: 10.1109/SOSE55356.2022.00008.
22	Proceedings	[27] Y. Romani, O. Tibermacine, and C. Tibermacine, "Towards Migrating Legacy Software Systems to Microservice-based Architectures: A Data-Centric Process for Microservice Identification", in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA: IEEE, 2022. doi: 10.1109/ICSA-C54293.2022.00010.
23	Proceedings	[28] M. Saidi, A. Tissaoui, and S. Faiz, "A DDD Approach Towards Automatic Migration To Microservices", in *2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, Hammamet, Tunisia: IEEE, 2023. doi: 10.1109/IC_ASET58101.2023.10150522.
24	Proceedings	[29] G. Filippone, N. Qaisar Mehmood, M. Autili, F. Rossi, and M. Tivoli, "From Monolithic to Microservice Architecture: An Automated Approach Based on Graph Clustering and Combinatorial Optimization", in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, L'Aquila, Italy: IEEE, 2023. doi: 10.1109/ICSA56044.2023.00013.
25	Proceedings	[30] J. Lourenço and A. R. Silva, "Monolith Development History for Microservices Identification: A Comparative Analysis", in *2023 IEEE International Conference on Web Services (ICWS)*, Chicago, IL, USA: IEEE, 2023. doi: 10.1109/ICWS60048.2023.00019.

26	Proceedings	[31] T. Lopes and A. R. Silva, "Monolith Microservices Identification: Towards An Extensible Multiple Strategy Tool", in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, L'Aquila, Italy: IEEE, 2023. doi: 10.1109/ICSA-C57050.2023.00034.
27	Article	[32] J. Hao, J. Zhao, and Y. Li, "Research on Decompostion Method of Relational Database Oriented to Microservice Refactoring", 2023.
28	Proceedings	[33] Y. Li, Y. Zhang, Y. Yang, W. Wang, and Y. Yin, "RM2MS: A Tool for Automatic Identification of Microservices from Requirements Models", in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Västerås, Sweden: IEEE, 2023. doi: 10.1109/MODELS-C59198.2023.00018.
29	Article	[34] G. Quattrocchi, D. Cocco, S. Staffa, A. Margara, and G. Cugola, "Cromlech: Semi-Automated Monolith Decomposition Into Microservices", *IEEE Transactions on Services Computing*, 2024, doi: 10.1109/TSC.2024.3354457.

Table 4: Selected publications (primary studies)

	Type	Publication
1	Proceedings	[35] J. Kazanavicius and D. Mazeika, "Migrating Legacy Software to Microservices Architecture", in *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Vilnius, Lithuania: IEEE, 2019. doi: 10.1109/eStream.2019.8732170.
2	Proceedings	[36] R. A. Schmidt and M. Thiry, "Microservices Identification Strategies : A Review Focused on Model-Driven Engineering and Domain Driven Design Approaches", in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, Sevilla, Spain: IEEE, 2020. doi: 10.23919/CISTI49556.2020.9141150.
3	Article	[37] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, "A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts", *Journal of Web Engineering*, 2021, doi: 10.13052/jwe1540-9589.20312.
4	Article	[38] Y. Abgaz *et al.*, "Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review". 2023.

5	Article	[39] I. Oumoussa and R. Saidi, "Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review", vol. 12, 2024.

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 Software Development Life Cycle (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. [37]. 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. [37].

Artifact	Type	Publications
Requirements documents and models	Static	[9], [15], [20], [28], [33]
Design documents	Static	[11], [22], [34]
Code	Static	[6]–[8], [12], [13], [16]–[18], [21], [23]–[25], [27], [29], [30], [32]
Execution data	Dynamic	[10], [13], [14], [19], [23], [26], [30], [32]

Table 6: SDLC artifact categories

5.3.1.1. Requirements documents and models

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 [40]. 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 documents to extract the relevant information, as they are not intended to be directly read by a machine. In many cases, requirements documents and models for legacy systems are no longer available or outdated, which makes this approach less suitable for automated microservice identification.

Amiri [9] and Daoud et al. [15] model a software system as a set of business process using the industry standard Business Process Model and Notation (BPMN), using the machine-readable XML representation as input for the algorithm. Yang et al. [20] tackle requirements engineering using problem frames.

Some approaches use schematic requirements documents in XML format as input for the algorithm, as described by Zhang et al. [10] and Saidi et al. [28]. The latter use domain-driven design techniques to extract functional dependencies from the software design as starting point in microservice identification. Li et al. [33] employ an intermediate format containing a precise definition of business functionality, generated from validated requirements documents.

5.3.1.2. Design documents

Design documents created by software architects are machine-readable representations of the software system. They describe the software functionalities in detail and are used to guide the implementation of the software system. 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 for legacy systems are often not available or outdated, although some design documents can be reconstructed from the software system (e.g., reverse engineering entity-relationship diagrams from the database schema).

For example, Al-Debagy & Martinek [11] propose a data-driven method based on the analysis of the software system's external API, specified in the OpenAPI format. The method extracts the information from the specification and converts it into vector representation for further processing.

Zhou & Xiong [22] use readily available design documents as well, in the form of UML class diagrams, use cases, and object sequence diagrams as starting point for the microservice identification algorithm.

Quattrocchi et al. [34] 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 prompted to generate different, more efficient decompositions when given additional domain-driven requirements.

5.3.1.3. Code

A third category of SDLC artifacts is the executable code of the software system. This can be the source code of the software system, or a binary distribution (e.g. a JAR file).

As the source code of the software system is the most detailed representation of how the software system works, it is most often used as input for the microservice candidate identification algorithm. The source code can be analyzed using static analysis (i.e., without executing the software system), dynamic analysis (i.e., during the execution of the software system or test suite), or a combination of both. Dynamic analysis has the advantage that it can be used if the source code is not available.

Additionally, the revision history of the source code can also be used as source for valuable information about the behaviour of the software system. For example, Mazlami et al. [7] use both the structure of the source code, as well as the revision history to drive the identification algorithm.

Escobar et al. [6] use the source code of the software system to construct an Abstract Syntax Tree (AST), and map the dependencies between the business and data layer. Kamimura et al. [8] use a more data-driven approach, and statically trace data access calls in the source code.

Several publications [12] construct a dependency graph from Java source code, and use the graph as input for a clustering algorithm.

[16] map object-oriented classes in the source code to specific microservices, although the list of microservices has to be specified beforehand.

Most of the publications tracing dependencies between classes (or modules) do this at the level of the classes (or modules). As Mazlami et al. [7] remarks, using a more granular approach at the level of

17

methods (or functions) and attributes has the potential to improve the quality of the decomposition. [13] use a more granular approach, identifying dependencies between methods in the source code.

5.3.1.4. Execution

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 Jin et al. [19]. 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	Example algorithms	Publications
Machine Learning	K-Means, Hierarchical Agglomerative Clustering, Affinity Propagation	[27], [23], [32], [28], [16], [15], [11], [25], [24], [30], [26]
Evolutionary algorithms	NSGA-II, NSGA-III	[10], [19], [22], [33], [21], [13], [9]
Graph algorithms	Kruskal, Louvain method	[20], [7], [29]
Other algorithms	Linear optimization, custom algorithms	[12], [17], [34], [8], [6], [18], [14], [31]

Table 7: Microservice identification algorithm

5.3.2.1. Machine Learning

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 [41], and agglomerative clustering.

Mazlami et al. [7] use Kruskal's algorithm [41] to isolate microservice candidates.

5.3.2.2. Evolutionary algorithms

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 [22], Kinoshita & Kanuka [21], and Zhang et al. [10]

The authors of Quattrocchi et al. [34] incorporated a linear optimization solver in their tool to identify microservice candidates.

5.3.2.3. Graph algorithms

5.3.2.4. Other algorithms

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	[9]–[13], [15]–[17], [19], [22], [24], [25], [29], [30], [33], [34]
Coupling	[9], [10], [12], [13], [15]–[19], [22], [25], [29], [33]
Network overhead	[13], [17], [22], [34]
Complexity	[11], [25], [30]
CPU and memory usage	[10], [34]
Modularity	[16], [33]
Similarity	[26], [30]
Other metrics	[33], [29], [7], [30], [7], [8], [32], [12], [29], [20], [23], [28], [21], [31]
No metrics	[14], [27]

Table 8: Quality metrics

5.3.3.1. Cohesion and coupling

5.3.3.2. Network overhead

5.3.3.3. Complexity

5.3.3.4. CPU and memory usage

5.3.3.5. Modularity

5.3.3.6. Similarity

5.3.3.7. Other metrics

5.3.4. Conclusion

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] D. Escobar *et al.*, "Towards the Understanding and Evolution of Monolithic Applications as Microservices", in *2016 XLII Latin American Computing Conference (CLEI)*, Valparaíso, Chile: IEEE, 2016. doi: 10.1109/CLEI.2016.7833410.

[7] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures", in *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA: IEEE, 2017. doi: 10.1109/ICWS.2017.61.

[8] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting Candidates of Microservices from Monolithic Application Code", in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Nara, Japan: IEEE, 2018. doi: 10.1109/APSEC.2018.00072.

[9] M. J. Amiri, "Object-Aware Identification of Microservices", in *2018 IEEE International Conference on Services Computing (SCC)*, San Francisco, CA, USA: IEEE, 2018. doi: 10.1109/SCC.2018.00042.

[10] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics", in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil: IEEE, 2020. doi: 10.1109/ICSA47634.2020.00021.

[11] O. Al-Debagy and P. Martinek, "Extracting Microservices Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach", in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, Budapest, Hungary: IEEE, 2020. doi: 10.1109/SoSE50414.2020.9130466.

[12] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From Monolithic Architecture Style to Microservice One Based on a Semi-Automatic Approach", in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil: IEEE, 2020. doi: 10.1109/ICSA47634.2020.00023.

[13] L. Carvalho *et al.*, "On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices", in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Adelaide, Australia: IEEE, 2020. doi: 10.1109/ICSME46990.2020.00060.

[14] F.-D. Eyitemi and S. Reiff-Marganiec, "System Decomposition to Optimize Functionality Distribution in Microservices with Rule Based Approach", in *2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE)*, Oxford, United Kingdom: IEEE, 2020. doi: 10.1109/SOSE49046.2020.00015.

[15] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki, "Towards an Automatic Identification of Microservices from Business Processes", in *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Bayonne, France: IEEE, 2020. doi: 10.1109/WETICE49692.2020.00017.

[16] C. Bandara and I. Perera, "Transforming Monolithic Systems to Microservices - An Analysis Toolkit for Legacy Code Evaluation", in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, Colombo, Sri Lanka: IEEE, 2020. doi: 10.1109/ICTer51097.2020.9325443.

[17] G. Filippone, M. Autili, F. Rossi, and M. Tivoli, "Migration of Monoliths through the Synthesis of Microservices Using Combinatorial Optimization", in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Wuhan, China: IEEE, 2021. doi: 10.1109/ISSREW53611.2021.00056.

[18] S. Agarwal *et al.*, "Monolith to Microservice Candidates Using Business Functionality Inference". 2021.

[19] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems Based on Execution Traces", *IEEE Transactions on Software Engineering*, vol. 47, no. 5, 2021, doi: 10.1109/TSE.2019.2910531.

[20] Z. Yang, S. Wu, and C. Zhang, "A Microservices Identification Approach Based on Problem Frames", in *2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI)*, Xiamen, China: IEEE, 2022. doi: 10.1109/SEAI55746.2022.9832106.

[21] T. Kinoshita and H. Kanuka, "Automated Microservice Decomposition Method as Multi-Objective Optimization", in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA: IEEE, 2022. doi: 10.1109/ICSA-C54293.2022.00028.

[22] X. Zhou and J. Xiong, "Automated Microservice Identification from Design Model", 2022.

[23] S. Wu and C. Zhang, "Identification of Microservices through Processed Dynamic Traces and Static Calls", in *2022 3rd International Conference on Computer Science and Management Technology (ICCSMT)*, Shanghai, China: IEEE, 2022. doi: 10.1109/ICCSMT58129.2022.00071.

[24] P. Zaragoza, A.-D. Seriai, A. Seriai, A. Shatnawi, and M. Derras, "Leveraging the Layered Architecture for Microservice Recovery", in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, Honolulu, HI, USA: IEEE, 2022. doi: 10.1109/ICSA53651.2022.00021.

[25] S. Santos and A. R. Silva, "Microservices Identification in Monolith Systems: Functionality Redesign Complexity and Evaluation of Similarity Measures", *Journal of Web Engineering*, 2022, doi: 10.13052/jwe1540-9589.2158.

[26] S.-P. Ma, T.-W. Lu, and C.-C. Li, "Migrating Monoliths to Microservices Based on the Analysis of Database Access Requests", in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Newark, CA, USA: IEEE, 2022. doi: 10.1109/SOSE55356.2022.00008.

[27] Y. Romani, O. Tibermacine, and C. Tibermacine, "Towards Migrating Legacy Software Systems to Microservice-based Architectures: A Data-Centric Process for Microservice Identification", in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA: IEEE, 2022. doi: 10.1109/ICSA-C54293.2022.00010.

[28] M. Saidi, A. Tissaoui, and S. Faiz, "A DDD Approach Towards Automatic Migration To Microservices", in *2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, Hammamet, Tunisia: IEEE, 2023. doi: 10.1109/IC_ASET58101.2023.10150522.

[29] G. Filippone, N. Qaisar Mehmood, M. Autili, F. Rossi, and M. Tivoli, "From Monolithic to Microservice Architecture: An Automated Approach Based on Graph Clustering and Combinatorial Optimization", in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, L'Aquila, Italy: IEEE, 2023. doi: 10.1109/ICSA56044.2023.00013.

[30] J. Lourenço and A. R. Silva, "Monolith Development History for Microservices Identification: A Comparative Analysis", in *2023 IEEE International Conference on Web Services (ICWS)*, Chicago, IL, USA: IEEE, 2023. doi: 10.1109/ICWS60048.2023.00019.

[31] T. Lopes and A. R. Silva, "Monolith Microservices Identification: Towards An Extensible Multiple Strategy Tool", in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, L'Aquila, Italy: IEEE, 2023. doi: 10.1109/ICSA-C57050.2023.00034.

[32] J. Hao, J. Zhao, and Y. Li, "Research on Decompostion Method of Relational Database Oriented to Microservice Refactoring", 2023.

[33] Y. Li, Y. Zhang, Y. Yang, W. Wang, and Y. Yin, "RM2MS: A Tool for Automatic Identification of Microservices from Requirements Models", in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Västerås, Sweden: IEEE, 2023. doi: 10.1109/MODELS-C59198.2023.00018.

[34] G. Quattrocchi, D. Cocco, S. Staffa, A. Margara, and G. Cugola, "Cromlech: Semi-Automated Monolith Decomposition Into Microservices", *IEEE Transactions on Services Computing*, 2024, doi: 10.1109/TSC.2024.3354457.

[35] J. Kazanavicius and D. Mazeika, "Migrating Legacy Software to Microservices Architecture", in *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Vilnius, Lithuania: IEEE, 2019. doi: 10.1109/eStream.2019.8732170.

[36] R. A. Schmidt and M. Thiry, "Microservices Identification Strategies : A Review Focused on Model-Driven Engineering and Domain Driven Design Approaches", in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, Sevilla, Spain: IEEE, 2020. doi: 10.23919/CISTI49556.2020.9141150.

[37] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, "A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts", *Journal of Web Engineering*, 2021, doi: 10.13052/jwe1540-9589.20312.

[38] Y. Abgaz *et al.*, "Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review". 2023.

[39] I. Oumoussa and R. Saidi, "Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review", vol. 12, 2024.

[40] "IEEE Guide for Software Requirements Specifications". 1984.

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

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