TÉCNICO LISBOA

Comparing Structure and Accesses Monolith Representations in Mono2Micro

Rodrigo Gonçalves dos Santos

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Antonio Manuel Ferreira Rito da Silva

Examination Committee

Chairperson: Prof. Andreas Miroslaus Wichert
Supervisor: Prof. Antonio Manuel Ferreira Rito da Silva
Member of the Committee: Prof. António Paulo Teles de Menezes Correia Leitão

October 2024

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to begin by expressing my deepest gratitude to my parents and my sister. Their love and support have been the foundation that allowed me to pursue this education, and without them, this journey would not have been possible. A heartfelt thanks to my grandparents for all they have done and continue to do for me, for their comfort and encouragement during challenging moments. To my aunt Bela, thank you for your invaluable help throughout my entire life.

I am also immensely grateful to my cousins, Cristina and António, for giving me a home during these past years of university and always making me feel welcome and loved. To all my family, thank you for being there when I needed you most, your unwavering support has never failed me.

To my girlfriend Léa, I thank you for your immeasurable love and support, for being my shelter through it all.

A special thanks to my close friends David and Joana, who have been with me since the very beginning of this journey at Técnico. And to Hugo, my friend since forever, thank you for always being by my side. I truly don't know where I'd be today without you.

To José, I am deeply grateful not only for reviewing this thesis but for your friendship and support from day one.

Lastly, to my basketball team, thank you for being a space where I always felt welcome, and for being there on the tough days when I needed you most.

This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) through projects UIDB/50021/2020 (INESC-ID) and PTDC/CCI-COM/2156/2021 (DACOMICO).

Abstract

The growing popularity of microservices architecture has led to interest in migrating monolithic systems to the microservices model. In this study, we focus on improving an existing tool designed to identify microservices candidates within monoliths. The primary objective is to extend the tool's capabilities by incorporating a novel type of monolith representation, specifically focusing on the structural aspects of the monolith.

Our work utilizes a collector based on structural analysis. Our objective is to conduct a thorough evaluation of this new collector and its similarity measures, by using the extended pipeline, to compare the microservices candidates produced against expert decompositions and decompositions generated using previous collectors. The evaluation is done on a set of monoliths used in similar studies of the state of the art.

This research adds to ongoing work on strategies for migration of monoliths to microservices. Provides valuable insight into the effectiveness of various monolith representation techniques and their impact on the identification of candidate microservices. The results of our evaluation shed light on the strengths and limitations of the extended pipeline, providing valuable information on how different kinds of monolith representations might affect the results of candidate decompositions.

Keywords

Microservices Architecture, Microservices Identification, Software Architecture, Monolith Decomposition, Structural Analysis

Resumo

A crescente popularidade da arquitetura de microserviços tem levado ao interesse em migrar sistemas monolíticos para o modelo de microserviços. Neste estudo, focamo-nos em melhorar uma ferramenta existente projetada para identificar candidatos a microserviços dentro de monólitos. O objetivo principal é expandir as capacidades da ferramenta, incorporando um novo tipo de representação do monólito, especificamente com foco nos aspetos estruturais do monólito.

O nosso trabalho utiliza um coletor baseado em análise estrutural. O objetivo é realizar uma avaliação detalhada deste novo coletor e das suas medidas de similaridade, utilizando a ferramenta estendida, para comparar os candidatos a microserviços produzidos com decomposições feitas por especialistas e com as decomposições geradas por coletores anteriores. A avaliação é feita num conjunto de monólitos utilizados em estudos semelhantes do estado da arte.

Esta investigação contribui para o trabalho em curso sobre estratégias de migração de monólitos para microserviços. Fornece informações valiosas sobre a eficácia de várias técnicas de representação de monólitos e o seu impacto na identificação de candidatos a microserviços. Os resultados da nossa avaliação destacam os pontos fortes e limitações da ferramenta estendida, fornecendo informações úteis sobre como diferentes tipos de representações de monólitos podem afetar os resultados das decomposições de candidatos.

Palavras Chave

Arquitetura de Microsserviços, Identificação de Microsserviços, Arquitetura de Software, Decomposição de Monólitos, Análise Estrutural

Contents

List of Figures

List of Tables

Acronyms

URI	Uniform Resource Identifier
DFD	Dataflow Diagram
UML	Unified Modeling Language
ERD	Entity-Relationship Diagrams
MIC	Model-based Information Collection
CIC	Code-based Information Collection
AST	Abstract Syntaxe Tree
LIC	Log-based Information Collection
VIC	Version-based Information Collection
DDD	Domain-Driven Design
TP	True Positives
FP	False Positives
TN	True Negatives
FN	False Negatives

1

Introduction

Contents

1.1 Introduction

In the dynamic landscape of modern software architecture, the quest for agility, scalability, and maintainability has led to a paradigm shift — a departure from monolithic systems towards the embrace of microservices. As organizations deal with the complexities of migrating from monoliths to microservices, a fundamental challenge emerges: How can we systematically identify and articulate the boundaries of microservices within the intricate structures of existing monolithic systems?

Currently, there are many different approaches to identify microservices within a monolith architecture system. These approaches can vary in many different aspects, for instance, the way they collect data within a monolith, how this data is processed, or how to come up with the best possible decomposition.

Recent work [1, 2] has developed an extensible tool that supports multiple strategies for the different stages of the microservice identification process in monolith systems. In our work, we are looking to extend this tool with a new feature. As stated above, the process of identifying microservices can differ

in many aspects, from the collection of monolith information to the generation of the decomposition. We aim to enable a new monolith representation, focusing specifically on its structural aspects. This representation contains the entities of the monolith and, for each entity, the names and types of its fields.

Inspired by this goal, a set of research questions emerged, and our aim is to address them comprehensively. Research questions are as follows:

- What is the effort necessary to add a new collector to the existing tool, considering that the generated decomposition is a cluster decomposition of domain entities?
- Are the decompositions obtained using the monolith structural representation statistically different from the decompositions obtained using the monolith sequences of accesses representation?
- How does an expert evaluate the best decompositions for each of the quality metrics?

To address these inquiries, we plan to carry out a thorough investigation using various codebases to conduct a comprehensive study. Our approach involves delving into the intricacies of a novel monolith representation and exploring its impact on the final decomposition. In addition, we will examine carefully how variations in metrics and similarity measures influence results. By employing statistical methods, our aim is to gain valuable insights into the dynamics of microservice identification within monolithic systems.

2

Background

Contents

2.1 Microservices

Microservice architecture [3] in software engineering refers to an architectural design that defines an application as a set of small, self-sufficient services communicating via lightweight protocols. A microservice is a compact application capable of independent deployment, scaling, and testing, dedicated to a single, specific task.

The microservice architecture style has been increasingly adopted over the years. Major companies such as Netflix and Amazon [4] have migrated their monolith code due to highly desirable advantages such as:

- **Scalability** – Since each microservice operates independently, it can be scaled individually without affecting the rest of the system. This allows for more efficient resource allocation, where only the

services experiencing high demand need additional resources, unlike monolithic systems where the entire application must be scaled.

- **Maintainability** – The decoupling of services allows teams to work on individual components without affecting the entire application. This modularity makes it easier to isolate bugs, apply updates, and introduce new features in a specific service without impacting the broader system.
- **Reusability** – Microservices can be reused across different projects because they are designed to perform a single, well-defined task. This modular approach allows for components to be repurposed in different contexts, reducing development time and avoiding redundancy.
- **Availability** – By isolating services, the failure of one microservice does not bring down the entire system. Other services can continue to operate, increasing the overall availability and fault tolerance of the system.

2.2 Pipeline

Previous work [1, 2] proposes a pipeline for the identification of microservices in monolith systems. Figure 2.1 describes the various stages that facilitate the identification of microservices.

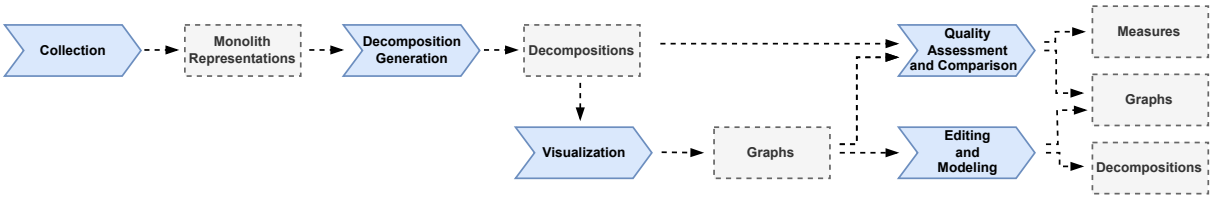

Figure 2.1: Pipeline Stages for Identification of Microservices in Monoliths

The identification pipeline begins with a monolith as input. The first stage is the *Collection*, which involves analyzing the monolith using methods such as static and dynamic analysis. The primary objective of this stage is to generate the representations of the monolith.

The second stage of the pipeline is the *Decomposition Generation*. This stage uses the monolith representations to obtain candidate microservice decompositions by applying an automatic algorithm. The most common algorithms used in this step are clustering algorithms and community detection algorithms. Several similarity criteria between the elements in the monolith representations need to be defined to feed the algorithms, such as accesses similarity (associates domain entities that are accessed by the same functionalities), semantic similarity (connects domain entities based on their textual similarity), author similarity (links domain entities that are changed by the same developers, and change similarity (relates domain entities that are changed together, in the same software configuration commit).

The generated candidate decompositions are then fed into different pipeline stages. In the *Visualization* stage, the candidate decomposition is represented as a set of graphs. In these graphs, nodes can represent clusters, classes, domain entities, or methods, while edges represent the dependencies between them. These dependencies are based on the criteria used during the decomposition generation. Another provided visualization perspective illustrates the sequence of accesses for a particular functionality within the context of the candidate decomposition, enabling the identification of which accesses occur in each of the candidate microservices.

In the other stage, we have the *Quality Assessment and Comparison*. The decomposition is analyzed to assess its qualities, through metrics such as cohesion, coupling, complexity, number of interfaces exposed; and to compare it with other decompositions, usually done through observing the differences between them.

Finally, the *Editing and Modeling* stage enables the architect to make adjustments in the candidate decompositions while concurrently recalculating the associated quality metrics. This iterative process allows for meticulous refinement, ensuring that the resulting architecture aligns closely with the desired specifications.

2.3 Design

As stated in [1, 2], *"The main design strategy to support the variation points is to decouple the modules that implement them"* inducing that the primary approach chosen to address variation points is to separate or reduce the interdependence between the modules responsible for implementing these variations. The Variation Points are areas in a system where different implementations or variations may occur to meet specific needs. Decoupling modules allow for greater flexibility and adaptability in handling different options or variations.

If we intend to extend a certain module, we need to be sure of what needs to be changed. It can involve either extending just the corresponding module or, alternatively, extending the intermediate artifacts as well. As an example, we have the case addressed by [1, 2]. Take into account the first stage of the microservices identification pipeline, the *Collection*, where the monolith representation is created and then consumed in the *Decomposition Generation*. In this case, we currently have support for limited types of representation. Extending these representations can mean something like using two already existing and supported functionalities of the tool (combining both static and dynamic analysis), or it can be implementing a totally new representation using a different type of collection technique. This will be discussed in more detail in Chapter 4.

Figure 2.2 presents a comprehensive representation of the database domain model of the Mono2Micro application. The *Codebase* entity encapsulates the monolith and is associated with a collection of mono-

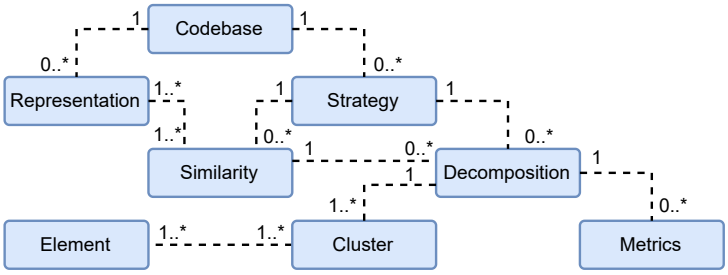

Figure 2.2: Domain Model of Mono2Micro

lith representations stored in the database. These representations, along with a set of strategies, serve as a foundation for creating similarities.

Strategies represent specific microservices identification approaches (e.g., lexical analysis), and utilize similarity measures to implement variations, often with specific weights. For instance, in a lexical approach, weights may differ based on the nature of monolith elements, such as their persistence or role as web services. Each *Strategy* is also characterized by the aggregation algorithm that is used.

The *Decomposition* entity decouples the initial stages of the Mono2Micro pipeline from subsequent visualization, modeling, and assessment stages. A *Decomposition* comprises clusters, each housing a set of elements. The entities—*Decomposition*, *Cluster*, and *Element*—serve as the building blocks for different types of decompositions. A *Decomposition* consists of multiple *Clusters*, each containing a set of *Elements*. These entities can be extended to support variations across different types of decompositions.

The *Representation* entity, Figure 2.2 and 2.3, undertakes the transformation of monolith representations in the context of a particular decomposition. It corresponds to a refinement of the monolith *Representation* in the context of a given decomposition. It also supports the variation points associated with the metrics calculations.

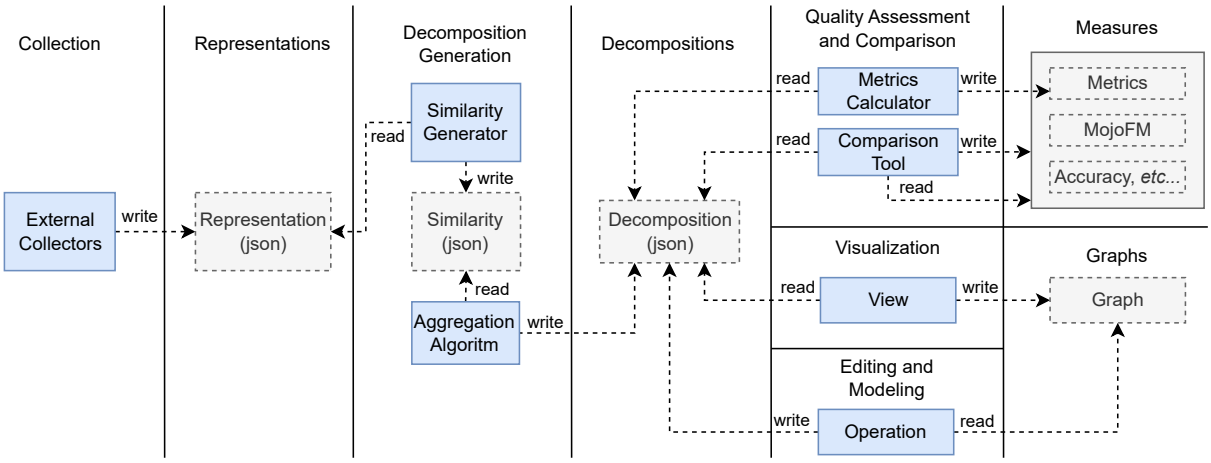

Figure 2.3: Design Structure of Mono2Micro for Identification of Microservices

Figure 2.3 provides a visual representation of the various stages within the microservices identification pipeline and illustrates how the modules interact with each other through the use of intermediate artifacts. These modules are specifically designed to implement variation points and offer hooks and interfaces to ensure pluggability and adaptability throughout the pipeline.

The initial stage, *Collection*, incorporates variation points that are facilitated by the `External Collectors` module. These collectors, implemented as external modules or independent tools, cater to a diverse range of data collection and execution tracing tools. They operate without a strict interface requirement, focusing instead on generating a monolith representation that can be processed in subsequent stages.

Moving on to the *Decomposition Generation* stage in Figure 2.3, this stage is implemented by two modules and an intermediate artifact. The modules address two critical variation points: the *Similarity Generator* module, responsible for similarity criteria generation, and the *Aggregation Algorithm* module, which offers various algorithms. These modules are effectively decoupled through a *Similarity* entity, allowing the introduction of new algorithms while maintaining compatibility with existing similarity criteria. The format of the *Similarity* entity must be agreed on between these modules. Extending these modules accommodates the use of various criteria and algorithms. The *Similarity Generator* module reads monolith representations to apply its similarity criteria and generates the similarity measures, while the *Aggregation Algorithm* module uses the similarity measures to output a *Decomposition* entity, representing the generated decomposition.

The *Decomposition* entity serves as a bridge, decoupling the *Decomposition Generation* stage from the downstream stages. It supports variation points within these stages, influencing subsequent components such as the *View*, *Metric Calculator*, *Comparison Tool*, and *Operation* modules. These latter modules play a crucial role in the final stages of the Mono2Micro pipeline, depending on the information contained in the decomposition, which includes the *Representation Information*, as illustrated in Figure 2.2.

3

Related Work

Contents

Identification of microservices within monolith systems is a challenging process. Consequently, a comprehensive analysis of the existing literature provides information on the various decomposition strategies, clustering methodologies, and associated metrics employed by different researchers.

In alignment with the previously proposed pipeline, our focus is to evaluate how the decomposition approaches of various authors align with the parameters outlined in Chapter 2.

3.1 Collection

The *Collection* corresponds to the analysis of the monolith and the generation of a *Monolith Representation*.

Abgaz et al. [5] describe different data collection techniques that can be classified into distinct categories based on the methodologies employed to extract information from monoliths. The representation of monoliths can take various forms, such as Dataflow Diagram (DFD), Unified Modeling Language (UML) models, Entity-Relationship Diagrams (ERD), and use cases, all falling under the Model-based Information Collection (MIC) category. In contrast, the Code-based Information Collection (CIC) category encompasses representations like source code, Abstract Syntaxe Tree (AST), interfaces, and stored procedure representations. The Log-based Information Collection (LIC) category includes representations of execution traces, runtime frequencies, and logs. Lastly, the Version-based Information Collection (VIC) category is suited to capture the version history, the contributor history, and related data.

3.1.1 Model-based Information Collection

MIC involves the extraction of data from monoliths using various graphical representations, such as DFD, UML models, ERD, and use cases. These models serve as visual representations of the system's structure, behavior, and interactions, aiding in the understanding of the monolith's functionalities. MIC plays a crucial role in understanding the underlying design and logic of the monolith, facilitating the analysis and extraction of pertinent information for further processing and interpretation.

3.1.2 Code-Based Information Collection

CIC involves the extraction of data from monoliths through the analysis of source code, AST, interfaces, and stored procedure representations. This method focuses on delving into the programming logic and structure of the monolith, aiming to capture essential information embedded within the codebase. By analyzing the code, developers can gain insight into the implementation details, dependencies, and system functionalities, which are crucial for comprehending the monolith's behavior and performance.

3.1.3 Log-Based Information Collection

LIC involves the extraction of data from monoliths through the analysis of execution traces, runtime frequencies, and log representations. By examining these logs, developers can gain insight into the system's runtime behavior, performance metrics, and error occurrences. LIC serves as a crucial mechanism for monitoring and diagnosing the operational aspects of the monolith, providing valuable information to

identify issues, analyze system performance, and ensure the overall stability and reliability of the monolith system. Understanding the data captured through LIC is essential for effective troubleshooting, performance optimization, and the overall management of the monolith's operational environment.

3.1.4 Version-Based Information Collection

VIC involves the extraction of data from monoliths by analyzing the version history, contributor history, and related data. This method aims to capture the evolution of the monolith over time, including the changes made to its codebase, the individuals or teams responsible for these modifications, and the overall progression of the system's development.

3.2 Monolith Representations

Various techniques of information collection are employed to generate different types of monolith representations. These representations serve to capture both the static and dynamic aspects of the system. For instance, a **structure graph** represents the static structure of the monolith, highlighting relationships between classes and their attributes. Meanwhile, a **call graph** illustrates the dynamic behavior of the system by depicting the flow of method calls and interactions between different functionalities.

Other representations include **sequences of accesses**, which document interactions with persistent entities and classes, and **frequencies**, which highlight the frequency of use of different functionalities within the monolith. In addition, **lexical elements** are examined to identify commonalities between classes, methods, and interfaces. The relationship between files is considered by analyzing common commits and contributors, often represented in an **information graph** that combines static structures with dynamic behaviors like method calls and attribute accesses.

Business processes are mapped out in the **business processes** representation, while **dependency graphs** illustrate the dependencies between different components of the system. **Data flow diagrams** provide a view of the data's path through the system, from input through processing to output stages. There are also **clusters of Uniform Resource Identifier (URI) partitions**, which group URIs based on similar resource consumption patterns gleaned from historical access logs.

Lastly, the **abstract syntax tree** offers a hierarchical, tree-like representation of the essential elements and their relationships within the code, providing a high-level overview of the monolith's structure.

Note that the representations that use a graph can be weighted/unweighted and direct/indirect.

The collection techniques outlined, like the relations between classes, the flow of interactions, sequence of accesses or frequency of certain functionalities all contribute to understanding monolith systems from various perspectives. The interplay between these techniques, coupled with a thoughtful

Algorithm Category	Advantages	Disadvantages
Clustering Algorithms	Efficient and scalable Clear, interpretable clusters	Predefined distance measures may oversimplify relationships Some algorithms require predefined number of clusters
Community Detection Algorithms	Good at identifying natural groupings No need to specify number of clusters in advance	Computationally expensive for large systems May produce overly fine-grained decompositions
Genetic Algorithms	Can optimize multiple objectivesEffective for large and complex search spaces	Computationally intensive Requires careful parameter tuningSlow convergence, results may vary

Table 3.1: Advantages and Disadvantages of Different Decomposition Algorithm Categories

choice of monolith representations, can significantly enhance the efficacy of microservices identification within monolith systems.

3.3 Decomposition Generation

In the *Decomposition Generation* phase, both criteria and algorithms play a crucial role. Criteria can involve maximizing or minimizing certain aspects that define good microservices to calculate similarity measures, such as the distance between monolith classes, methods, and functionalities to derive the best candidate microservices.

Decomposition algorithms fall into several categories: **Clustering Algorithms**, **Community Detection Algorithms**, and **Genetic Algorithms**. Clustering algorithms include methods like hierarchical clustering [6], collaborative clustering [7], affinity propagation [8], CO-GCN [9], K-means [10], SarF Map [11], and minimum spanning tree with Kruskal's algorithm [12]. Community detection approaches feature the Louvain algorithm [13], LDA classifier combined with Louvain [13], network-based community detection [14], Girvan-Newman algorithm [15], and fast community detection methods [16]. Genetic algorithms are represented by techniques like the Non-Dominated Sorting Genetic Algorithm - III [17].

3.4 Visualization

Visualization plays a crucial role in providing an overview of the decomposed system, aiding developers in understanding the entities and their relationships within the microservices architecture. Visualization methods include cluster graphs, entities graphs, community graphs, call graphs, cluster dendrograms, and tables of classes. This phase enhances the overall comprehension of the candidate decomposition during microservices migration.

3.5 Quality Assessment and Comparison

The *Quality Assessment and Comparison* phase evaluates the effectiveness and quality of the decomposition using various metrics. These include coupling, cohesion, complexity, precision, recall, the number of interfaces exposed, network overhead, feature modularization, modularity, coverage, CPU utilization, memory utilization, and execution times. These metrics provide a quantitative analysis to assess whether the candidate decomposition results in an optimal system structure.

This comprehensive set of metrics used to evaluate decompositions provides a view of various aspects, including structure, performance, and functionality. These metrics collectively contribute to informed decision making during monolith migration. Knowing whether or not the microservice candidates are highly cohesive and loosely coupled, knowing how the new architecture affects the system in terms of performance, or even how many exposed interfaces the system ends up with, all this helps the decomposition process, enabling developers to fine-tune and optimize the architecture.

3.6 Investigation of Existing Approaches

In alignment with the previously proposed pipeline, our focus is on researching how the decomposition approaches of various authors align with the parameters outlined. In Table 3.2, Table 3.3, and Table 3.4 it is presented the research carried out by a variety of authors in this field of study. In Table 3.2 we can observe the collection mechanisms and the representation of monoliths adopted by the authors. In Table 3.3 the criteria and algorithms that will provide the decompositions into microservices adopted by the authors are presented. Table 3.4 follows the codebases chosen by the authors and the metrics used to evaluate their proposed decomposition.

However, it is important to note that there is currently no comprehensive study evaluating which alternatives are objectively better. This gap leaves the question of optimal decomposition approaches open for further exploration.

Table 3.2: Collection & Monolith Representations

Article	Date	Collection	Monolith Representation	Article	Date	Collection	Monolith Representation
[18]	2023	Static Code Analysis	Information Graph	[19]	2020	Dynamic Analysis	Package Trace Representation
[8]	2021	Code2Vec	Abstract Syntaxe Tree	[20]	2020	Mapping between OOP concepts and the microservices concepts	Not Discussed
[17]	2021	Static Analysis Dynamic Analysis	Direct Graph	[21]	2019	Static Analysis Dynamic Analysis	Data Flow Diagrams
[7]	2021	Gathering of Business Processes	Set of logically related activities	[6]	2019	Static Analysis	Text Call Graph
[9]	2021	Static Code Analysis	Direct Call Graph	[22]	2019	Dynamic Analysis	Execution Traces
[13]	2021	Static Analysis	Abstract Syntaxe Tree	[23]	2019	Static Analysis	Dependency graph
[10]	2021	Static Code Analysis Semantic Analysis Syntatic Analysis	Word Matrix	[24]	2019	Every class is a microservice	Every class is a microservice
[14]	2021	Concept Analysis	Directed Call Graph	[25]	2019	URI Space Partitioning	Clusters of URI Groups
[26]	2019	Dynamic Analysis	Execution Paths	[16]	2018	Static and Evolutionary Coupling	Relational Graph displaying coupling between classes
[27]	2020	Microservices Miner Orchestrator and Searcher	Not Discussed	[11]	2018	Static Analysis	List of Program Groups
[28]	2020	Dynamic Analysis	Matrix of Metrics	[29]	2018	Static Analysis Dynamic Analysis	Set of Call Graphs
[15]	2020	Static Analysis Dynamic Analysis	Global Dependency Graph	[30]	2018	Static Analysis Dynamic Analysis	Dependency Graph Abstract Syntaxe Tree(AST)
[31]	2020	Extraction of operations and parameters from the API	Vector representations of operation names	[12]	2017	Semantic Coupling Strategy Contributor Coupling Strategy	Weighted Graph: higher weight value indicates stronger coupling.
[32]	2020	Domain Analysis Static Analysis Dynamic Analysis	Runtime Behaviour Visualization	[33]	2017	Manual Construction of a DFD Condense the DFD into a decomposable DFD	Data Flow Diagram
[34]	2020	Dynamic analysis	Not Discussed	[35]	2017	Lexical Analysis Algorithm 1: Decomposition Algorithm	Not Discussed
[36]	2020	Static Analysis Structural Analysis	Not Discussed	[37]	2016	Machine-readable representation artifacts describing intermidiate stages of analysis and design	Weighted unidrected graph

Table 3.3: Criteria & Algorithms

Article	Date	Criteria	Algorithm
[18]	2023	Maximizes a modularity score	Louvain Algorithm on complete information graph Louvain Algorithm on subgraph
[8]	2021	Generate attention weights for: - Method vectorization (aggregation function of abstract syntaxe tree paths) - Class vectorization (aggregation functions of method vectors)	Affinity Propagation Algorithm: Clustering method on class vectors
[17]	2021	Usage of fitness function to maximize/minimize coupling, cohesion, network overhead, feature modularization	Non-dominated Sorting Genetic Algorithm NSGA-III
[7]	2021	Foster both cohesion and loose-coupling	Colaborative Clustering - Algorithm 4, 5
[9]	2021	Derive vector representations of the nodes Minimize the effect of outlier nodes Obtain communities in the graph	CO-GCN
[13]	2021	Use the extracted information to fit a topic modelling technique allowing to identify topics and their distributions	LDA classifier - Latent Dirichlet Allocation Louvain Algorithm
[10]	2021	Syntatic and Semantic relationships	K-means
[14]	2021	Based on the inferred representative business logic programs/classes for each business functionality, referred to as ”seeds”	Network-based community detection algorithm
[26]	2019	Metric Based Ranking	Matric Based Ranking Algorithm
[27]	2020	Not Discussed	AST to text
[28]	2020	Identification of controller and subordinate objects. Each pair, CO SO is evaluated by a relation matrix based on the frequency and Invocation Strengh of the calls between CO and SO	Not Discussed
[15]	2020	Analysis of the strenght of the dependencies between components	Girvan-Newman
[31]	2020	Clustering the vector representations	Affinity Propagation
[38]	2020	Identification Based on the Entire Set of Gravity Centers Identification Based on the Exact Number of Microservices	Algorithm 1 Algorithm 2
[32]	2020	Analysis of the behaviour to identify microservice candidates	Not Discussed

Table 3.3 continued from previous page

Article	Date	Criteria	Algorithm
[34]	2020	Clustering the URI taking into consideration two clustering parameters (i) mean request response time and (ii) frequency of invocation of each request	K-means clustering algorithm to identify the clusters
[36]	2020	Clustering based on the euclidian distance to the centroid	Algorithm 1 - Discovery of BO and class relationships K-means clustering
[19]	2020	Extensive set of rules defined and explained in their research	6-Rule Approach Application
[20]	2020	Fitness Function based on the values: Functionality(Coarse-Grained) Composability(Composable) Self-Containment(Loose-Coupled) Usage(Discoverable)	Clustering algorithm
[21]	2019	Frequency of communication among processes and the execution time of processes to determine the clustering	Clustering algorithm
[6]	2019	Weighted entities	Hierarchical Clustering Algorithm
[22]	2019	Maximizing structural intra-connectivity Maximizing structural inter-connectivity Maximizing conceptual intra-connectivity Maximizing conceptual interconnectivity	Functional Atom Grouping
[23]	2019	Document collection Preprocessing Latent Dirichlet Allocation	Latent Dirichlet Allocation Seeded Latent Dirichlet Allocation
[24]	2019	Maximize Cohesion Minimize Coupling	NSGA-II
[25]	2019	Identify a k discrete number of URI groups with similar resource requirements deploys the identified URI partitions as separate microservices without any human intervention. dynamically scaling the resources allocated for each microservice.	Scale Weighted K-means
[16]	2018	Maximizing the modularity function	Fast Community
[11]	2018	Dedication Score	SArF Map

Table 3.3 continued from previous page

Article	Date	Criteria	Algorithm
[29]	2018	Similarity Value of subgraphs - Algorithm 1 Create set Z - set composed of all the functions that operate on a single business object	Algorithm 1 and 2
[30]	2018	Degree of coupling expressed by cosine distance Sum of Squared Errors -¿ Evaluate the accuracy of clustering results	K-means hierarchical clustering Algorithm 1
[12]	2017	Analysis of weight between classes High weight will cluster them in the same microservice or make them connected components	Kruskal Algorithm - calculates minimum spanning tree Algorithm 1
[33]	2017	Individual modules of operation represent potential microservices	Algorithm 1
[35]	2017	Computes the best mappings between the specifications and the reference vocabulary	Algorithm 1, 2
[37]	2016	Coupling Criteria Catalog	Girvan-Newman Epidemic Label Propagation

Table 3.4: Codebases & Metrics

Article	Date	Codebases	Metrics
[18]	2023	JPetStore - https://github.com/mybatis/jpetstore-6 Spring Petclinic - https://github.com/spring-projects/spring-petclinic SpringBlog - https://github.com/Raysmond/SpringBlog Cargo Tracking - https://github.com/citerus/dddsample-core	AverageCoupling AverageCohesion IFN - Average number of interfaces exposed by a microservice
[8]	2021	JPetStore - https://github.com/mybatis/jpetstore-6 SpringBlog - https://github.com/Raysmond/SpringBlog JForum - https://sourceforge.net/projects/jforum2/ Roller - https://github.com/apache/roller Spring Petclinic - https://github.com/spring-projects/spring-petclinic Spring Petclinic Microservices - https://github.com/spring-petclinic/spring-petclinic-microservices	Cohesion at Message Level (CHM) Cohesion at Domain Level (CHD)
[17]	2021	Tecgraph Institute codebase	Coupling Cohesion Network Overhead Feature Modularization
[7]	2021	Not Discussed	Afferent Coupling Efferent Coupling Instability Relational Cohesion
[9]	2021	DayTrader - https://github.com/WASdev/sample.daytrader7 PBW - https://github.com/WASdev/sample.plantsbywebsphere Acme-Air - https://github.com/acmeair/acmeair DietApp - https://github.com/SebastianBienert/DietApp/	Modularity Structural Modularity Non-Extreme Distribution (NED) Interface Number (IFN)

Table 3.4 continued from previous page

Article	Date	Codebases	Metrics
[13]	2021	JPetStore - https://github.com/mybatis/jpetstore-6	Interface Number - IFN Cohesion at Message Level - CHM Cohesion at Domain Level - CHD Interaction Number - IRN Strucural Modularity Quality - SMQ Conceptual Modularity Quality - CMQ
[10]	2021	Dolibarr open-source enterprise management system - https://github.com/Dolibarr/dolibarr	Scalability Availability Execution Efficiency CPU Utilisation
[14]	2021	DayTrader - https://github.com/WASdev/sample.daytrader7 Acme-Air - https://github.com/blueperf/acmeair-monolithic-java TNTConcept - https://github.com/autentia/TNTConcept Spring Petclinic - https://github.com/spring-projects/spring-petclinic	Number of Clusters Coverage Modularity Cohesion Non-extreme distribution (NED) Average microservice size (AMS)
[26]	2019	Not Discussed	Scalability Availability Execution Efficiency CPU Utilisation
[28]	2020	JPetStore - https://github.com/mybatis/jpetstore-6 Prodruction SSM - https://github.com/megagao/production_ssm	Non-Functional Metrics: Invocation Strengh Invocation Frequency Functional Metrics: Functional Metrics related to object(FO) CPU related Objective (CO) Memory Related Objective (MO)
[31]	2020	Not Discussed	Lack of Cohesion Number of Operations (Complexity)

Table 3.4 continued from previous page

Article	Date	Codebases	Metrics
[38]	2020	FindSportMates - https://github.com/chihweil5/FindSportMates Springblog - https://github.com/Raysmond/SpringBlog InventoryManagmentSystem - https://github.com/gtiwari333/java-inventory-management-system-swing-hibernate	Number of excellent, good and bad microservices
[32]	2020	in—FOCUS - sales and management software solution for lottery providers	Not Discussed
[34]	2020	Teachers Feedback Web Application (TFWA)	Not Discussed
[36]	2020	SugarCRM - https://github.com/sugarcrm ChurchCRM - https://github.com/ChurchCRM/CRM	Lack of Cohesion (LOC) Structural Coupling (StrC) Scalability [CPU] Scalability [DB CPU] Scalability network Availability Efficiency
[19]	2020	eShopOnWeb - https://github.com/dotnet-architecture/eShopOnWeb	Percentage of calls made between packages Percentage of calls made to a package from other packages Percentage of calls made by a package to other packages
[20]	2020	Sample Project	Fitness Function based on the values: Functionality(Coarse-Grained) Composability(Composable) Self-Containment(Loose-Coupled) Usage(Discoverable)
[21]	2019	DDD Sample Core - https://github.com/citerus/dddsample-core	Afferent Coupling Efferent Coupling Instability Relational Cohesion

Table 3.4 continued from previous page

Article	Date	Codebases	Metrics
[6]	2019	LdoD - https://github.com/socialsoftware/edition Blended Workflow - https://github.com/socialsoftware/blended-workflow	Number of Singleton Clusters Maximum Cluster Size Silhouette Score Precision Recall F-score
[22]	2019	JPetStore - https://github.com/mybatis/jpetstore-6 SpringBlog - https://github.com/Raysmond/SpringBlog Solo - https://github.com/lzyzsd/b3log-solo JForum - https://sourceforge.net/projects/jforum2/ Apache Roller - https://roller.apache.org/ Agilefant - https://sourceforge.net/projects/agilefant/ Xwiki-platform - https://www.xwiki.org/xwiki/bin/view/Documentation/	Interface Number Cohesion at Message Level Cohesion at Domain Level Structural Modularity Quality Conceptual Modularity Quality Internal Co-Change Frequency External Co-Change Frequency Ratio of ICF to ECF
[23]	2019	Daytrader - https://github.com/davemulley/daytrader-ee6.git	Precision Recall F- measure
[24]	2019	JPetStore - https://github.com/mybatis/jpetstore-6 SpringBlog - https://github.com/Raysmond/SpringBlog	Cohesion at Message Level Cohesion at Domain Level Operation Number Interaction Number
[25]	2019	Acme-Air - https://github.com/blueperf/acmeair-monolithic-java	Residual Error Response Time Throughput Average CPU utilization Allocated VMS
[16]	2018	eQuality Academics	MoJoSim Similarity Metric
[11]	2018	Spring Petclinic - https://github.com/spring-projects/spring-petclinic	Low Coupling

Table 3.4 continued from previous page

Article	Date	Codebases	Metrics
[29]	2018	SugarCRM - https://github.com/sugarcrm ChurchCRM - https://github.com/ChurchCRM/CRM	Number of Requests Execution Time Average Memory Average Disk Scalability Availability Efficiency Cohesion Coupling
[30]	2018	DayTrader - https://github.com/WASdev/sample.daytrader7 JPetStore - https://github.com/mybatis/jpetstore-6 TCP-W RUBiS	Average Throughput Maximum Load Steady Load
[12]	2017	Not Discussed	Execution Times Team size reduction ratio metric Average domain redundancy metric
[33]	2017	Not Discussed	Fine grain and focus High cohesion and loose coupling Neutral development technology
[35]	2017	Cargo Tracking - https://github.com/citerus/dddsample-core	Compare with expert decompositions Compare with original microservices architecture
[37]	2016	Cargo Tracking - https://github.com/citerus/dddsample-core A fictitious Trading System	Cohesiveness Criteria Compatibility Criteria Constraints Criteria

As can be observed, there is high variability in the current approaches and a lack of comparison between them. In this work, we aim to address this gap by directly comparing some of the different collection techniques to evaluate their impact on decomposition quality. Therefore, returning to our research questions, Table 3.2 highlights the numerous collection methods utilized to extract information from monoliths. In particular, there is no singularly predominant method favored by the community, although static or dynamic analysis is utilized in 62.5% of the papers analyzed. We recognize that enhancing support for diverse collection techniques contributes to the tool's versatility, enabling it to accommodate a broader spectrum of studies in the future. Examining Table 3.3, we observe a variety of criteria employed by different authors to generate candidate decompositions. Although clustering and community detection algorithms are a common choice, the specific criteria applied vary among researchers. In the context of validation, presented in Table 3.4, we will utilize some codebases present in the table to test our own solution. Notably, the metrics column reveals a multitude of evaluation criteria for decompositions. However, there is a predominant emphasis on metrics related to coupling and cohesion, and their various derivatives, as at least one of them is present in 47% of the papers.

4

Solution

Contents

4.1 Solution Overview

The goal is to extend the framework by incorporating support for a new monolith representation derived from a structural collector. This extension provides structural data that will be processed throughout the entire microservice identification pipeline, allowing us to assess its impact from data collection to final decomposition.. We will then experiment with various codebases to analyze and compare the results.

The primary objective of this project is to enhance the existing framework by introducing support for a new type of collector. This extension introduces a new monolith representation into the pipeline,

allowing us to investigate its impact on the microservices identification process. By enriching the information gathered during the collection stage, we aim to improve the accuracy and effectiveness of the decomposition process. The study will provide a detailed analysis of how these new collectors influence various stages of the pipeline.

To measure the effectiveness of these enhancements, we will thoroughly test their impact on the pipeline. Specifically, we will incorporate support for a novel structural representation of monoliths. We expect that these enhancements will lead to decompositions characterized by loose coupling and high cohesion. This expectation comes from the fact that the structural representation captures important relationships between monolith entities, improving the identification of natural microservice boundaries.

This extension of collection techniques is intended to provide an enriched understanding of the results of the automatic identification of microservices in monoliths, by comparing with the results achieved when other collection techniques are used.

Therefore, we plan to conduct rigorous experiments using various codebases, comparing the results obtained with the extended framework against those achieved with the original setup, as well as results from other authors and against expert decompositions. This comparative analysis will reveal the potential advantages and disadvantages of each of the collectors and representations, offering insights into their practical applicability.

In summary, our solution focuses on extending the framework, conducting a detailed study of its impact on the microservices identification pipeline, and performing experiments with diverse codebases to validate the applicability of the identification pipeline.

4.2 Pipeline Extension

The stages in the pipeline that were extended include the *Decomposition Generation*, where support for a new type of monolith representation was added. This involved integrating a new structural representation in the pipeline, allowing for a detailed analysis and decomposition generation.

In the *Visualization*, enhancements were made to display information related to the new representation, such as the entities each cluster references and the inheritance hierarchies within the system. This information is associated with the representation information associated with the generated decomposition. Visual enhancement helps to better understand and analyze the structural relationships and dependencies within the proposed decomposition.

Additionally, in the *Quality Assessment and Comparison* a new metric is introduced to evaluate the decomposition groups, known as the *Purity* metric. This metric allows for a more nuanced evaluation of the generated decompositions, comparing them against expert-defined clusters, and providing insight into the structural quality of the decompositions.

4.3 Strategy for Implementation

The development process began with an analysis of the existing codebase, focusing on understanding the implementation of the *Sequence of Accesses* representation. This initial exploration was crucial for identifying the core structures and processes that would guide the development of new features and fixes.

Development Steps

1. **Initial Implementation:** The first commit included a basic implementation that generated a manual decomposition. This initial step focused on establishing a working foundation, even though with many components hard-coded. The primary goal was to create a proof-of-concept that could be incrementally refined.

2. **Frontend Enhancements:** Subsequent efforts improved the frontend, enabling interaction with all graph elements. This included fixing bugs related to distance calculations and edge generation in the visualization of clusters and entities.

3. **Structural Recommendations:** The next significant milestone was making the recommendation feature 4.5.2 functional for structure representations. This involved integrating and debugging the logic that underpins the recommendation system.

4. **Inheritance Relationships:** The codebase was extended to include inheritance relationships between entities, allowing for a more nuanced representation of data structures. This included addressing issues in the visualization and interaction between clusters and entities, particularly in the context of inheritance.

5. **Metric Addition:** A notable feature addition was the Purity metric, designed to analyze proposed clusters against expert-defined clusters. This feature was crucial for evaluating the effectiveness of the clustering algorithm. However, detailed information regarding the Purity metric are deferred to Section 5.2.

6. **Data Expansion:** Further refinements included expanding the data available for Purity analysis, particularly in mapping clusters and identifying common entities. This enhancement aimed to provide deeper insights into the clustering process and its outcomes.

Throughout the development process, the focus remained on iterative improvement and refinement. The approach was characterized by initial broad implementations followed by targeted debugging and feature enhancement, ensuring a robust and functional result.

4.4 Similarity Measures and Decomposition

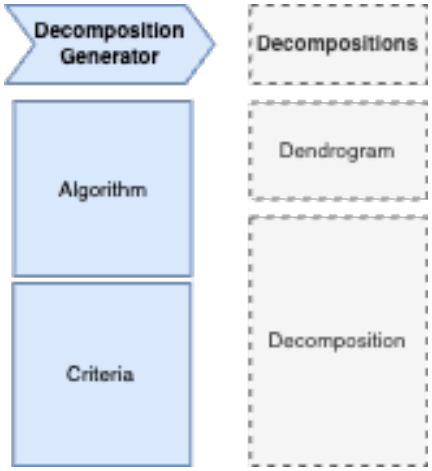

Figure 4.1: Decomposition Generation

Considering the challenges of extending modules, let us revisit the example mentioned earlier. In the initial stages of the microservice identification pipeline, specifically during the *Collection* phase, the creation and consumption of the monolith representation lay the foundation for subsequent processes, notably the *Decomposition Generation*.

The outcome of this structural collector is a JSON file containing comprehensive information about all entities within the monolith. For each entity, the file includes details of its interactions and covers the names and types of entities involved. These interactions can take the form of one-to-one connections or one-to-many relationships. For example, an entity may be linked to another entity in a one-to-one manner or to an entire list of entities in a one-to-many configuration. In addition, we consider inheritance relationships between classes, assigning a specific weight to represent the value of these relationships in the monolith structure.

After this analysis, various similarity measures are generated which are then utilized by the clustering algorithm responsible for generating dendrograms 4.2. These similarity measures indicate the closeness of two entities, reflecting the number of references between them and the presence of inheritance relationships.

The set of all references of a given entity *e* is given by

$$\text{ref}(e) = \{\text{att in } e : \text{att.target is Entity}\}$$

so a set of references of *e* is composed of all attributes of *e* that refer an Entity.

Similarly, the set of references of an entity *e1* that refer an entity *e2* is given by

$$\text{ref}(e1, e2) = \{\text{att } \in \text{ref}(e1) : \text{att.target is } e2\}$$

where a reference is considered from *e1* to *e2* if the attribute of *e1* is a reference contained in *e1* and refers to *e2*.

Relationship Weights:

To calculate the similarity measure, we also need a notion of weight, which we define as follows:

$$\textit{weight}(att) = \begin{cases} o2o & \text{if att.target.multiplicity} = 1 \\ o2m & \text{if att.target.multiplicity} > 1 \end{cases}$$

Here, *o2o* and *o2m* are parameters that represent the weights assigned to one-to-one and one-to-many relationships, respectively. These weights are configured by the developer based on the specific needs and context of the application.

Inheritance Weight:

In addition to the relationship weights, we introduce an inheritance weight *inher*, which accounts for the significance of inheritance relationships between classes. This weight is applied when a class e_1 is either a subclass or superclass of another class e_2. The inheritance weight *inher* is configurable by the developer and can be set based on how much importance is given to the inheritance in the system's architecture.

The inheritance weight between two classes e_1 and e_2, denoted as *inher*(e_1, e_2), is defined as follows:

$$\textit{inher}(e_1, e_2) = \begin{cases} \textit{inher}, & \text{if } e_1 \text{ is a superclass of } e_2 \text{ or } e_2 \text{ is a superclass of } e_1, \\ 0, & \text{otherwise.} \end{cases}$$

Here, *inher*(e_1, e_2) represents the inheritance weight between classes e_1 and e_2, and *inher* is the predefined weight value set by the developer.

Developer Configuration

The weights $o2o$, $o2m$, and $inher$ are configurable by the developer, such that their sum is 100. For instance, if one-to-one relationships are considered more significant than one-to-many relationships and inheritance, the developer might set:

$$o2o = 70, \quad o2m = 20, \quad inher = 10$$

Alternatively, if inheritance is deemed more critical, the weights might be configured differently:

$$o2o = 30, \quad o2m = 30, \quad inher = 40$$

This flexibility allows the developer to fine-tune the similarity measure calculation according to the specific requirements and priorities of the application.

Entity Similarity Distance

The distance between two entities is represented by a value between 0 and 1 and is calculated as follows:

$$d(e_1, e_2) = \frac{\sum_{\text{att} \in \text{ref}(e_1, e_2)} \textit{weight(att)} + \textit{inher}(e_1, e_2)}{\sum_{\text{att} \in \text{ref}(e_1)} \textit{weight(att)} + \textit{inher}(e_1)}$$

$$\text{provided that} \sum_{\text{att} \in \text{ref}(e_1)} \textit{weight(att)} + \textit{inher}(e_1) \neq 0.$$

where e_1 and e_2 are two entities, and the distance is given by the sum of the weights of the references from e_1 to e_2 (including the inheritance weight *inher*(e_1, e_2)) divided by the sum of the weights of all references in e_1 (plus the inheritance weight *inher*(e_1, e_2)).

This measure accounts for both direct relationships and inheritance hierarchies, providing a comprehensive similarity metric for the entities in the system.

Dendrogram

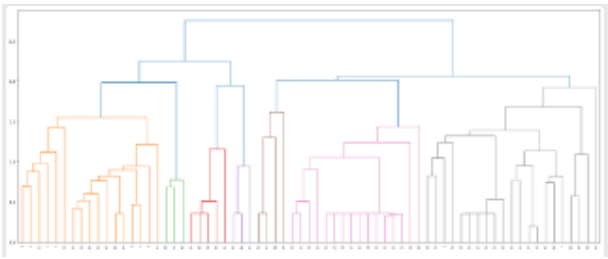

Figure 4.2: Dendrogram

After calculations, the tool generates a dendrogram, like the one depicted in Figure 4.2. It serves as a visual representation of the dependencies between all classes, which aids the class allocation process to microservices. To illustrate, in Figure 4.2 on the horizontal axis we have the classes that the monolith is composed of, while as we go up vertically we can see the relations between the classes. If our objective is to create a decomposition with four microservices, the dendrogram is segmented at a specific height, which groups the classes into four clusters. This height-based segmentation facilitates the identification of microservices.

4.5 Implementation Design

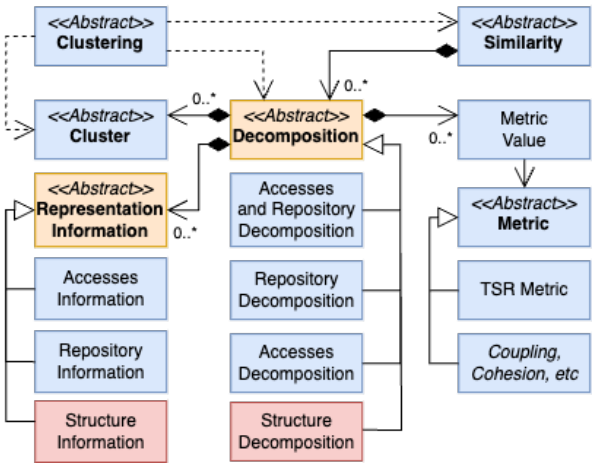

Figure 4.3: Model Decomposition

4.5.1 Structural Representation

The first part of our implementation involved creating a new type of representation, called the *Structural Representation*. This new representation, along with the extensions made to the system, depicted in the red rectangles of Figure 4.3, also includes support for a new type of file generated by the structural collector, depicted in Figure 4.4. This structure file contains most of the information needed to execute the decompositions. To implement these changes, several new classes were developed, while existent classes were extended to handle the new representation.

- **Extended Classes:**
  - Backend: `StructureInformation`, `StructureInformationDto`, `StructureRepresentation`, `StructureRepresentationDto`
  - Frontend: `SimilarityMatrixScipyStructureForm`, `StructureRepresentation`
- **Customized Classes:**
  - Backend: `Representation`, `RepresentationFactory`, `RepresentationInformationFactory`, `RepresentationInformationDtoFactory`, `RepresentationDtoFactory`
  - Frontend: `Decompositions`, `Similarities`, `Representation`, `RepresentationFactory`

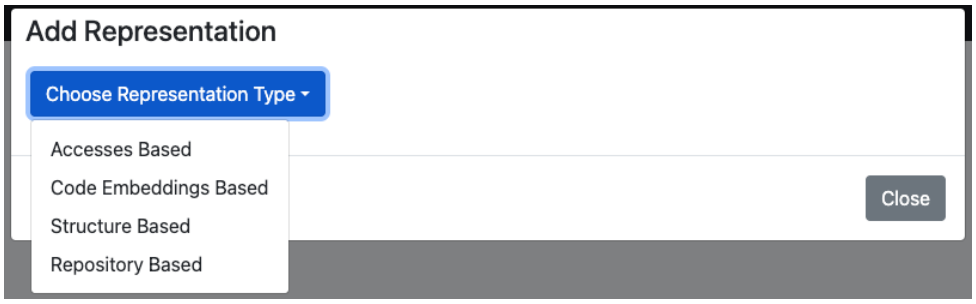

Figure 4.4: Choices of representations, including the new one

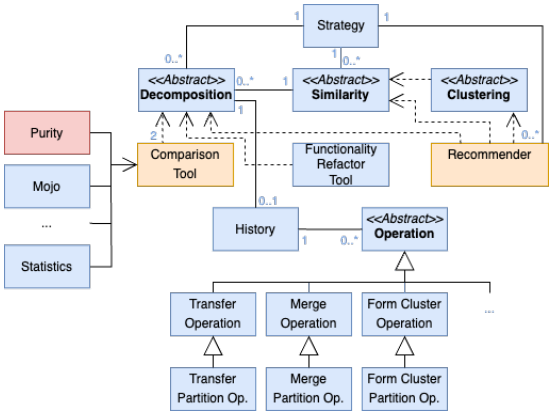

Figure 4.5: Mono2Micro Tools

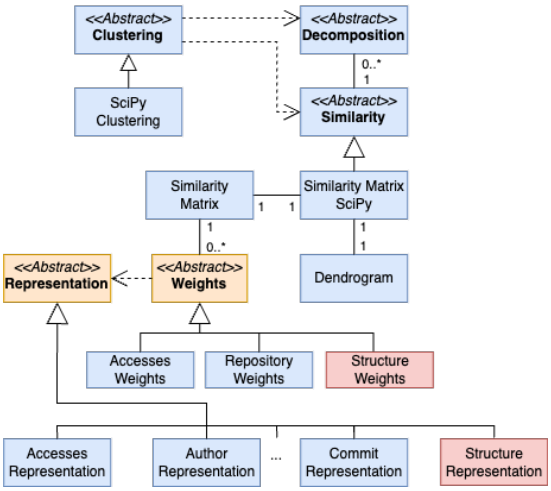

Figure 4.6: Extensions

4.5.2 Decomposition Generation

The next step in the implementation focuses on the decomposition generation, where we compute all similarity measures to create candidate decompositions. During this process, users can define the weights necessary to compute the similarity measures, as described in Section 4.4.

Decomposition generation can be carried out in two ways:

- **Manual:** Users manually set up the weights required for computing the similarity measures. Fig. 4.6
- **Automatic:** The Mono2Micro tool generates multiple recommendations by automatically assigning values to the weights. These recommendations are presented alongside the evaluation metrics, allowing the user to select the most suitable ones. Fig. 4.5

To support these two approaches, the framework includes extensions and customizations of key classes in both the frontend and backend. The following outline the relevant extended and customized classes:

- **Extended Classes:**
  - Backend: `SimilarityScipyStructure`, `StructureWeights`, `SimilarityScipyStructureDto`, `SimilarityStructureIterator`
  - Frontend: `SimilarityScipyStructure`, `StructureWeights`
- **Customized Classes:**
  - Backend: `Weights`, `WeightsFactory`, `SimilarityDto`, `SimilarityDtoFactory`, `Strategy`, `StrategyService`, `RecommendationFactory`, `RecommendationsType`, `RecommendMatrixSciPy`, `RecommendationDto`, `RecommendationDtoFactory`
  - Frontend: `SimilarityFactory`, `StrategyTypes`, `WeightsFactory`, `Recommendations`, `RecommendationFactory`, `RecommendationTypes`

4.5.3 Graphical Representation of Decompositions

After generating the decompositions, the system presents two visualizations of the decomposition results. One view represents clusters as nodes (Figure 4.7), while the other view represents entities as nodes (Figure 4.8).

In the **Cluster as Nodes** view, clicking a node reveals all entities within that cluster (Figure 4.9), while clicking an edge shows references between entities across clusters (Figure 4.10). This view also identifies relationships such as inheritance or extension between entities in different clusters.

In the **Entity as Nodes** view, clicking on an entity node shows its references and inheritance hierarchy (Figure 4.11), while clicking an edge between two entities reveals the type of reference, such as an instance, list, or set (Figure 4.12).

- **Extended Classes:**
  - Frontend: `StructureView`, `StructureViewModal`
- **Costumized Classes:**
  - Backend: `StructureInformation`, `StructureInfoDto`, `StructureWeights`
  - Frontend: `ClusterView`, `PartitionsDecomposition`

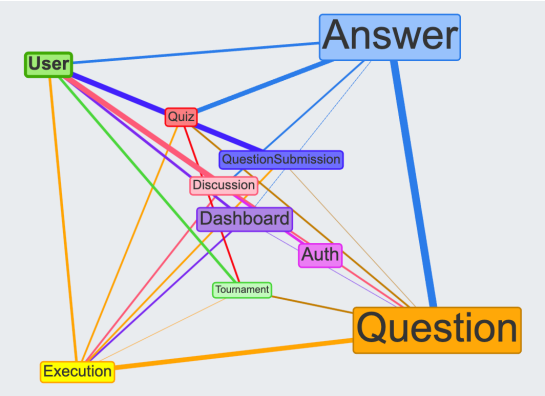

Figure 4.7: Cluster Representation as Nodes

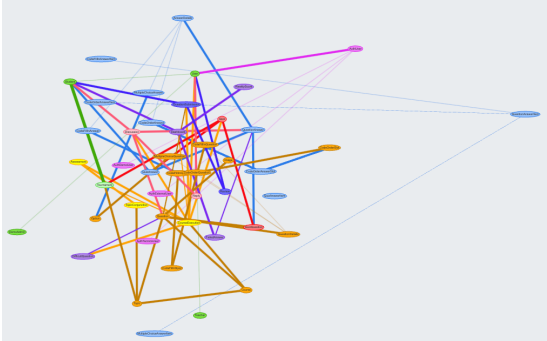

Figure 4.8: Entity Representation as Nodes

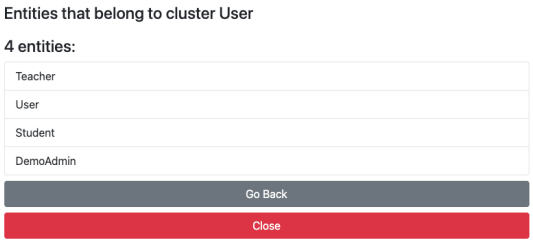

Figure 4.9: Entities Contained in the Selected Cluster

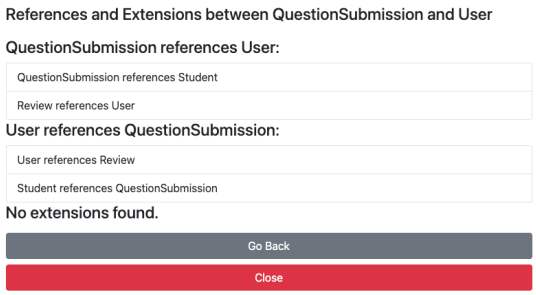

Figure 4.10: References Between Entities Across Clusters

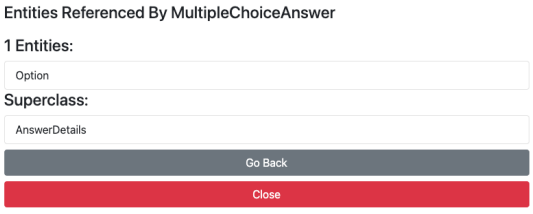

Figure 4.11: References and Inheritance Hierarchy of the Selected Entity

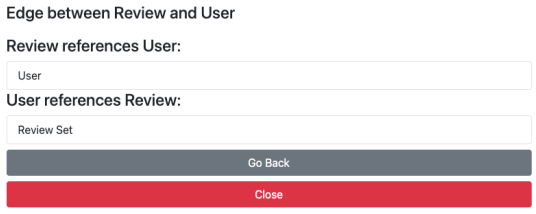

Figure 4.12: Type of Reference Between Selected Entities

4.5.4 Clustering Evaluation: Purity Metric

To evaluate the success of our decompositions, we implemented the Purity metric, with the results shown in Figure 4.13 in our comparison tool. Along with the purity value, the tool displays detailed information for each cluster, including expert cluster mapping and shared entities between the proposed and expert clusters. The backend class `Purity` handles this calculation, while the frontend class `PurityResults` presents the results. We can see this integration in the context of the application in Fig. 4.5.

- **Extended Classes:**

  - Backend: `Purity`
  - Frontend: `PurityResults`

- **Costumized Classes:**
  - Backend: `DefaultComparisonToolResponse`
  - Frontend: `ComparisonTool`

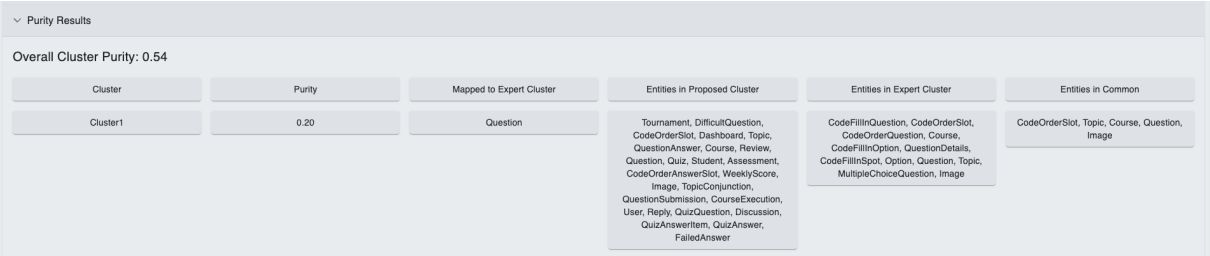

Figure 4.13: Purity Metric Results Display

5

Evaluation

Contents

The primary objective of the evaluation is to determine the effectiveness of our newly implemented collector in generating a decomposed system that adheres to the key principles of microservices architectures, such as ensuring each candidate microservice adheres to the single responsibility principle and operates independently, without reliance on other services. To achieve this, we make use of cohesion, coupling, and complexity metrics, quantifying the extent to which design principles are observed, and evaluating the structure of the decomposition. Our assessment involves a comparative analysis over several codebases, benchmarking our results against other state-of-the-art approaches, our own

implementation using a different collection technique, and an expert analysis.

5.1 Selected Codebases

Throughout our evaluation, we will utilize a diverse range of codebases to conduct a comprehensive analysis of the capabilities and performance of our microservices identification framework. We will particularly focus on a selection of codebases:

1. **Spring Petclinic** [1] - Spring Petclinic is an open-source sample application that demonstrates the usage of the Spring Framework in the context of a veterinary clinic management system. This codebase comprises **9 entities** and **13 functionalities**

2. **Cargo Tracking** [2] - The Cargo Tracking project is a sample application that serves as an implementation of Domain-Driven Design (DDD) principles. It is often used as a reference in the software development community to showcase how DDD concepts can be applied in practice. This codebase comprises **11 entities** and **10 functionalities**

3. **Quizzes Tutor** [3] - Quizzes Tutor is an educational platform designed to manage and deliver quizzes to students. It is a more complex codebase when compared to the others, demonstrating a larger number of functionalities and entities. This codebase comprises **46 entities** and **125 functionalities**.

While exploring the contributions of other authors, such as Filippone, Brito and Al-Debagy [8, 13, 18], and others, we observed the frequent utilization of these specific codebases. In alignment with this trend, we chose to employ these codebases in our study, allowing comparisons not only against our own results but also against findings from other researchers. These codebases also present diverse characteristics that encompass various application domains.

5.2 Evaluation Metrics

In order to conduct a statistical analysis we will use metrics that evaluate different aspects of the decomposition, such metrics are *Complexity, Cohesion, Coupling* [39] and *Purity* [40].

[1] https://github.com/spring-projects/spring-petclinic
[2] https://github.com/citerus/dddsample-core
[3] https://github.com/socialsoftware/quizzes-tutor

Complexity

Complexity measures the difficulty of implementing monolith functionalities as distributed transactions. It evaluates how challenging it is to break down a functionality into local transactions. Specifically, it looks at the number of interactions between a local transaction and other distributed functionalities based on domain entities. Higher complexity indicates more intricate coordination and integration across different parts of the system.

Cohesion

Cohesion assesses whether the functionalities within a cluster adhere to the principle of single responsibility, meaning that all domain entities accessed by a cluster are related and accessed together. It measures how well the functionalities in a cluster are aligned with each other in terms of the domain entities they interact with. High cohesion indicates that a cluster is well-organized, with domain entities involved together in the same functionalities, leading to a more cohesive and coherent cluster.

Coupling

Coupling evaluates the extent to which different clusters interact with each other. It measures how many domain entities a cluster exposes to other clusters through its interfaces. High coupling means that a cluster exposes most of its entities when other cluster interact with it. Reducing coupling is often desirable to minimize dependencies and improve modularity.

Purity

Purity is a metric used to evaluate the quality of clusters by measuring the extent to which each cluster contains elements from a single, pre defined, expert cluster. It is particularly useful in scenarios where the goal is to categorize data into distinct, meaningful groups. The *Purity* of a clustering solution is defined as follows:

$$\text{Purity}(D_E, D_P) = \frac{1}{N} \sum_i \max_j |C_i \cap L_j|$$

where:

- D_E represents the expert decomposition,
- D_P represents the proposed decomposition,
- C_i represents the set of elements in cluster i from the proposed decomposition D_P,

- L_j represents the set of elements in cluster j from the expert decomposition D_E,
- N is the total number of elements across all clusters,
- $|C_i \cap L_j|$ is the cardinality (number of elements) of the intersection between cluster C_i and cluster L_j.

Purity essentially measures how well the proposed clustering D_P aligns with the expert clustering D_E by considering the maximum overlap between each cluster in the proposed clustering and any cluster in the expert clustering.

This metric is valuable for assessing clustering quality because it provides insight into how well the clustering process has segregated the data according to the true underlying categories. However, it should be noted that high *Purity* does not necessarily indicate a perfect clustering solution, as it can be trivially maximized by creating a large number of clusters with a single element each. Therefore, *Purity* is often used in conjunction with other metrics.

Definition of True Positives, False Positives, True Negatives, and False Negatives

In the context of comparing a proposed decomposition with an expert decomposition, we define the concepts of true positives, false positives, true negatives, and false negatives as follows:

True Positives (TP): A true positive occurs when two entities are clustered together in both the proposed decomposition and the expert decomposition. This indicates that the proposed decomposition has correctly identified the relationship between these two entities.

False Positives (FP): A false positive occurs when two entities are clustered together in the proposed decomposition, but they are not clustered together in the expert decomposition. This means the proposed decomposition incorrectly grouped two entities that should have been separated according to the expert.

False Negatives (FN): A false negative occurs when two entities are clustered together in the expert decomposition, but they are not clustered together in the proposed decomposition. This indicates that the proposed decomposition has failed to identify a correct relationship between the two entities.

True Negatives (TN): A true negative occurs when two entities are not clustered together in both the proposed decomposition and the expert decomposition. This shows that the proposed decomposition has correctly recognized that these two entities do not belong in the same cluster.

By using these definitions, we can calculate various metrics such as Accuracy, Precision, Recall, Specificity, and F-score to evaluate how well the proposed decomposition aligns with the expert decomposition. These metrics provide insights into the performance of the decomposition in terms of correctly identifying relationships between entities and avoiding incorrect groupings.

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy evaluates how well the proposed decompositions perform overall. By measuring the proportion of correctly identified elements or classifications out of the total number of elements, we can determine how effectively each approach generalizes across different cases. Higher accuracy indicates that the decomposition or algorithm is performing well across the board, but it may not fully capture performance nuances.

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision reflects the proportion of true positives among the instances classified as positive, which is particularly important when false positives have significant implications. For example, in the context of functional decomposition, high precision ensures that each identified entity is correctly categorized, minimizing the risk of incorrectly grouping entities and ensuring precise results.

Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall is essential for evaluating how effectively our decompositions or algorithms capture all relevant instances. It measures the proportion of actual positives that are correctly identified, which is crucial for ensuring that no important functionalities or elements are missed. In our thesis, high recall indicates that the decomposition or algorithm is thorough in identifying all necessary components, which is vital for comprehensive and effective analysis.

Specificity

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Specificity complements recall by assessing how well the decompositions or algorithms avoid false positives. It measures the proportion of true negatives among the actual negatives, helping to ensure that non-relevant instances are correctly excluded. In the context of functional decomposition, high specificity means that the decomposition or algorithm effectively distinguishes between relevant and non-relevant functionalities, which helps in refining and validating the analysis.

F-score

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The *F-score* provides a balanced view of our decompositions performance by combining *Precision* and *Recall*. This metric is particularly useful in our thesis for evaluating the overall effectiveness of the decompositions, as it reflects both the accuracy of positive predictions and the completeness in identifying all relevant instances. A higher F-score indicates a well-balanced approach that achieves a good trade-off between precision and recall, ensuring that both the accuracy and completeness of the decomposition or algorithm are considered in our analysis.

5.3 Spring Petclinic

Spring Petclinic is a sample application for the spring framework. It shows how to build a Spring-based enterprise application to manage information about pet, pet owners, visits and veterinarians to a veterinarian clinic.

5.3.1 Entities

Following the pipeline collection stage, our collector identified several entities in the Spring Petclinic codebase: `BaseEntity`, `NamedEntity`, `Pet`, `PetType`, `Vet`, `Specialty`, `Person`, `Owner`, and `Visit`.

Mono2Micro generates multiple decompositions for each selected collection technique. The number of decompositions depends on the number of adjustable weights configured by the developer - the more weights involved, the more decompositions are suggested. Despite the large number of possible decompositions for each technique, our focus is on those that maximize cohesion while minimizing complexity and coupling. Given the simplicity of the application, one structural decomposition stands out, offering the lowest complexity and coupling, along with the highest cohesion amongst all recommendations.

5.3.2 Expert Decomposition

The expert-driven decomposition of the Spring Petclinic [4] application organizes the system into three distinct clusters. This decomposition reflects the domain knowledge of experienced developers, ensuring a meaningful grouping of entities based on functionality and structure. The expert decomposition is illustrated in Figure 5.1.

[4] https://github.com/spring-projects/spring-petclinic

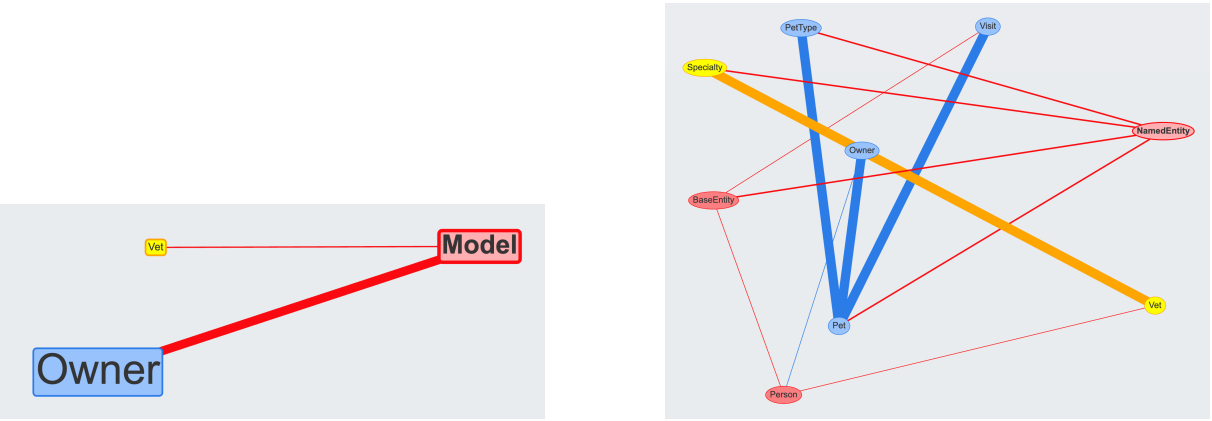

Figure 5.1: Expert decomposition of the Spring PetClinic application.

Since the expert decomposition consists of three clusters, we will limit our analysis to the proposed decompositions that also contain three clusters. This alignment ensures a consistent basis for comparison between the expert's understanding of the system and the automatically generated decompositions from Mono2Micro.

5.3.3 Evaluation of Automatic Decompositions

Decomposition	Accuracy	Precision	Recall	Specificity	F-Score	Purity
Structure	0.53	0.23	0	0.62	0.26	0.56
Accesses Highest Cohesion	0.53	0.23	0.3	0.62	0.26	0.56
Accesses Lowest Coupling & Complexity	0.56	0.31	0.5	0.58	0.38	0.67

Table 5.1: Comparison of Automatic Decompositions against Expert Decompositions for Spring Petclinic

Comparing Structural decomposition against the Expert

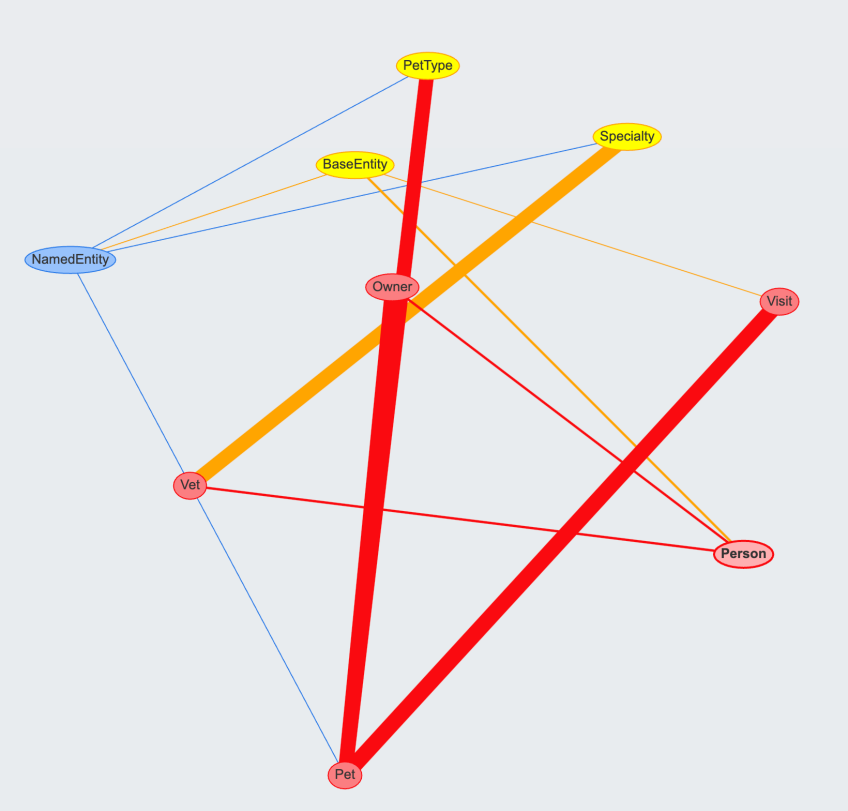

Figure 5.2: PetClinic Structural Decomposition

The evaluation of the Spring Petclinic codebase, using the decomposition generated by the tool which optimizes for lowest coupling, lowest complexity, and highest cohesion, reveals several important insights when compared to the expert-driven decomposition. The metrics obtained from the comparison can be observed in Table 5.1

The structural decomposition shows a moderate accuracy of 0.53, but it suffers from low precision (0.23) and recall (0.00). This suggests that the generated clusters contain significant false positives, with no true positives identified, reflecting a misalignment with the expert decomposition.

The evaluation of the Spring Petclinic decomposition reveals that the generated decomposition, while optimized for lowest coupling, lowest complexity, and highest cohesion, does not align closely with the expert decomposition. The expert decomposition, on the other hand, does not specifically aim to optimize these metrics. Instead, it focuses on grouping entities based on a domain-driven perspective, prioritizing maintainability and logical separation of concerns.

Comparing Access-Based Decompositions against the Expert

The evaluation of the Spring Petclinic codebase, using the decomposition generated by the tool that optimizes for the highest cohesion and the lowest complexity and coupling, reveals several key insights compared to expert-driven decomposition. The metrics obtained from this comparison can be observed in Table 5.1.

Highest Cohesion Decomposition

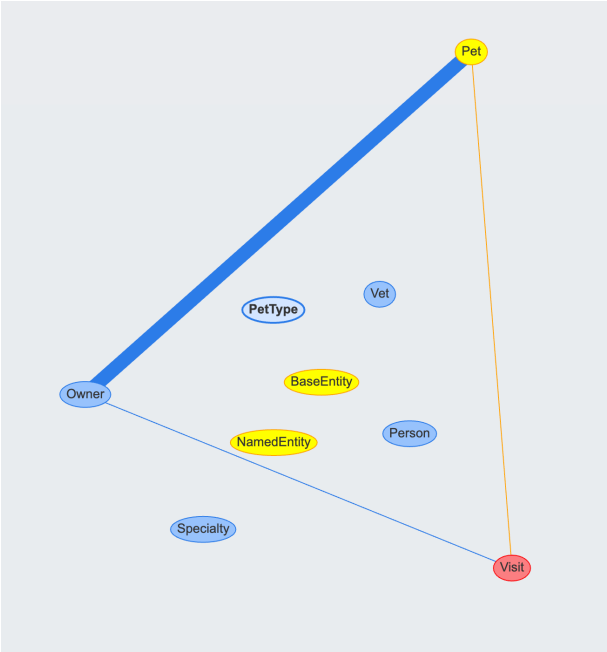

Figure 5.3: PetClinic Accesses Decomposition with Highest Cohesion

By analyzing Figure 5.3, we observe that certain entities, such as `PetType`, `Vet`, `Person`, `Specialty`, `NamedEntity`, and `BaseEntity`, appear isolated. This suggests that there are no transactional relationships linking these entities to others. This indicates potential issues in the dataset, where the example does not fully capture the relationships that typically guide decomposition. Consequently, the metrics can be calculated on incomplete or incorrect values, reducing the reliability of the results.

Although this decomposition optimizes for cohesion, the metrics are similar to the structural decomposition, with no significant improvements in accuracy (0.53) or precision (0.23). However, it shows a moderate recall improvement (0.30), indicating that some correct relationships between entities were captured.

The evaluation of the highest cohesion decomposition reveals that, despite being optimized for internal coherence within clusters, it shares many of the same shortcomings as the structural decomposition.

Accuracy and purity are moderate, but precision remains low and recall, although improved, is still insufficient. This suggests that while optimizing for cohesion does capture some of the expert's clustering logic, it does not do so comprehensively.

Lowest Complexity and Coupling Decomposition

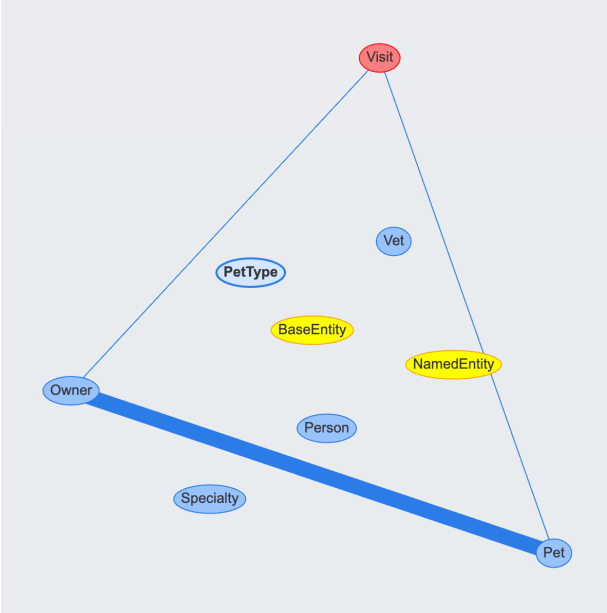

Figure 5.4: PetClinic Accesses Decomposition with Lowest Complexity and Coupling

This decomposition produces the best overall results, with the highest accuracy (0.56), precision (0.31), and recall (0.50). The improvement in F-score (0.38) and purity (0.67) suggests that minimizing complexity and coupling leads to better alignment with the expert-driven decomposition.

The evaluation of the lowest complexity and coupling decomposition shows that this approach results in the best overall alignment with the expert decomposition, as indicated by its higher accuracy, precision, recall, F-Score, and purity. These results suggest that focusing on minimizing complexity and coupling produces a decomposition that more accurately reflects the expert's understanding of how entities should be grouped.

Overall Comparison and Conclusion

The comparison between access-based and structural decompositions against the expert decomposition reveals that the access-based approach—especially when optimized for lowest complexity and coupling—yields better results across most metrics. However, both approaches fail to fully replicate the logic used by domain experts in clustering entities. The expert decomposition's adherence to domain-

driven design principles likely explains why it performs poorly on metrics designed to assess technical characteristics, such as coupling and complexity. This highlights the importance of considering both technical and domain-specific perspectives in decomposition generation.

Interestingly, the structural decomposition aligns better with the expert's logic in some areas, particularly where `Owner`, `Pet`, and `Visit` are clustered together in both the structural and expert decompositions. The structural decomposition's stronger alignment could be explained by the fact that many entities in the access-based decomposition have no associated transactions. The incomplete implementation of transactions within the codebase affects the clustering results, with only `Pet`, `Visit`, and `Owner` having actual interactions among them. This lack of transactional data may limit the effectiveness of the access-based decomposition.

5.4 Cargo Tracking

The Cargo Tracking Codebase is a part of the DDDSample project, which is a sample application that demonstrates the principles of DDD in action. It provides a reference implementation of a cargo shipping system, including various bounded contexts, entities, value objects, repositories, and services that focus on the core domain logic related to tracking cargo shipments.

5.4.1 Entities

The entities that compose this codebase are `Cargo`, `Location`, `Handling Event`, `Leg`, `Carrier Movement`, `Voyage`, `Delivery`, `Handling Activity`, and `Route Specification`.

Expert Decomposition

The expert-driven decomposition of the Cargo Tracking [5] application organizes the system into four distinct clusters. The expert decomposition is illustrated in Figure 5.5.

[5] https://github.com/citerus/dddsample-core

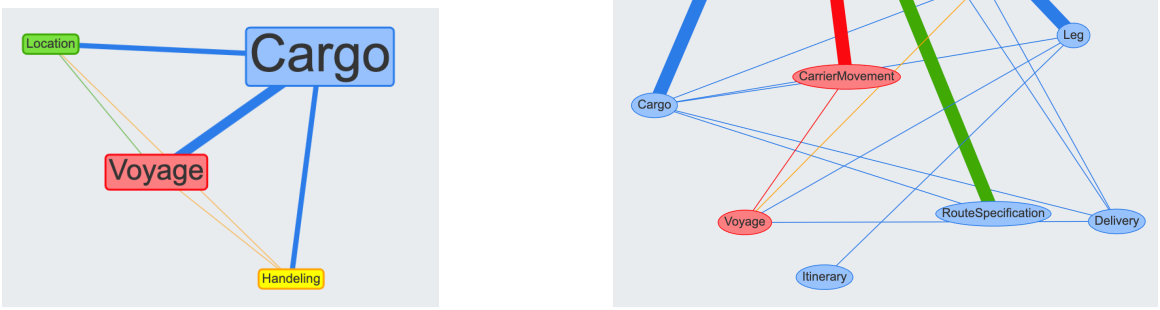

Figure 5.5: Expert decomposition of the Cargo Tracking application.

Since the expert decomposition consists of four clusters, we will limit our analysis to the proposed decompositions that also contain four clusters. This alignment ensures a consistent basis for comparison between the expert's understanding of the system and the automatically generated decompositions from Mono2Micro.

5.4.2 Evaluation of Automatic Decompositions

Decomposition	Accuracy	Precision	Recall	Specificity	F-Score	Purity
Highest Cohesion (Structure)	0.42	0.25	0.39	0.43	0.30	0.64
Lowest Complexity (Structure)	0.49	0.22	0.22	0.62	0.22	0.64
Lowest Coupling (Structure)	0.56	0.29	0.22	0.73	0.25	0.64
Access-Based	0.42	0.25	0.39	0.43	0.30	0.64

Table 5.2: Comparison of Automatic Decompositions against the Expert Decomposition for Cargo Tracking

Evaluation of the Cargo Tracking Decomposition With Structure Decomposition

This section evaluates the DDD Sample Core Cargo Tracking application by comparing three different decompositions generated by Mono2Micro (Highest Cohesion, Lowest Complexity, and Lowest Coupling) against the expert-driven decomposition. The evaluation is based on several key metrics: Accuracy, Precision, Recall, Specificity, F-Score, and Purity.

Highest Cohesion Decomposition

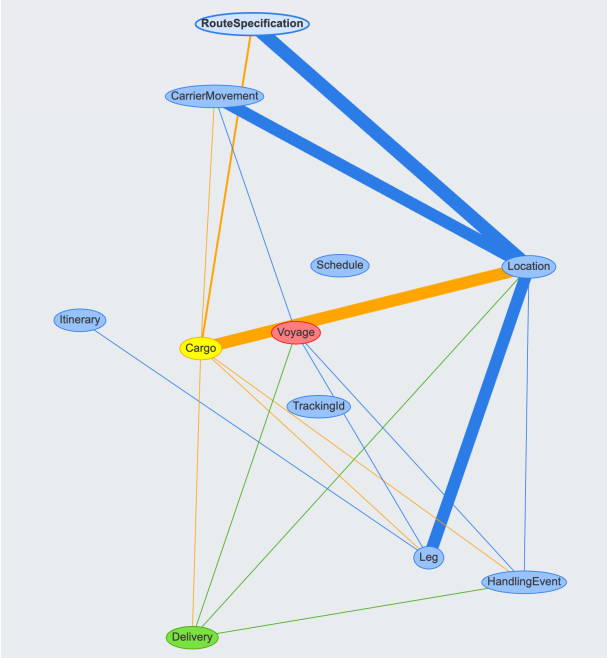

Figure 5.6: Cargo Tracking Struture Decomposition with Highest Cohesion

The Highest Cohesion decomposition prioritizes the grouping of entities to maximize internal coherence. The metrics for this decomposition are shown in Table 5.2. The Highest Cohesion decomposition provides moderate purity and recall, but it falls short in terms of accuracy, precision, and specificity. While it effectively groups entities into cohesive clusters, it does not align closely with the expert's understanding, leading to a significant number of incorrect groupings.

Lowest Complexity Decomposition

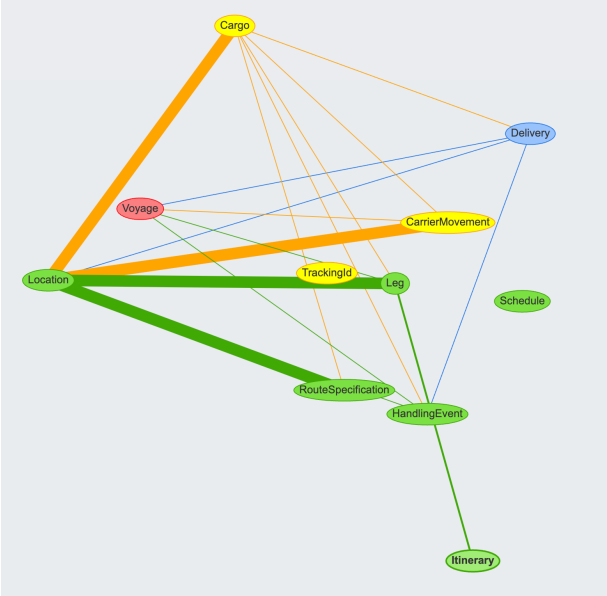

Figure 5.7: Cargo Tracking Struture Decomposition with Lowest Complexity

The Lowest Complexity decomposition focuses on simplifying the overall structure by minimizing the number of clusters and their interactions. The metrics for this decomposition are in Table 5.2..

The Lowest Complexity decomposition achieves moderate accuracy and specificity, indicating a balance between simplification and correctness. However, the low precision, recall, and F-Score highlight the trade-offs involved in minimizing complexity, leading to a decomposition that, while simpler, does not closely align with the expert's logic.

Lowest Coupling Decomposition

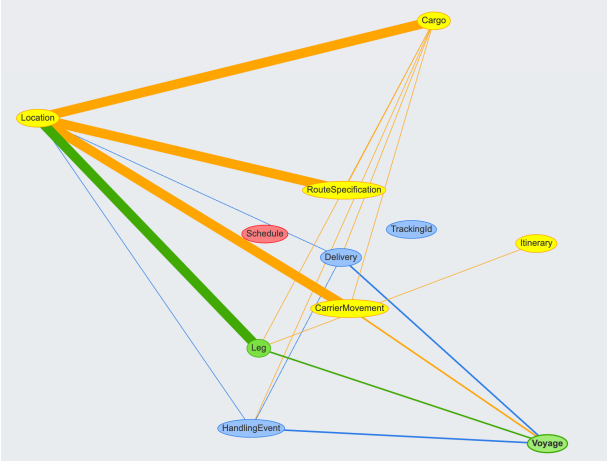

Figure 5.8: Cargo Tracking Struture Decomposition with Lowest Coupling

The Lowest Coupling decomposition prioritizes reducing the dependencies between clusters. The evaluation metrics for this decomposition can be observed in Table 5.2.

The Lowest Coupling decomposition achieves the highest accuracy and specificity among the three structural approaches, suggesting it is more effective at aligning with the expert's clustering while reducing dependencies. However, its precision, recall, and F-Score still indicate significant room for improvement in fully capturing the expert's understanding.

Access-Based Decomposition

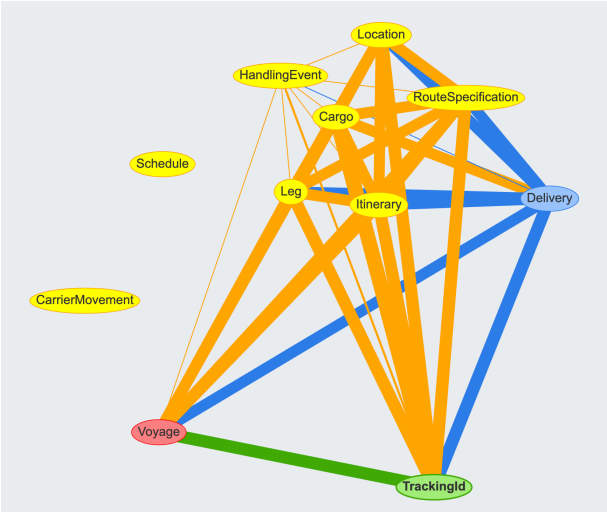

Figure 5.9: Cargo Tracking Accesses Decomposition

The access-based decomposition considers the sequences of accesses to optimize the decomposition process. The metrics for this decomposition are in Table 5.2.

The access-based decomposition performs similarly to the Highest Cohesion decomposition, with identical metrics across all categories. This suggests that focusing on access patterns produces a decomposition that is both cohesive and somewhat aligned with the expert's understanding, but still has notable shortcomings in precision, recall, and specificity.

Overall Summary

The evaluation of the structural decomposition shows a lack of consistent groupings, likely due to the approach of optimizing each metric individually. This fragmentation might also be influenced by limitations in the codebase, where certain entities like `Schedule` and `TrackingID` have no structural connections with other entities. These gaps may indicate incomplete implementation within the codebase, preventing these entities from being fully incorporated into meaningful clusters.

In contrast, the access-based decomposition presents different results. Since this decomposition optimizes all metrics collectively, it demonstrates a closer alignment with the expert's logic, albeit not perfectly. A key observation is that four entities—`Cargo`, `RouteSpecification`, `Leg`, and `Itinerary`—are clustered together in both the expert and access-based decompositions. Additionally, `Schedule` and `CarrierMovement` are grouped together in both cases. While the expert decomposition places `Voyage` in a small cluster, the access-based decomposition isolates it in its own cluster, which still shows some resemblance to the expert's grouping.

These results suggest that the access-based approach, by considering a holistic optimization of metrics, is better at capturing the relationships between entities as envisioned by the expert. However, neither decomposition fully aligns with the expert's clustering, highlighting the inherent challenges in creating an automatic decomposition that satisfies multiple criteria simultaneously.

5.5 Quizzes Tutor

Quizzes Tutor is an online tool that allows teachers to create and reuse multiple-choice questions with images and topics which can be inserted in assessments and quizzes. It is currently integrated with IST authentication so that it can be used for any course. Students can then answer those questions in generated or sugested quizzes (pseudo-random) providing them with a useful self-assessment tool to improve their learning.

5.5.1 Entities and Functionalities Overview

The Quizzes Tutor application is composed of a substantial number of entities and functionalities, which together form the backbone of its comprehensive system. In total, the application comprises 46 distinct entities. These entities represent the various objects, concepts, and resources that the application needs to manage and interact with, ranging from users and quizzes to specific topics and questions.

In addition to the entities, the application implements a broad range of functionalities, totaling 108 distinct functionalities. These functionalities are distributed across 18 different controllers, each responsible for managing a particular aspect of the application's operation. Controllers act as intermediaries between the application's data layer and user interface, handling requests, processing data, and ensuring that the system's operations run smoothly.

This wide array of entities and functionalities enables the Quizzes Tutor application to offer a rich set of features and capabilities, supporting its goal of providing a versatile and effective tool for both teachers and students in the creation, management, and completion of quizzes.

Expert Decomposition

To evaluate the modularization of the Quizzes Tutor [6] application, an expert-driven decomposition was also considered. This decomposition was manually crafted by a domain expert who has a deep understanding of the functionality and structure of the application. The expert decomposition aims to reflect an ideal partitioning based on practical experience and knowledge of the system's architecture, rather than relying solely on automated tools.

The expert-driven decomposition has two possible views: one focusing on the accesses and the other on the structure of the Quizzes Tutor application. These decompositions are intended to serve as a benchmark for evaluating the effectiveness of the automated strategies generated earlier.

The expert-driven decomposition of the Quizzes Tutor application organizes the system into ten distinct clusters. Both views of the expert decomposition are illustrated in Figures 5.10 and 5.11.

[6] https://github.com/socialsoftware/quizzes-tutor

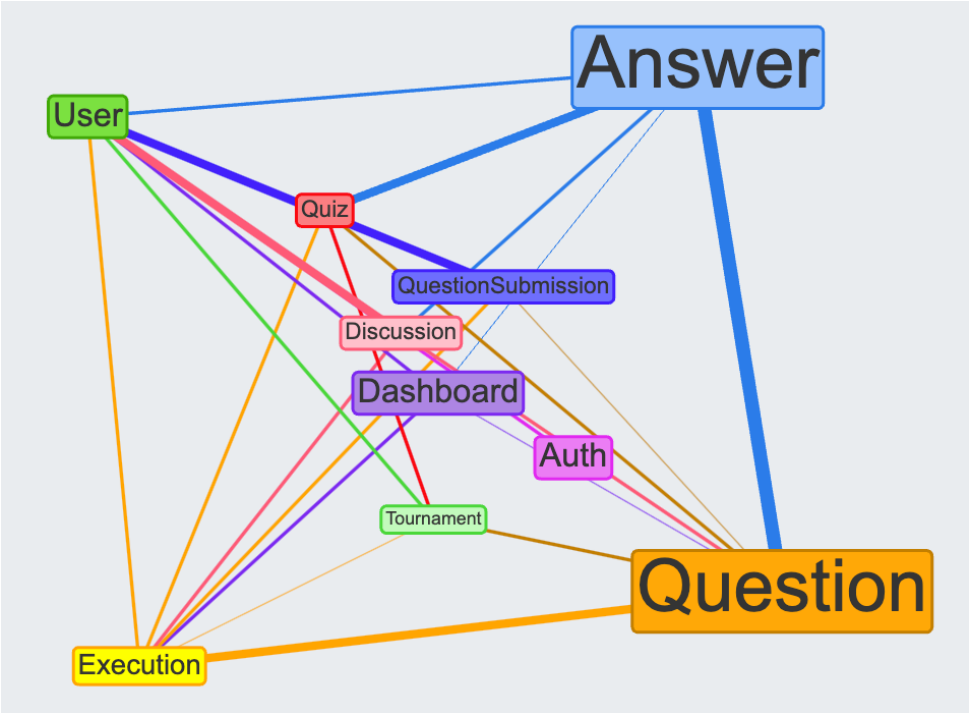

Figure 5.10: Expert decomposition of the Quizzes Tutor application. Clusters View

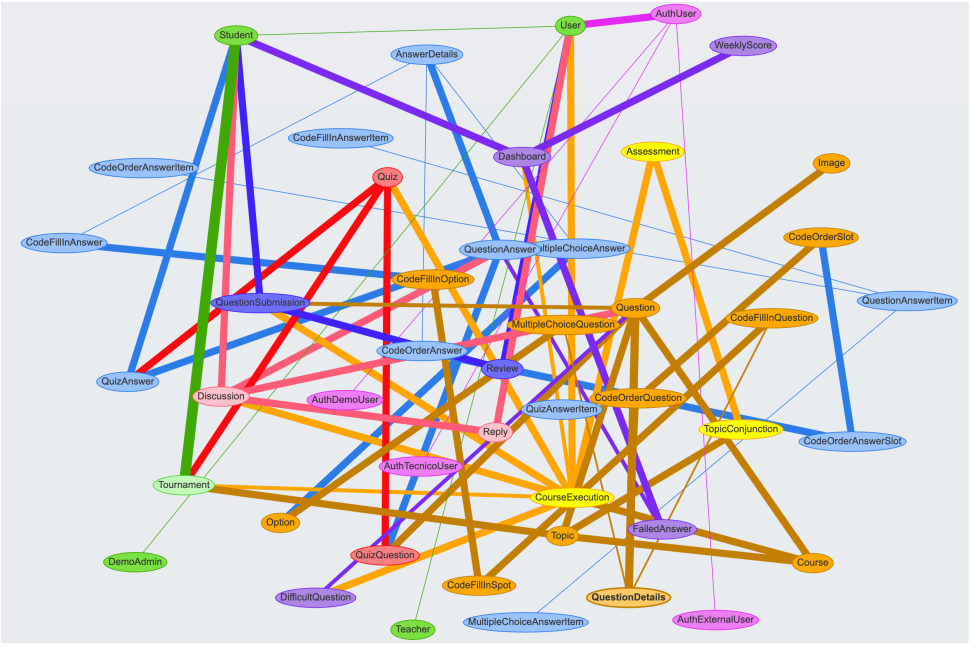

Figure 5.11: Expert decomposition of the Quizzes Tutor application. Entities View

Since the expert decomposition consists of ten clusters, we will limit our analysis to the proposed decompositions that also contain ten clusters. This alignment ensures a consistent basis for comparison between the expert's understanding of the system and the automatically generated decompositions from

Mono2Micro.

5.5.2 Evaluation of Automatic Decompositions

Decomposition Strategy	Accuracy	Precision	Recall	Specificity	F-Score	Purity
Highest Cohesion & Lowest Coupling (Accesses)	0.73	0.27	0.47	0.77	0.34	0.63
Lowest Complexity (Accesses)	0.73	0.25	0.40	0.79	0.31	0.61
Highest Cohesion (Structure)	0.62	0.15	0.34	0.67	0.21	0.57
Lowest Coupling (Structure)	0.63	0.15	0.31	0.69	0.20	0.57
Lowest Complexity (Structure)	0.63	0.14	0.28	0.69	0.19	0.54

Table 5.3: Comparison of Automatic Decompositions against the Expert Decomposition for Quizzes Tutor

5.5.3 Accesses Decomposition Results

To evaluate the modularization of the Quizzes Tutor application, two decomposition strategies were generated using the Mono2Micro tool. These decompositions aimed to optimize different architectural qualities such as cohesion, coupling, and complexity, resulting in two distinct partitioning strategies.

The first decomposition strategy was focused on achieving the highest cohesion and lowest coupling. The second decomposition strategy, on the other hand, was designed to minimize the overall complexity of the system.

5.5.4 Structure Decomposition Results

In addition to the accesses decomposition, a structure decomposition of the Quizzes Tutor application was performed. This approach also aimed to optimize each of the key architectural metrics. As a result, three distinct decomposition strategies were generated, each focusing on a specific metric.

5.5.5 Comparison of Accesses Decomposition with the Expert Decomposition

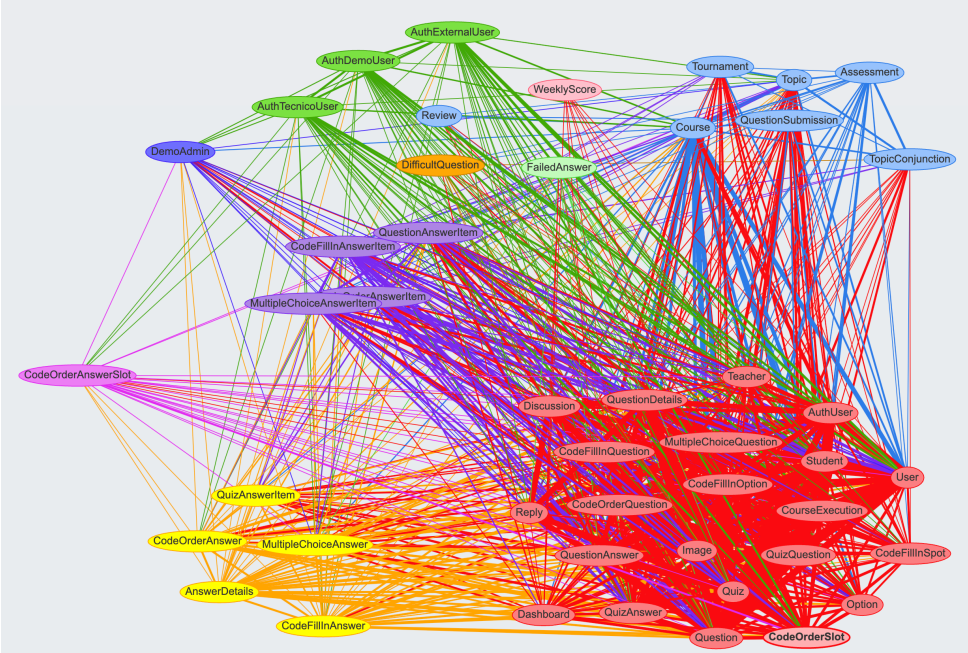

Figure 5.12: Quizzes Tutor Accesses Decomposition with Highest Cohesion and Lowest Coupling

Comparing Highest Cohesion and Lowest Coupling against Expert decompositions

The Mojo metrics (Accuracy, Precision, Recall, Specificity, F-Score, and Purity) provide a quantitative evaluation of how well the decomposition strategy with the highest cohesion and the lowest coupling aligns with the expert-driven decomposition. These metrics presented in Table 5.3 are crucial to understanding the effectiveness and quality of the automated partitioning strategy compared to expert intuition.

These metrics provide a more nuanced understanding of the clustering success. Although the automated decomposition has a moderate level of purity, the relatively low precision and F-score indicate that there are still notable discrepancies between the automated and expert-driven decompositions. This suggests that while the automated strategy is somewhat effective in grouping related entities, it may still benefit from refinement to better align with expert knowledge.

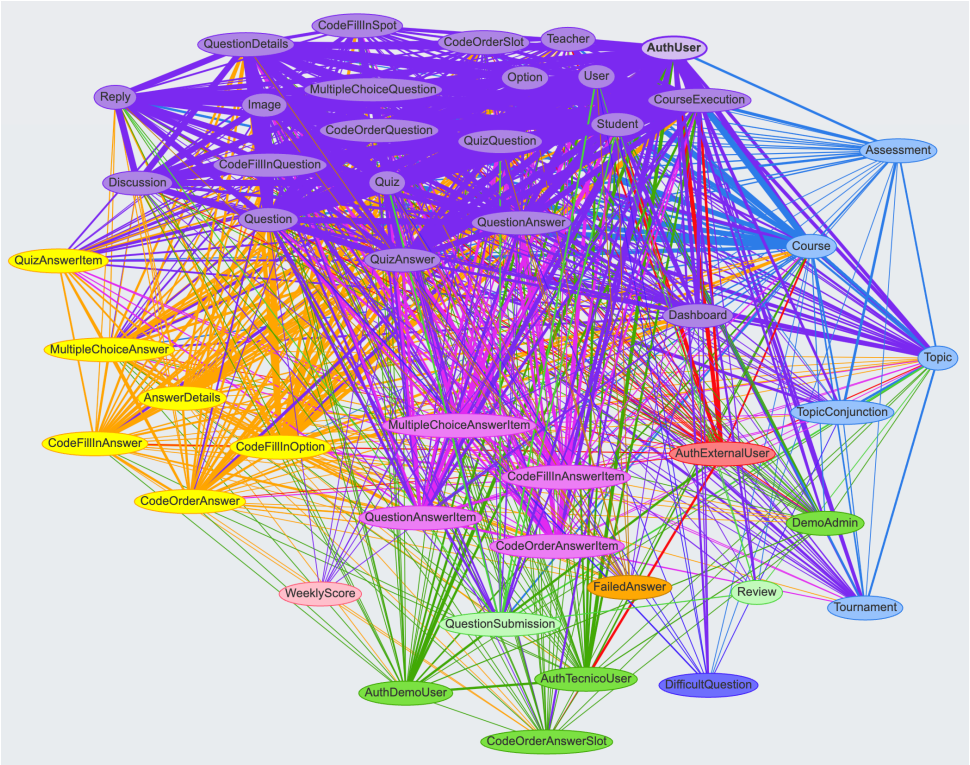

Figure 5.13: Quizzes Tutor Accesses Decomposition with Lowest Complexity

Comparing the Lowest Complexity against the Expert decompositions

We compare the Lowest Complexity decomposition of the Quizzes Tutor application with the Expert decomposition. The respective metrics are available in Table 5.3.

The comparison reveals that while the Lowest Complexity decomposition strategy maintains a reasonable accuracy and high specificity, it has low precision, recall, and F-score. This indicates that while the strategy effectively minimizes complexity, it may not always align closely with the expert's understanding of how entities should be clustered. The moderate cluster purity further supports this, showing that while some clusters are well-formed, others could benefit from refinement.

In conclusion, the Lowest Complexity strategy is effective for simplifying the system's architecture but might not always align with the expert's detailed clustering, potentially missing some important groupings that the expert suggested.

5.5.6 Comparison of Structure Decomposition with the Expert Decomposition

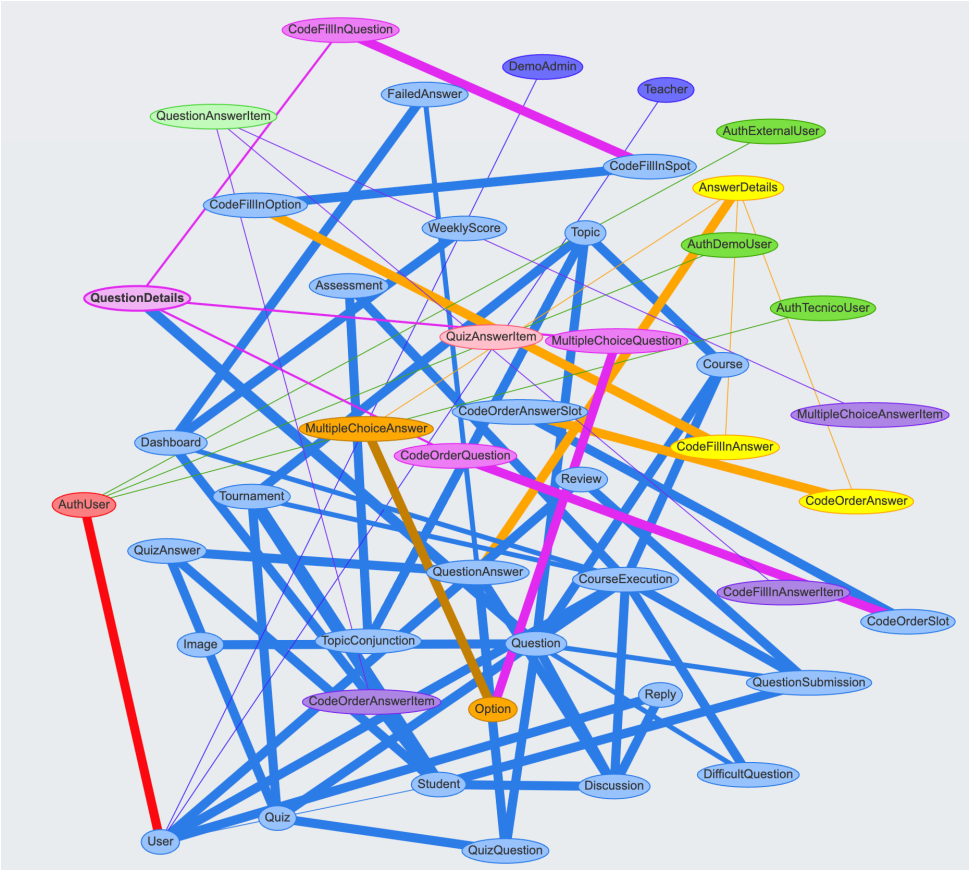

Figure 5.14: Quizzes Tutor Structure Decomposition with Highest Cohesion

Comparing Highest Cohesion against Expert decomposition

In this section, we compare the Highest Cohesion decomposition on the structure side of the Quizzes Tutor application against the Expert decomposition. The decomposition metrics can be consulted in Table 5.3.

The comparison reveals that the Highest Cohesion decomposition on the structure side achieves moderate accuracy and specificity, but suffers from low precision and recall, leading to a low F-score. This suggests that while the decomposition effectively creates cohesive clusters, it does not align well with the expert's understanding of entity relationships. The moderate cluster purity indicates some degree of success in forming homogeneous clusters, but there is significant room for improvement.

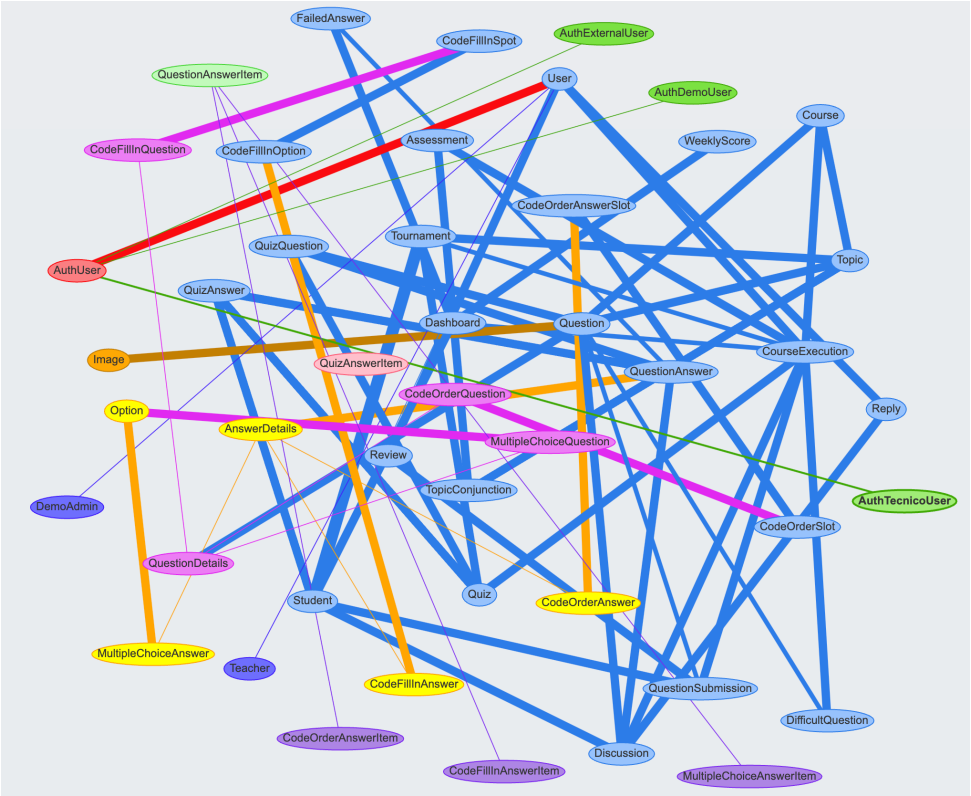

Figure 5.15: Quizzes Tutor Structure Decomposition with Lowest Coupling

Comparing Lowest Coupling against Expert decompositon

Considering the Lowest Coupling decomposition based on the structure information of the Quizzes Tutor application while comparing it with the Expert decomposition. The metrics resulting from this decomposition are in Table 5.3.

The comparison highlights that while the Lowest Coupling decomposition achieves moderate accuracy and specificity, it suffers from low precision and recall, leading to a low F-score and moderate cluster purity. This indicates that the strategy effectively reduces coupling but at the expense of clustering accuracy and homogeneity when compared to the expert decomposition.

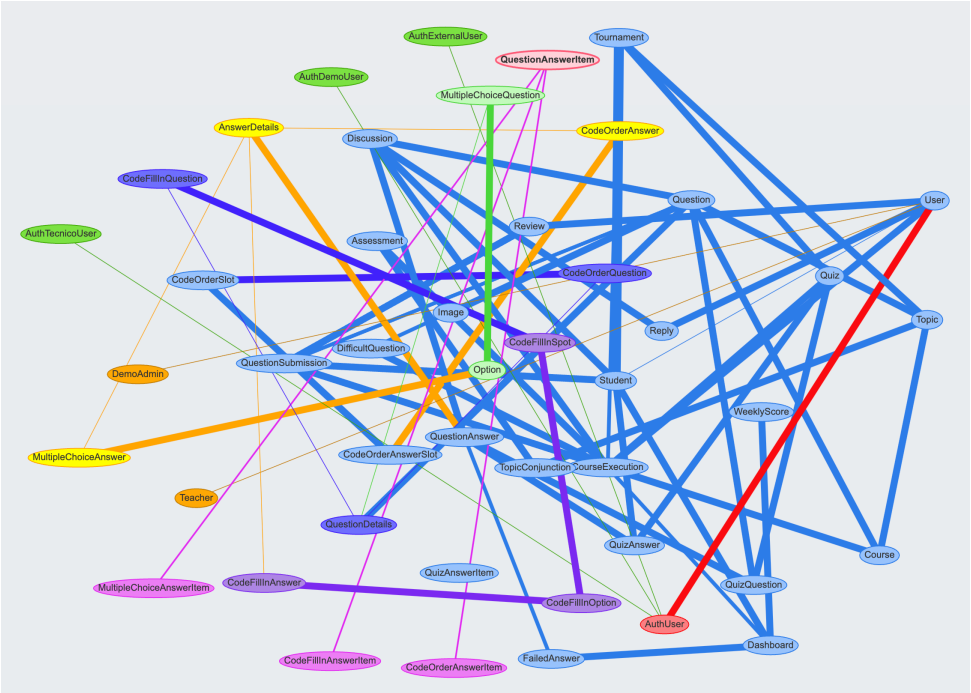

Figure 5.16: Quizzes Tutor Structure Decomposition with Lowest Complexity

Comparing Lowest Complexity against Expert decomposition

Lastly, we compare the Lowest Complexity decomposition on the structure side of the Quizzes Tutor application against the Expert decomposition. Metrics available in Table 5.3.

The comparison highlights that while the Lowest Complexity decomposition achieves moderate accuracy and specificity, it suffers from low precision and recall, leading to a low F-score and moderate cluster purity. This indicates that the strategy reduces complexity but at the expense of clustering accuracy when compared to the expert decomposition.

Overall Summary

The evaluation of the Quizzes Tutor decompositions reveals varying levels of alignment with the expert-driven approach, depending on the strategy employed. The access-based decomposition, particularly when optimizing for the highest cohesion and lowest coupling, shows moderate alignment with the expert decomposition. However, the relatively low precision and F-score suggest that the automated strategy still struggles to fully replicate the expert's clustering logic. This is especially true in cases where more intricate relationships between entities need to be captured, such as interactions across different controllers.

On the other hand, the lowest complexity decompositions, while effective at simplifying the system's

architecture, tend to sacrifice accuracy in clustering, leading to lower precision and recall. This shows that while the decomposition reduces complexity, it doesn't fully capture the expert's understanding of how entities should be grouped, as seen in the lower F-scores and moderate cluster purity.

The structure-based decompositions, similar to the access-based ones, also show moderate alignment with the expert's decomposition, but the lack of high precision and recall indicates room for improvement. Both strategies (accesses and structure) succeed in forming cohesive clusters but struggle to fully reflect the expert's domain knowledge, highlighting the difficulty of balancing technical optimizations with the practical domain-driven decomposition strategies employed by experts.

Given the insights from our evaluation of the decompositions, it is essential to delve deeper into specific aspects that will enhance our understanding of the framework. We need to consider the challenges associated with integrating new collectors, as well as the implications of using different representations for decompositions. Additionally, understanding expert evaluations will provide valuable perspectives on the quality of our results. These considerations lead us to several key research questions that will guide our investigation moving forward.

5.6 Research Question 1

What is the effort necessary to add a new collector to the existing pipeline, considering that the generated decomposition is a cluster decomposition of domain entities?

By revisiting our initial question, we aim to evaluate the effort required to add a new collector to our existing pipeline. Specifically, this question seeks to uncover the challenges and resources needed to integrate a new collector into a system where the outcome is a cluster decomposition of domain entities. This evaluation provides insights into the scalability and adaptability of our framework in real-world scenarios, particularly in the context of extending the pipeline with additional collectors.

5.6.1 Implementation of the Structural Collector

To address this research question, we enhanced our tool by incorporating support for structural analysis. This enhancement involved integrating the output of a structural collector, which provides the derived monolith representation, into our existing framework. The subsequent steps involved using this representation to generate the cluster decomposition.

5.6.2 Effort Analysis

The following metrics were considered to assess the effort required for this enhancement:

- **Classes Updated:** A total of **34 existing classes** were updated to integrate the new collector. This represents approximately **20**% of the project's **175** classes, with **15** classes on the front-end and **19** classes on the back-end.
- **Extended Classes: 16 classes** were extended to support the structural collector functionality, including **5 front-end classes** and **11 back-end classes**.
- **Lines of Code Added:** A total of **2324 lines** of code were introduced during the implementation.
- **Lines of Code Removed: 54 lines** of existing code were removed or refactored as part of the integration process.

5.6.3 Complementary Analysis

In addition to the raw metrics, it is important to consider several qualitative aspects that contribute to the overall assessment of effort:

- **Complexity of Integration:** The complexity of extending existing classes and implementing new ones can vary significantly based on the cohesion of the current system and the design of the pipeline. In this case, **34 existing classes** were updated, with **16 classes** extended specifically to support the new structural collector. Despite this, the integration process was not overly complex, as the updated and extended classes accounted for only **20%** of the total project. This suggests that the system was designed with enough modularity to accommodate such extensions without significant refactoring.
- **Time Investment:** Although the number of classes and lines of code provide a quantitative measure, the time required to design, implement, and test the new collector is also a critical factor in assessing effort. This includes the time spent understanding existing code, implementing new features, and debugging. Since the beginning of the project it has taken around 14 months of work to achieve this final stage.
- **Impact on System Stability:** The number of lines of code removed or refactored might indicate an effort to maintain or enhance the stability of the system during integration. Ensuring that new additions do not introduce regressions is a key consideration in evaluating the overall effort.

5.6.4 Conclusion

In summary, the effort to add a new collector to the existing pipeline resulted in a significant enhancement of the system's capabilities. With **34 classes updated**, **16 classes extended**, **2324 lines of code added**, and **54 lines of code removed**, the integration expanded the system without altering its core

structure. This reflects the adaptability of the framework and its ability to accommodate new functionality while maintaining its original design.

5.7 Research Question 2

Are the Decompositions Obtained Using the Monolith Structural Representation Statistically Different from the Decompositions Obtained Using the Monolith Sequences of Accesses Representation?

5.7.1 Statistical Analysis of Decompositions

To determine if there are statistically significant differences between the decompositions obtained using the monolith structural representation and those obtained using the monolith sequences of accesses representation, we conducted independent t-tests for each of the metrics: Accuracy, Precision, Recall, Specificity, F-Score, and Purity.

5.7.2 Data Summary

The values for each metric are summarized below:

- **Structural Representation:**
  - Accuracy: 0.53, 0.42, 0.49, 0.56, 0.62, 0.63, 0.63
  - Precision: 0.23, 0.25, 0.22, 0.29, 0.15, 0.15, 0.14
  - Recall: 0.00, 0.39, 0.22, 0.22, 0.34, 0.31, 0.28
  - Specificity: 0.62, 0.43, 0.62, 0.73, 0.67, 0.69, 0.69
  - F-Score: 0.26, 0.30, 0.22, 0.25, 0.21, 0.20, 0.19
  - Purity: 0.56, 0.64, 0.64, 0.64, 0.57, 0.57, 0.54
- **Sequences of Accesses Representation:**
  - Accuracy: 0.73, 0.73, 0.53, 0.56, 0.42
  - Precision: 0.27, 0.25, 0.23, 0.31, 0.25
  - Recall: 0.47, 0.40, 0.30, 0.50, 0.39
  - Specificity: 0.77, 0.79, 0.62, 0.58, 0.43
  - F-Score: 0.34, 0.31, 0.26, 0.38, 0.30
  - Purity: 0.63, 0.61, 0.56, 0.67, 0.64

5.7.3 Independent T-Test Calculations

Given that the number of observations for the structural representation and sequences of accesses representation differ in some metrics, we performed an independent t-test rather than a paired t-test. This test compares the means of two independent samples to determine if they are significantly different.

The t-test statistic is calculated as follows:

$$t = \frac{\bar{X}_{\text{struct}} - \bar{X}_{\text{access}}}{\sqrt{\frac{\sigma^2_{\text{struct}}}{n_{\text{struct}}} + \frac{\sigma^2_{\text{access}}}{n_{\text{access}}}}}$$

where:

- $\bar{X}_{\text{struct}}$ and $\bar{X}_{\text{access}}$ are the means of the structural and access representations, respectively.
- σ_{struct} and σ_{access} are the standard deviations of the structural and access representations, respectively.
- n_{struct} and n_{access} are the sample sizes for the structural and access representations, respectively.

5.7.4 P-Value Calculation

To determine the significance of the t-test results, we calculated the p-values corresponding to the computed t-values. The p-value represents the probability of obtaining a t-value at least as extreme as the one observed, assuming the null hypothesis is true. We used the cumulative distribution function of the t-distribution to calculate these p-values.

The Python code used for these calculations is provided in Appendix A.

5.7.5 Results of the T-Tests

The independent t-tests yielded the following t-values and p-values for each metric:

- **Accuracy:** $t \approx -0.64$, $p \approx 0.53$
- **Precision:** $t \approx -2.01$, $p \approx 0.07$
- **Recall:** $t \approx -2.50$, $p \approx 0.03$
- **Specificity:** $t \approx -0.03$, $p \approx 0.97$
- **F-Score:** $t \approx -3.50$, $p \approx 0.01$
- **Purity:** $t \approx -1.11$, $p \approx 0.29$

5.7.6 Conclusion

Based on the statistical analysis, we draw the following conclusions:

- **Recall:** The t-value is approximately -2.50 with a p-value of 0.03. Since the p-value is less than 0.05, this indicates that the difference in Recall between the monolith structural representation and the sequences of accesses representation is statistically significant. The negative t-value suggests that the sequences of accesses representation has a higher mean Recall compared to the structural representation, meaning it performs better in identifying relevant instances.
- **F-Score:** The t-value is approximately -3.50 with a p-value of 0.01. The p-value is below 0.05, indicating a statistically significant difference in F-Score. The negative t-value indicates that the sequences of accesses representation achieves a higher mean F-Score than the structural representation, suggesting it provides a better balance between precision and recall.

The remaining metrics—Accuracy, Precision, Specificity, and Purity—did not show statistically significant differences (p-values greater than 0.05). Therefore, we do not discuss them further in this context, as the differences observed in these metrics are not statistically significant, indicating that the performances in these aspects are similar between the two representations.

In summary, the sequences of accesses representation outperforms the structural representation in terms of Recall and F-Score. For other metrics, no significant performance differences were observed.

5.8 Research Question 3

How does an expert evaluate the best decompositions for each of the quality metrics?

This research question explores the expert's insights on the decomposition process used during the development of the Quizzes Tutor application. The objective of the interview was to understand the rationale behind key design decisions, compare the expert's approach with automated decomposition methods, and gather reflections on the future of the system decomposition.

5.8.1 Understanding the Decomposition Process

The main goal when developing the Quizzes Tutor application, according to the expert, was primarily to provide a platform for both experience and investigation. The expert, a university professor, revealed that most of the modules in the application were developed by his students as part of their Software Engineering course. Each academic year, a new module was built, adding to the microservices architecture. Besides these modules, additional microservices were developed based on dependency

relations, where core entities like questions and answers were separated into their own microservices, forming the backbone of the application.

5.8.2 Comparison with Automated Decompositions

Accesses Decompositions

Considering the automatic decompositions, the expert evaluated the accesses decomposition, which had the highest cohesion and lowest coupling. One observation made was that, despite not focusing on the color-coded clusters, the relative distances between entities allowed a clear visual understanding of the clusters. A notable point was that the expert's decomposition grouped the dashboard functionality with WeeklyScores, FailedAnswers, and DifficultQuestions—entities accessible within the dashboard. However, in the automatic decomposition, these were all separated into distinct microservices. This raised questions about the transactions that led to this separation.

As shown in Fig 5.12, the green cluster, corresponding to the authentication part of the system, was consistent with expectations. The expert also emphasized the advantage of the entity visualization over cluster visualization, noting that the entity view offered a clearer understanding of relative distances between entities. Traces of microservices could be identified, such as a question microservice in the red cluster and an answer microservice in the yellow and purple clusters, while the blue cluster somewhat represented a tournament microservice. The expert noted that transactional context does not always align with modularity in real systems, as optimizing all metrics is not always feasible. The decomposition appeared to show a tendency for retaining a monolith while extracting distinct functionalities such as authentication, questions, and answers.

The lowest complexity accesses decomposition closely resembled the highest complexity and lowest coupling decomposition, but appeared marginally better. The expert pointed out that kernel classes in the automated decompositions tend to attract other classes due to their centrality and numerous functionalities. This led to the conclusion that further exploration and interaction with the decomposition were necessary to understand the relationships and patterns, aligning with the goal of the Mono2Micro tool. The expert concluded that the differences between the two decompositions were minor and essentially similar in overall structure.

Structure Decompositions

When comparing the structure decompositions for each metric, the expert noted similarities between them, though from a visualization perspective, it was harder to discern clusters by entity distances compared to the accesses decomposition. From a modularity perspective, the structure decomposition

was less effective. The largest cluster contained a mix of entities, though the authentication cluster remained correctly identified.

From a metric standpoint, the purity of the clusters was not substantially different from the accesses decomposition, although the expert emphasized that, from an architectural point of view, the difference was significant. This suggests that purity, as a standalone metric, is not the most reliable indicator and should be considered alongside other metrics. While the accesses decomposition revealed hints of microservices, the structure decomposition made this more difficult. The conclusion drawn was that behavioral and access-based decompositions offer better modularity than structure-based ones.

5.8.3 Impact on the Current Project

When asked if any changes would be made following the review of the automated decompositions, the expert responded in the negative. Although the accesses decomposition favored metric optimization at the expense of some modularity, it was not enough to warrant changes to the original expert-designed decomposition. The original design still held up well, considering the trade-offs involved.

5.8.4 Conclusion

In summary, the interview provided key insights into the decomposition process, highlighting the challenges and trade-offs involved in balancing cohesion, coupling, and modularity. The expert valued the ability to interact with and analyze the decomposition to understand its underlying structure, and while some compromises were made for metric optimization, the original decomposition remained the preferred choice. Future directions may involve refining tools like Mono2Micro to enhance the interaction and exploration of decompositions, helping architects make more informed decisions.

6

Conclusion

Contents

This thesis explored the challenges and benefits involved in the decomposition of monolithic systems for microservices identification, focusing on the integration of a new structural collector into an existing pipeline. The primary objectives were to evaluate the effort required for this integration, analyze the resulting decompositions compared to expert-driven decompositions, and assess the practical implications of such integrations in real-world applications.

6.1 Summary of Contributions

The main contributions of this thesis are as follows:

- **Pipeline Extension:** We successfully extended the microservice identification pipeline by incorporating a structural collector. This collector generated a representation of the monolith based on structural data, offering a new representation of monoliths in the pipeline.

- **Effort and Complexity Analysis:** Through the analysis of the code changes, including class extensions and the lines of code refactored, we evaluated the development effort required to integrate the new collector. The results indicated that while adding a new collector required significant modifications, the complexity of integration was manageable due to the modular design of the pipeline.
- **Decomposition Evaluation:** We conducted a thorough evaluation of the structural decompositions generated by the pipeline, comparing them with expert decompositions. Using key metrics such as Accuracy, Precision, Recall, Specificity, and F-score, we found that while structural decompositions showed moderate alignment with expert decompositions, there were clear limitations, particularly in precision and recall.
- **Comparative Study:** We compared structural and access-based decompositions, highlighting the advantages and shortcomings of both approaches. The access-based decompositions generally produced better results, especially when minimizing complexity and coupling, though they did not fully align with expert knowledge.

6.2 Reflection on Research Questions

Research Question 1: Effort Required for Integration

The integration of a structural collector into the existing pipeline required updating approximately 20% of the project's classes and adding over 2,300 lines of code. Despite the extensive changes, the modularity of the pipeline design helped mitigate integration complexity, demonstrating that the pipeline can be extended without compromising system stability.

Research Question 2: Comparison of Decomposition Results

The decompositions generated using the structural collector were compared with those generated by the access-based collectors. While the structural approach produced moderate accuracy, it often fell short in precision and recall. On the other hand, the access-based decompositions, particularly those optimized for low complexity and coupling, showed better alignment with expert decompositions. This suggests that a hybrid approach may yield the best results for future decomposition strategies.

Research Question 3: Expert Evaluation of Decompositions

In the interview conducted with the expert behind the Quizzes Tutor application, several important insights were gathered regarding the decomposition process. The expert emphasized that the primary goal of the application's decomposition was to balance modularity with the flexibility to scale the system

over time. The development process, involving incremental additions by students, resulted in microservices structured around core domain entities like questions and answers.

When comparing automated decompositions with the expert's original design, the access-based decomposition generally provided better modularity, with clusters that aligned more closely to the microservices in the expert's decomposition. The expert also highlighted that one advantage of using access-based decompositions was that it allowed for more intuitive insights based on the relative distances between entities.

The structural decompositions, while offering a different perspective, were seen as less effective for identifying clear boundaries between microservices. The metrics used by automated tools, such as cohesion and coupling, were not always aligned with the modularity needed for real-world systems, which often require trade-offs in performance, scalability, and maintainability.

Ultimately, the expert concluded that while the automated decompositions provided interesting insights, they were not sufficient to replace expert-driven decompositions entirely. However, they could still serve as a valuable tool for refining system designs, provided there was the ability to interact and explore the decompositions more deeply.

6.3 Future Work

The evaluation of the structural and access-based collectors provides a solid foundation for future research. Several opportunities for future work have been identified:

- **Improving Decomposition Alignment:** Future work could explore methods to better align automated decompositions with expert-driven insights, possibly through Artificial Intelligence-based approaches.
- **Hybrid Collectors:** Investigating a hybrid decomposition strategy that combines structural and access-based data could provide better decompositions by leveraging the strengths of both approaches.
- **Refinement of Metrics:** While the purity metric and other traditional metrics provided valuable insights, further refinement or development of new metrics may improve decomposition evaluation.
- **Improved Similarity Measures for Structural Decompositions:** Developing new and more effective similarity measures for structural decompositions could result in more accurate decompositions that better align with expert-driven designs. By improving the way structural relationships are quantified, automated methods could potentially capture the nuances of expert intuition and produce decompositions that are closer to the optimal modularity and cohesion required in real-world systems.

In conclusion, this research demonstrates the feasibility of integrating new collectors into microservice identification pipeline and provides valuable insights into the performance of structural versus access-based decompositions. While challenges remain, particularly in fully capturing expert intuition, the findings of this thesis increase the groundwork for future advancements in monolith decomposition and microservice identification.

Bibliography

[1] T. Lopes, “Monolith microservices identification: An extensible multiple strategy tool,” Master’s thesis, Instituto Superior Técnico, University of Lisbon, 2022.

[2] 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)*, 2023, pp. 111–115.

[3] J. Thönes, “Microservices,” *IEEE Software*, vol. 32, no. 1, pp. 116–116, 2015.

[4] C. O’Hanlon, “A conversation with werner vogels: Learning from the amazon technology platform: Many think of amazon as ’that hugely successful online bookstore.’ you would expect amazon cto werner vogels to embrace this distinction, but in fact it causes him some concern.” *Queue*, vol. 4, no. 4, p. 14–22, may 2006. [Online]. Available: https://doi.org/10.1145/1142055.1142065

[5] Y. Abgaz, A. McCarren, P. Elger, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yilmaz, J. Buckley, and P. Clarke, “Decomposition of monolith applications into microservices architectures: A systematic review,” *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4213–4242, 2023.

[6] L. Nunes, N. Santos, and A. Rito Silva, “From a monolith to a microservices architecture: An approach based on transactional contexts,” in *Software Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds. Cham: Springer International Publishing, 2019, pp. 37–52.

[7] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki, “A multi-model based microservices identification approach,” *Journal of Systems Architecture*, vol. 118, p. 102200, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121001442

[8] O. Al-Debagy, “A microservice decomposition method through using distributed representation of source code,” *Scalable Comput. Pract. Exp.*, vol. 22, pp. 39–52, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231878837

[9] U. Desai, S. Bandyopadhyay, and S. Tamilselvam, “Graph neural network to dilute outliers for refactoring monolith application,” 2021.

[10] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, “Microservice remodularisation of monolithic enterprise systems for embedding in industrial iot networks,” in *Advanced Information Systems Engineering*, M. La Rosa, S. Sadiq, and E. Teniente, Eds. Cham: Springer International Publishing, 2021, pp. 432–448.

[11] 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)*, 2018, pp. 571–580.

[12] G. Mazlami, J. Cito, and P. Leitner, “Extraction of microservices from monolithic software architectures,” in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 524–531.

[13] M. Brito, J. Cunha, and J. a. Saraiva, “Identification of microservices from monolithic applications through topic modelling,” in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, ser. SAC ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1409–1418. [Online]. Available: https://doi.org/10.1145/3412841.3442016

[14] S. Agarwal, R. Sinha, G. Sridhara, P. Das, U. Desai, S. Tamilselvam, A. Singhee, and H. Nakamuro, “Monolith to microservice candidates using business functionality inference,” in *2021 IEEE International Conference on Web Services (ICWS)*, 2021, pp. 758–763.

[15] T. Matias, F. F. Correia, J. Fritzsch, J. Bogner, H. S. Ferreira, and A. Restivo, “Determining microservice boundaries: A case study using static and dynamic software analysis,” 2020.

[16] S. Eski and F. Buzluca, “An automatic extraction approach: Transition to microservices architecture from monolithic application,” in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, ser. XP ’18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3234152.3234195

[17] W. K. G. Assunção, T. E. Colanzi, L. Carvalho, J. A. Pereira, A. Garcia, M. J. de Lima, and C. Lucena, “A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study,” in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 377–387.

[18] 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)*, 2023, pp. 47–57.

[19] 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)*, 2020, pp. 65–71.

[20] 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)*, 2020, pp. 95–100.

[21] 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, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219301475

[22] 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, pp. 987–1007, 2021.

[23] I. Pigazzini, F. Arcelli Fontana, and A. Maggioni, “Tool support for the migration to microservice architecture: An industrial case study,” in *Software Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds. Cham: Springer International Publishing, 2019, pp. 247–263.

[24] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, “Towards automated microservices extraction using muti-objective evolutionary search,” in *Service-Oriented Computing*, S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari, Eds. Cham: Springer International Publishing, 2019, pp. 58–63.

[25] M. Abdullah, W. Iqbal, and A. Erradi, “Unsupervised learning approach for web application auto-decomposition into microservices,” *Journal of Systems and Software*, vol. 151, pp. 243–257, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300408

[26] D. Taibi and K. Systä, “A decomposition and metric-based evaluation framework for microservices,” 2019.

[27] A. Bucchiarone, K. Soysal, and C. Guidi, “A model-driven approach towards automatic migration to microservices,” in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds. Cham: Springer International Publishing, 2020, pp. 15–36.

[28] 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)*, 2020, pp. 135–145.

[29] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, “Function-splitting heuristics for discovery of microservices in enterprise systems,” in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds. Cham: Springer International Publishing, 2018, pp. 37–53.

[30] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, and T. Huang, “Migrating web applications from monolithic structure to microservices architecture,” in *Proceedings of the 10th Asia-Pacific Symposium on Internetware*, ser. Internetware ’18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3275219.3275230

[31] 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)*, 2020, pp. 289–294.

[32] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kröger, “Microservice decomposition via static and dynamic analysis of the monolith,” 2020.

[33] R. Chen, S. Li, and Z. Li, “From monolith to microservices: A dataflow-driven approach,” in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 466–475.

[34] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, “Partial migration for re-architecting a cloud native monolithic application into microservices and faas,” in *Information, Communication and Computing Technology*, C. Badica, P. Liatsis, L. Kharb, and D. Chahal, Eds. Singapore: Springer Singapore, 2020, pp. 111–124.

[35] L. Baresi, M. Garriga, and A. De Renzis, “Microservices identification through interface analysis,” in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 19–33.

[36] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, “Remodularization analysis for microservice discovery using syntactic and semantic clustering,” in *Advanced Information Systems Engineering*, S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. Pant, Eds. Cham: Springer International Publishing, 2020, pp. 3–19.

[37] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, “Service cutter: A systematic approach to service decomposition,” in *Service-Oriented and Cloud Computing*, M. Aiello, E. B. Johnsen, S. Dustdar, and I. Georgievski, Eds. Cham: Springer International Publishing, 2016, pp. 185–200.

[38] 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)*, 2020, pp. 157–168.

[39] S. Santos and A. R. Silva, “Microservices identification in monolith systems: Functionality redesign complexity and evaluation of similarity measures,” *Journal of Web Engineering*, vol. 21, no. 05, p. 1543–1582, Aug. 2022. [Online]. Available: https://journals.riverpublishers.com/index.php/JWE/article/view/11745

[40] E. Amigo, J. Gonzalo, J. Artiles, and F. Verdejo, “A comparison of extrinsic clustering evaluation metrics based on formal constraints,” *Information retrieval*, vol. 12, pp. 461–486, 2009.

A

Code for Statistical Analysis

This appendix includes the Python code used for performing the independent t-tests to compare the performance metrics of the two decomposition representations: the monolith structural representation and the monolith sequences of accesses representation.

A.1 Python Code for Statistical Analysis

The following Python code was used to calculate t-values and p-values for each metric:

Listing A.1: Python Code for T-Test Analysis

```
import scipy.stats as stats

# Sample data
structural = {
    'accuracy': [0.53, 0.42, 0.49, 0.56, 0.62, 0.63, 0.63],
    'precision': [0.23, 0.25, 0.22, 0.29, 0.15, 0.15, 0.14],
    'recall': [0.00, 0.39, 0.22, 0.22, 0.34, 0.31, 0.28],
    'specificity': [0.62, 0.43, 0.62, 0.73, 0.67, 0.69, 0.69],
```

```
    'fscore': [0.26, 0.30, 0.22, 0.25, 0.21, 0.20, 0.19],
    'purity': [0.56, 0.64, 0.64, 0.64, 0.57, 0.57, 0.54]
}

accesses = {
    'accuracy': [0.73, 0.73, 0.53, 0.56, 0.42],
    'precision': [0.27, 0.25, 0.23, 0.31, 0.25],
    'recall': [0.47, 0.40, 0.30, 0.50, 0.39],
    'specificity': [0.77, 0.79, 0.62, 0.58, 0.43],
    'fscore': [0.34, 0.31, 0.26, 0.38, 0.30],
    'purity': [0.63, 0.61, 0.56, 0.67, 0.64]
}

# Function to perform t-test and calculate p-value
def calculate_t_p_values(metric_name):
    # Perform an independent t-test
    t_statistic, p_value = stats.ttest_ind(structural[metric_name],
    accesses[metric_name], equal_var=False)
    return t_statistic, p_value

# Calculate and print t-values and p-values for each metric
metrics = ['accuracy', 'precision', 'recall', 'specificity', 'fscore', 'purity']

for metric in metrics:
    t_stat, p_val = calculate_t_p_values(metric)
    print(f"Metric: {metric.capitalize()}\n t-value: {t_stat:.2f},
    p-value: {p_val:.2f}\n")
```

The provided code calculates the t-values and p-values for the metrics of Accuracy, Precision, Recall, Specificity, F-Score, and Purity by comparing the results from the two different representations.

A.2 Explanation of Results

In the context of statistical significance, the t-values and p-values help determine whether the differences observed in the metrics between the two representations are statistically significant. A p-value less than 0.05 indicates that the difference is statistically significant. Specifically:

- For metrics where the p-value is less than 0.05, such as Recall and F-Score, the differences are considered statistically significant. This suggests that one representation performs better than the

other on these metrics.

- For metrics where the p-value is greater than 0.05, such as Accuracy, Precision, Specificity, and Purity, the differences are not statistically significant, indicating no clear performance advantage between the two representations for these metrics.