Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 21(4): 268-280, Dec. 2023 Regular paper

Microservice Identification by Partitioning Monolithic Web Applications Based on Use-Cases

Si-Hyun Kim[1], Daeil Jung[1], Norhayati Mohd Ali[2], Abu Bakar Md Sultan[2], and Jaewon Oh[1]*

[1]School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon 14662, Republic of Korea
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Malaysia

Abstract

Several companies have migrated their existing monolithic web applications to microservice architectures. Consequently, research on the identification of microservices from monolithic web applications has been conducted. Meanwhile, the use-case model plays a crucial role in outlining the system's functionalities at a high level of abstraction, and studies have been conducted to identify microservices by utilizing this model. However, previous studies on microservice identification utilizing use-cases did not consider the components executed in the presentation layer. Unlike existing approaches, this paper proposes a technique that considers all three layers of web applications (presentation, business logic, and data access layers). Initially, the components used in the three layers of a web application are extracted by executing all the scenarios that constitute its use-cases. Thereafter, the usage rate of each component is determined for each use-case and the component is allocated to the use-case with the highest rate. Then, each use-case is realized as a microservice. To verify the proposed approach, microservice identification is performed using open-source web applications.

Index Terms: Monolithic applications, Microservices, Microservice identification, Use-cases, Web application reengineering

I. INTRODUCTION

These days, there has been a continuous increase in the amount of data that needs to be processed by web applications (web apps), and the requirements for web apps are becoming more and more complex [1]. Under these circumstances, the modularization of web apps makes it easier to maintain increasingly complex web apps, and it can improve their performance [2]. Hence, various approaches, such as a method to partition web apps into layers [3] and a method to transform web apps into service-oriented architectures [4], have been proposed as methods for the modularization of web apps [4-19]. In addition, to achieve modularization, several companies (including Netflix, Amazon, and eBay) have recently migrated their monolithic web apps to microservice architectures [5]. In this context, the term *microservice architecture* refers to an architecture composed of microservices, which are small apps that are independently developed, released, and expanded [6]. The transformation of a monolithic web app to a microservice architecture is a challenging task [8] since it requires the identification of reusable features from the monolithic web app [4].

Hence, to alleviate the difficulties in the transformation process, research has focused on methods to automatically identify microservices within monolithic web apps [9-16,18, 19]. However, in many prior studies, components such as database tables and views in the data access and presentation layers of web apps were overlooked in the identification of microservices. As a result, existing techniques were not sufficient to ensure that each of the identified microservices can

Received 19 September 2023, **Revised** 14 November 2023, **Accepted** 30 November 2023
***Corresponding Author** Jaewon Oh (E-mail: jwoh@catholic.ac.kr)
School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon 14662, Republic of Korea

Open Access https://doi.org/10.56977/jicce.2023.21.4.268 print ISSN: 2234-8255 online ISSN: 2234-8883

properly perform its function as a web app.

To address these problems, this paper proposes a method for transforming monolithic web apps into microservice architectures by achieving the following three research objectives.

Microservice identification utilizing the use-case model: To transform a monolithic web app into a microservice architecture, it is essential to determine the parts of the web app that are converted into microservices. The main goal of this transformation is to ensure that each identified microservice provides a single function. To accomplish this, we utilize a use-case model outlining the functional requirements of a system. In other words, our approach extracts a use-case model from a monolithic web app and maps each use-case to a microservice.

Identification of microservices composed of all three web app layers: A web app generally comprises three layers: presentation, business logic, and data access [20]. Similarly, when constructing a web app based on the microservice architecture it is essential to ensure that each microservice can function as a small independent web app [6]. Consequently, each microservice must encompass all components running in these three layers for complete web app functionality. Considering this aspect of microservices in the identification process, this paper introduces a microservice identification technique that, through dynamic analysis, considers all components executed in the three layers of web apps. It ensures the inclusion of these components in the identified microservices. It is worth noting that, unlike prior studies utilizing use-cases for microservice identification, this paper accounts for components executed in the presentation layer.

Evaluation: To assess the applicability and effectiveness of the proposed technique, it is essential to compare it with various existing microservice identification techniques. The comparison between our approach and existing methods is made using five types of open-source web apps. Additionally, an evaluation is conducted using an accuracy metric widely used in machine learning.

This paper makes the following contributions by accomplishing the aforementioned research objectives. First, it decomposes a monolithic web app based on its functions by utilizing use-cases for microservice identification. Second, all three layers of the web apps are considered. Third, this paper demonstrates that our approach outperforms existing methods.

The remainder of this paper is organized as follows. Section II describes related studies. Section III presents a method for microservice identification. In Section IV, the effectiveness of our approach is evaluated. Finally, Section V provides conclusions and directions for future research.

II. MOTIVATION AND RELATED WORK

According to a related study [7], microservice identification methods or Service Identification Approaches (SIAs) can be divided into six types: wrapping, genetic algorithms, Formal Concept Analysis (FCA), clustering, custom heuristics, and general guidelines. In particular, clustering and custom heuristics have been the preferred approaches in recent studies [7].

In SIAs, microservices can be identified using various input data. In [7], the input data were classified into three categories: executable models (source code, databases, etc.), non-executable models (execution traces, use-cases, etc.), and domain artifacts (ontologies, documents, etc.).

Our approach uses classes, tables, and views as executable models, and execution traces and use-cases as non-executable models. Consequently, it is necessary to review studies that utilize these two types of models.

First, studies that identify microservices using use-cases include the following. Dmitry et al. [12] introduced a method using use-cases to partition a monolithic web app into components with a single function. This technique is similar to ours, as it identifies microservices by dividing web apps based on use-cases. However, their method lacked evaluation using real web apps, and it did not account for tables and views.

In a study by Bajaj et al. [13], relationships between use-cases and those between use-cases and tables were extracted. The relationships were then analyzed to identify microservices. However, as this technique focuses on microservice identification at the use-case level, it does not take into account classes and views.

In the approach presented by Tyszberowicz et al. [15], the functional requirements of the system were analyzed using use-cases. Based on this analysis, they proposed a method for identifying microservices by partitioning web apps by function. This method is similar to our approach in that it considers tables in the process of identifying microservices. However, they did not consider views such as JSP.

Furthermore, Kalia et al. [18] proposed a method to extract execution traces based on use-cases and subsequently analyze the traces to depict the call relationships between classes in a graph. This study is similar to our approach in that class information is obtained from execution traces. However, their method is constrained since table and view information cannot be obtained.

Second, it is necessary to review studies that consider not only classes but also tables and views. Some previous studies [9-11,21-23] considered both classes and tables to identify microservices.

Levcovitz et al. [9] introduced a method to represent components including database tables and business logic, and

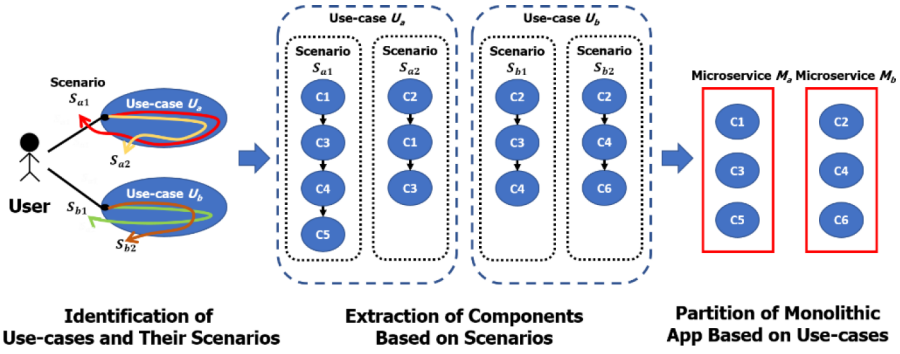

Fig. 1. Microservice identification approach.

call relationships between components, in the form of a graph through static analysis of monolithic web apps. A limitation of their approach, in contrast to ours, is the inability to identify dynamically generated tables due to its reliance on static analysis. Moreover, it does not consider cases where multiple microservices share tables.

Kamimura et al. [10] proposed a method to represent classes and tables as nodes and call relationships between nodes as edges through static analysis of the source code of web apps. This technique considers only web apps that use Object Relational Mapping (ORM[1]) in the process of identifying tables; therefore, it can not be used to identify tables in the case of apps not using ORM.

Mazlami et al. [11] presented a method for utilizing the information from version control systems to identify microservices. Their approach generates a graph where classes and tables are nodes, and the relationships between nodes are edges. The graph is then clustered to identify microservices. At that time, edges are generated based on the change history of classes obtained from the version control system, from information on contributors who changed the classes, or from information on source code similarity between classes obtained by static analysis of the source code. However, this method is limited to apps using ORM, rendering it unable to identify tables in apps without ORM.

Del Grosso et al. [21] proposed a technique to identify microservices by dynamically extracting SQL queries to obtain database tables and fields and clustering the queries with the obtained information. This method is similar to ours in that it uses dynamic analysis to extract SQL queries. However, because their approach identifies microservices using only SQL queries, it does not consider classes and views.

Recent studies [22,23] have expanded their focus beyond classes to include tables and views. These studies highlight that certain components in web apps are overlooked by existing microservice identification methods. Thus, those studies proposed the following process to solve the problem. In the first step, any existing microservice identification method is executed. In the subsequent step, the components not considered by the existing method are classified as microservices by reusing the identification results of the previous step. The tables and views are classified in the subsequent step. While these studies directly classify all components into microservices, these studies reuse the results of other microservice identification techniques for this classification. In addition, these studies statically analyze the source code, unlike this paper.

1) https://docs.spring.io/spring-framework/docs/2.5.5/reference/orm.html

III. MICROSERVICE IDENTIFICATION VIA USE-CASES

This paper introduces a method for transforming a monolithic web app into a microservice architecture where each identified microservice functions as an independent web app with its unique purpose. A use-case represents a function provided by a system at a high level of abstraction from the user's perspective [12,13]. Therefore, this paper proposes a microservice identification approach that realizes each use-case as a microservice.

As depicted in Fig. 1, the proposed approach consists of three steps: identifying use-cases and their scenarios, extracting components based on these scenarios, and clustering the components according to the use-cases. In this figure, the system consists of two use-cases (U_a and U_b). Each use-case comprises two scenarios. By executing each scenario, its components are identified in the second step. Each component is allocated to an appropriate use-case, and each use-case is realized as a microservice in the final step.

A. Identification of Use-cases and Scenarios

A use-case is a set of scenarios, each of which is a series of steps through which a web app is used. Consequently, the identification of use-cases also requires the identification of their scenarios. This paper proposes the following method

for identifying use-cases and their scenarios.

First, use-cases and their scenarios can be statically identified by referring to the web app documentation. Documents for web apps can be found in various locations. For example, for open-source apps publicly available in an open-source repository (e.g., GitHub[2]), a file with a description of the web app (e.g., HELP.html and README.md) is sometimes provided. In such cases, use-cases and their scenarios can be identified through static analysis of the file.

Fig. 2 shows the process of extracting use-cases by referring to the HELP.html file in the open-source web app called JPetStore6[3]. The use-case called *Account Management* can be statically identified by referring to the functions called *Signing Up* and *Signing In* described in this document.

Second, use-cases and scenarios can be dynamically identified through the execution of a web app. Analyzing documents for the extraction of use-cases and scenarios may encounter limitations due to insufficient information in the documents or the absence of relevant documents. In such cases, use-cases and scenarios can be identified by executing a web app.

For example, in the scenario identification step of Fig. 2, the *Failed Login* scenario is not specified in the HELP.html file, but can be dynamically identified by executing the web app. On the other hand, the *Successful Login* scenario is statically identified in the file.

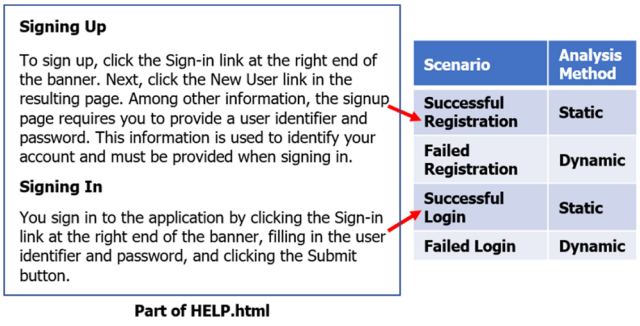

Fig. 2. The extraction of the use-cases and their scenarios of JPetStore6.

B. Extracting Components from Scenarios

A scenario is one of the operational flows within a web app. Thus, an execution trace can be extracted for each scenario. In this context, the execution trace refers to a list of components used during scenario execution. The Aspect Oriented Programming (AOP) technique [24,25] can be used to extract such execution traces. As shown in Fig. 3, when employing AOP in Java web app, execution traces can be generated by detecting the execution of the classes and methods constituting the web app. In this figure, the web app consists of components (c_1-c_6) developed by developers and existing components (lib_1-lib_4) provided in the framework. Scenario s_{a1} is used to identify the components developed by developers among those running in this scenario. In this figure, c_1, c_3, c_4, and c_5 are identified to constitute the execution trace using the aspects of the AOP.

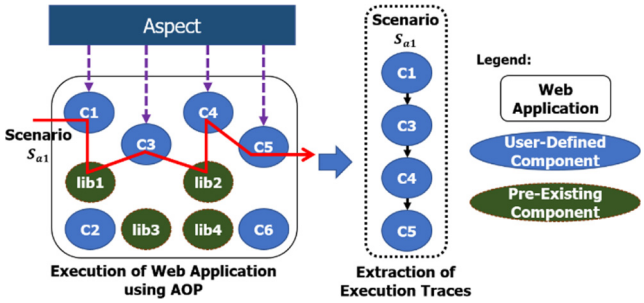

Fig. 3. Extracting execution traces by utilizing AOP.

1) Extracting Execution Traces with Aspectj

AspectJ [24] is one of the technologies supporting AOP in Java apps and is a library that can be used to extract execution traces. As shown in Fig. 4, the execution traces can be extracted by adding to a Java web app the special construct called an *aspect* provided by AspectJ. This aspect can contain a pointcut annotation that is used to specify the classes and methods to be recorded in the execution traces. Fig. 4 shows part of the aspect code used to extract the execution traces of JPetStore6. Fig. 5 shows some of the extracted execution traces from running the aspect code. To trace the execution of the classes written by the developers, they are

```
//User-defined classes and packages
1 @Pointcut("(within(org.mybatis.jpetstore.web.actions..*) ||" +
2 "within(org.mybatis.jpetstore.service.CatalogService) ||" +
3 "within(org.mybatis.jpetstore.service.OrderService) ||" +
4 "within(org.mybatis.jpetstore.service.AccountService) ||" +
// Location of the compiled JSPs
5 "within(org.apache.jsp..*) ||"+
// JDBC API
6 "call(boolean java.sql.Statement.execute(String,int)) ||" +
7 "call(int java.sql.Statement.executeUpdate(String)) ||" +
8 "call(boolean java.sql.Statement.executeQuery(String)) ||" +
...
```

Fig. 4. An AspectJ pointcut for JPetStore6.

```
1 <AccountService, Class>
2 <AccountMapper, Class>
3 <ACCOUNT, Table>
4 <PROFILE, Table>
5 <SIGNON, Table>
6 <BANNERDATA, Table>
...
7 <main, JSP>
```

Fig. 5. The execution trace of the successful login scenario for JPetStore6.

2) http://github.com/
3) https://github.com/mybatis/jpetstore-6

specified within the pointcut construct in lines 1-4 of Fig. 4. Only the classes written by the developers are specified because the execution records of the components not written by them are not required during the identification of microservices.

When executing a *Successful Login* scenario using the aspect in Fig. 4, classes like *AccountService* and *AccountMapper* can be extracted, as depicted in lines 1-2 of Fig. 5. Lines 3-7 of Fig. 5 display the tables and views detected. To detect tables and views, it is necessary to perform an additional step that is explained in detail below.

2) Identification of Tables

This paper considers database tables in the data access layer of web apps. For this purpose, it is essential to identify tables accessed during the execution of the scenarios.

In general, we can obtain information on tables by analyzing SQL queries. The methods used to generate SQL queries in web apps can be divided into three types. First, it is possible to completely generate SQL queries at compile time if the developer directly writes them in the source code. Second, while an app is executed, SQL queries can be dynamically generated by utilizing user input. Third, SQL queries can be statically or dynamically generated using database frameworks (e.g., MyBatis[4] and Spring Data JPA[5]).

Regarding the above methods, approaches to extracting SQL queries can be divided into two types. The first method uses static analysis. In a study by Meurice et al. [26], static analysis of source code was performed to identify a method of a class that accesses database tables. The SQL query, treated as a value in the method argument, is then statically extracted. There is a limitation in the technique proposed by Meurice et al., which assumes the source code includes static information on tables; therefore, it does not consider cases in which tables are dynamically generated.

The second type of approach uses dynamic analysis. In a study by Cleve et al. [27], the execution of a method of a class used to access database tables was identified, and the argument value of this method was dynamically extracted to obtain a SQL query. If this approach is used, cases in which tables are generated dynamically can be considered.

Our approach is a dynamic analysis technique. Nevertheless, it is different from the study by Cleve et al. in that our approach considers cases in which SQL queries are generated by using database frameworks.

According to one previous study [28], most Java database frameworks are implemented based on the Java Database Connectivity (JDBC)[6] Application Programming Interface (API); therefore, access to tables can be discerned by tracing the execution of the JDBC API. In our approach, AspectJ is used to detect the execution of classes. Similarly, to detect access to tables, the JDBC API with a SQL query as its argument is specified within the pointcut of the aspect class. Subsequently, when the execution of this API is detected dynamically, the SQL query used as the argument value of this API can be extracted. These JDBC methods can be found on the website for Kieker[7], one of the dynamic analysis tools.

Line 2 in Fig. 5 shows the class *AccountMapper*, which calls the *execute()* method of the JDBC API. The argument value of this method includes the following query:

select ... from ACCOUNT, PROFILE, SIGNON, BANNERDATA where ...

Our approach uses line 6 of Fig. 4 to detect the *execute()* method. The argument of this method is subsequently parsed, and the tables mentioned above are detected, as illustrated in lines 3-6 of Fig. 5. Parsing tasks can be accomplished using tools such as General SQL Parser (GSP)[8].

3) Identification of Views

Web apps generally employ views to display the outcomes of business logic. In Java web apps, views can be written using two methods. The first is the standard method of utilizing JSP and HTML. The second uses presentation frameworks (such as Stripes[9]).

Therefore, to identify views, it is necessary to use a technique that is suitable for the presentation method applied to web apps. In this paper, the following two approaches described below are used to identify views.

First, when employing JSP and HTML in a standard Java web app, views are identified as follows. The JSP is executed after being converted into a servlet class by the web server. Since the final JSP form is a servlet class, the JSP can be identified if the final servlet class is specified within the pointcut of the aspect class, as shown in line 5 of Fig. 4.

HTML can not be identified using AspectJ because it is not a Java class. In this case, the interceptor pattern [29] can be used to detect HTML. In other words, this pattern allows the interceptor written as a Java class to run before HTML page *h* is accessed. Additionally, the interceptor obtains information about *h* and records it in an execution trace. This interceptor pattern can be implemented using the filter and wrapper functions [30] provided by standard Java web technology.

Second, in web apps that employ presentation frameworks such as FreeMarker[10], a distinct class method is used to execute JSP and HTML. By specifying this particular method within the AspectJ pointcut, views such as JSP and HTML

4) https://mybatis.org/mybatis-3/
5) https://spring.io/projects/spring-data-jpa
6) https://www.oracle.com/java/technologies/javase/javase-tech-database.html
7) https://github.com/kieker-monitoring/kieker
8) https://www.sqlparser.com
9) https://github.com/StripesFramework
10) https://freemarker.apache.org

can be identified. For instance, in FreeMarker, the *getTemplate()* method of the *Configuration* class holds information about the HTML page to be used as a view. Hence, detecting *getTemplate()* and analyzing its method argument can identify an HTML page.

In JPetStore6, which uses Stripes as its presentation framework, the constructor of the *ForwardResolution* class contains information on the view in its argument. Thus, line 7 in Fig. 5 shows that the *main* view is identified by detecting the value of the constructor's argument.

c. Component Clustering Based on Use-cases

In this paper, each use-case is realized as a microservice. Specifically, the clustering of a web app's components is conducted based on its use-cases, and a set of components belonging to each cluster (i.e., a use-case) is implemented as a single microservice. In this phase, to identify the components that belong to microservices, this paper uses information obtained in Section III.A and III.B, defined as follows.

Definition 1: The information used for microservice identification is as follows.

- $U = \{u_1, u_2, ..., u_n\}$: a set of use-cases for a web app
- $S_i = \{s_{i1}, s_{i2}, ..., s_{im}\}$: a set of scenarios for use-case u_i
- $C = \{c_1, c_2, ..., c_k\}$: a set of components in a web app
- $C_{ij} \subset C$: C_{ij} denotes a set of components used in scenario s_{ij}. In other words, it is a set of views, tables, and classes accessed when s_{ij} is executed.

This paper assumes that a microservice should serve a single function; hence, it is considered an appropriate approach to separate the components based on use-cases. From this perspective, the method for allocating components to use-cases is shown in Fig. 6.

This algorithm computes the usage rate of each component for each use-case and allocates the component to the use-case with the highest usage rate. The usage rate of a component for a use-case is determined by the ratio of scenarios using the component within the use-case (see line 5 in Fig. 6). For example, for a specific use-case, if a particular component is used in all scenarios belonging to the use-case, the usage rate is one and if the component is used in half of the scenarios, the usage rate is 0.5. A high usage rate of a component for a use-case indicates that the component is a key component of the use-case. Thus, the usage rate serves as an appropriate criterion for allocating a component to the specific use-case.

If two or more use-cases have the same usage rate (lines 11-16), the number of times the component is used is calculated, and the component is allocated to the use-case with the most uses. The number of uses of a component for a use-case is the number of scenarios in which the component is used from among all the scenarios constituting the use-case.

If the number of uses is primarily considered, rather than the usage rate, even though a component is used in all the scenarios of use-case u, it may be allocated to another use-case if u consists of fewer scenarios. Therefore, the usage rate is regarded as the primary criterion before considering the number of uses. The method for computing the number of uses is outlined in the *countScenarios* function in lines 18-25. In this regard, when determining the number of uses, the frequency of a component's execution in a particular scenario is not considered. In other words, for a scenario, if a component is executed once or more, the value for the number of uses is incremented by one only, and if it is not executed at all, the value is not increased (lines 21-23).

```
Input: component c in C, the set U of use-cases, and the set S_i of scenarios in u_i for each u_i ∈ U
Output: The use-case to which component c is assigned
1  num_i   // denotes the number of scenarios in S_i that use c
2  rate_i  // denotes the ratio of scenarios in S_i that use c
3  foreach use-case u_i ∈ U do
4     num_i ← countScenarios(c, S_i)
5     rate_i ← num_i / |S_i|
6  end
7  maxRateUsecases ← argmax_{u_i ∈ U} rate_i
8  if |maxRateUsecases| = 1 then
9     Assign c to the unique use-case in maxRateUsecases
10 else
11    maxNumUsecases ← argmax_{u_i ∈ maxRateUsecases} num_i
12    if |maxNumUsecases| = 1 then
13       Assign c to the unique use-case in maxNumUsecases
14    else
15       Assign c to a use-case randomly selected from
         maxNumUsecases
16    end
17 end
18 Function countScenarios(c, S_i)
19    cnt ← 0
20    foreach scenario s_ij ∈ S_i do
21       if c is used in s_ij ∈ then
22          cnt ← cnt + 1
23       end
24    end
25 return cnt
```

Fig. 6. Clustering of components based on use-cases.

Fig. 7 illustrates a segment of the microservice identification process for JPetStore6. To determine the appropriate use-case for allocating the *AccountActionBean* component should be allocated to, the *AccountActionBean* component is input for the proposed clustering algorithm. The results for lines 3-6 of the algorithm are shown in Fig. 7(a). For the *AccountActionBean* component, the use-cases with the highest usage rate are *Order Management* and *Account Management*. However, the *AccountActionBean* component needs to be allocated to a single use-case; therefore, the numbers of uses (*num*) are compared. As shown in Fig. 7(a), the number

of uses for *AccountActionBean* is 7 for *Order Management* and 9 for *Account Management*; consequently, this component is allocated to *Account Management*. In addition, the *CatalogActionBean* component is used in *Catalog Management* and *Account Management* use-cases, as shown in Fig. 7(b). This component is allocated to *Catalog Management*, which has the highest usage rate among these use-cases.

Component	Use-cases	Num	Rate
AccountActionBean	Catalog Management	0	0.00%
	Cart Management	0	0.00%
	Order Management	7	100.00%
	Account Management	9	100.00%

(a) Usage rates and the numbers of uses of *AccountActionBean*

Component	Use-case				Recommended Microservice
	Catalog Management	Cart Management	Order Management	Account Management	
Account-ActionBean	0.00%	0.00%	100.00%	100.00%	Account Management
AccountMapper	0.00%	0.00%	0.00%	33.33%	Account Management
AccountService	0.00%	0.00%	0.00%	33.33%	Account Management
Catalog-ActionBean	100.00%	0.00%	0.00%	33.33%	Catalog Management

(b) Components and the microservices to which they are allocated

Fig. 7. Application of the microservice identification method to JPetStore6.

IV. EVALUATION

This section describes experiments conducted to evaluate the proposed approach. It addresses the following research questions.

RQ1: Is our approach more effective than existing methods?

RQ2: Are tables and views appropriately allocated to microservices?

a. Subject Apps

The performance of the proposed approach was assessed using the open-source web apps listed in Table 1. These apps were chosen for several reasons. First, they are publicly available in well-known open-source repositories, and JPetStore6 and PetClinic have been used in previous studies [10,11,14, 18]. Second, they were selected to evaluate the effectiveness of our approach across different types of web apps (e.g., JPetStore2, JPetStore6, PetClinic) using various frameworks as well as standard Java web apps (e.g., ShoppingApp). Third, because PetClinic is an open-source web app with a microservice version currently available, it enables an objective evaluation of the results of microservice identification.

B. Baseline Approaches

In this paper, we evaluate the effectiveness of our approach by comparing it to multiple baseline approaches. The approaches compared with our proposed technique are described below.

MEM: This technique introduced by Mazlami et al. [11], operates by taking a monolithic web app uploaded to a version control system, such as Git[11], as input. The resulting output is a graph representing classes as nodes and illustrating relationships between nodes as edges. In the graph, edges are generated based on the change history of classes obtained from the version control system, information on the contributors who modified the classes, or information on the source code similarity between classes that can be obtained by static analysis. Following the generation of this graph, Kruskal's algorithm [31] is employed to derive a Minimum Spanning Tree (MST) [32] for microservice identification. In the experiment described in this paper, edges were generated using information on source code similarity.

Bunch: Introduced by Mitchell et al. [33], this technique initiates by extracting call relationships between components at the source code level through static analysis of a web app. Subsequently, a Module Dependency Graph (MDG) is generated based on the results. This MDG represents components as nodes and represents the relationships between components as edges. Then, using the MDG as input, clustering is performed using the Hill-Climbing algorithm [34] to identify microservices.

Mono2Micro: Introduced by Kalia et al. [18], this technique initiates by extracting execution traces based on use-cases. These traces are analyzed to represent call relationships between classes in the form of a graph. Clustering is then executed using a hierarchical clustering algorithm [35] for microservice identification.

C. Evaluation Metrics

Our task in this paper is to cluster multiple components of a monolithic web app into two or more microservices. This problem can be regarded as a multiclass classification problem for classifying each component as one of multiple microservices.

To assess the quality of the identified microservices in web apps, it is crucial to establish the ground truth. The performance of identification algorithms is evaluated by comparing their results against the ground truth. The method used to generate the ground truth is detailed in the following section. In terms of evaluation metrics, this paper utilizes the accuracy metric [36], which is commonly used to evaluate the performance of multiclass classification algorithms.

11) https://git-scm.com/

Table 1. Subject applications

Name	The number of components: Class (LOC), View (Type), Table	Technologies used: Application framework, Presentation framework, Database technology	Number of use-cases	Number of scenarios	Description
JPetStore2[a)]	48(1602), 21(JSP)&2(HTML), 13	Spring[b)], Spring Web MVC[c)], iBatis[d)]	4	30	Shopping mall for pet animals
JPetStore6	24(1406), 20(JSP)&2(HTML), 13	Spring, Stripes, MyBatis	4	28	Shopping mall for pet animals
PetClinic[e)]	25(782), 12(HTML), 7	SpringBoot[f)], Thymeleaf[g)], JPA[h)]	3	21	Animal hospital management system
ShoppingApp[i)]	22(1353), 13(JSP), 4	N/A, N/A, JDBC	4	21	Fashion shopping mall
DayTrader[j)]	108(10376), 24(JSP)&19(HTML) &15(XHTML), 6	N/A, N/A, JPA	5	46	Stock trading system

[a)]https://github.com/KimJongSung/jPetStore
[b)]https://spring.io/projects/spring-framework
[c)]https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html
[d)]https://ibatis.apache.org
[e)]https://github.com/spring-projects/spring-petclinic
[f)]https://spring.io/projects/spring-boot
[g)]https://www.thymeleaf.org
[h)]https://docs.oracle.com/javaee/7/tutorial/partpersist.htm
[i)]https://github.com/manhduydl/Shopping-web-Jsp-Servlet
[j)]https://github.com/WASdev/sample.daytrader7

The ground truth in this study resulted from the identification of microservices based on their functions through the expertise of web app professionals, utilizing both static and dynamic analyses of monolithic web apps. The process of generating ground truth is as follows. Web app experts are divided into two groups. The first group identifies microservices. The second group verifies the identified microservices. These two groups repeat the above steps until they agree on the result of the microservice identification task.

In this paper, ground truth is represented as G: $C \rightarrow M$, where C is the set of components of a web app as defined in Section III.C. The set of microservices is defined as $M = \{m_1, m_2, ..., m_l\}$, where l indicates the number of microservices in a web app, and m_i denotes the ith microservice. The result from the application of a microservice identification technique is represented as R: $C \rightarrow M$. Based on the definitions above, the TP_i (True Positive) is defined below.

- TP_i: the number of components classified as m_i in R from among components classified as m_i in G

Finally, the accuracy [36] for the entire microservice app is defined as follows:

Accuracy: the ratio of correctly classified components in R according to G with respect to the total components

$$\text{Accuracy} = \frac{\Sigma_{i=1}^{l} TP_i}{|C|} \quad (1)$$

D. RQ1: Is Our Approach more Effective than Existing Approaches?

It is necessary to evaluate the effectiveness of the proposed approach through a comparison with existing approaches. Therefore, the evaluation was conducted as follows:

1) Evaluation Method

Microservice identification was conducted using four types of techniques: the proposed approach, MEM, Bunch, and Mono2Micro. Only the proposed approach considers tables and views; the other three do not. However, to compare the four approaches more appropriately, all the components unclassified by these algorithms were randomly allocated to microservices. Following this adjustment, the accuracy of the identification result from each method was measured to evaluate the quality of the identification method.

2) Results and Analysis

Table 2 presents the results. Below the name of each web app in this table, the total numbers of components, classes, views, and tables classified as microservices are listed. However, these numbers differ from those listed in Table 1. This difference arises due to the inability to categorize some components into specific microservices. These components have the following characteristics. These components are the basic components that each microservice should have. For example, the *JsonDecoder* class in DayTrader can not be classified as a specific microservice. This class is responsible for parsing JSON strings and constructing the corresponding objects in a web app. In this case, parsing is a fundamental function essential for each microservice. Consequently, components with this characteristic can not be classified as specific microservices.

The *Hit* and the *Accuracy* columns in Table 2 present four values each, representing perspectives on total components,

Table 2. Comparison of our approach with baseline approaches

Approach	Web app (Total number of components, Number of classes, Number of views, Number of tables)									
	JPetStore2 (73, 44, 16, 13)		JPetStore6 (53, 23, 17, 13)		PetClinic (32, 18, 7, 7)		ShoppingApp (37, 21, 12, 4)		DayTrader (131, 85, 40, 6)	
	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy
Mono2Micro	48, 38, 3, 7	0.66, **0.86**, 0.19, 0.54	30, 18, 8, 4	0.57, 0.78, 0.47, 0.31	15, 10, 4, 1	0.47, 0.56, 0.57, 0.14	21, 16, 4, 1	0.57, 0.76, 0.33, 0.25	58, 46, 10, 2	0.44, 0.54, 0.25, 0.33
Bunch	45, 37, 5, 3	0.62, 0.84, 0.31, 0.23	29, 19, 6, 4	0.55, 0.83, 0.35, 0.31	20, 14, 3, 3	0.62, 0.78, 0.43, 0.43	20, 14, 5, 1	0.54, 0.67, 0.42, 0.25	58, 45, 10, 3	0.44, 0.53, 0.25, 0.50
MEM	35, 25, 4, 6	0.48, 0.57, 0.25, 0.46	25, 13, 8, 4	0.47, 0.57, 0.47, 0.31	17, 11, 3, 3	0.53, 0.61, 0.43, 0.43	15, 11, 3, 1	0.41, 0.52, 0.25, 0.25	59, 45, 12, 2	0.45, 0.53, 0.30, 0.33
Proposed approach	59, 35, 13, 11	**0.81**, 0.80, **0.81, 0.85**	46, 20, 15, 11	**0.87, 0.87, 0.88, 0.85**	27, 16, 7, 4	**0.84, 0.89, 1.00, 0.57**	30, 17, 10, 3	**0.81, 0.81, 0.83, 0.75**	83, 50, 27, 6	**0.63, 0.59, 0.68, 1.00**

classes, views, and tables. For example, for JPetStore2, the Mono2Micro results can be interpreted as follows: First, the accuracy from the total component perspective ($Accuracy_{total}$) is calculated as 0.66, indicating that 48 out of 73 components are correctly classified based on the ground truth. Second, accuracy from the class component perspective ($Accuracy_{class}$) is 0.86, denoting that 38 out of 44 class components are correctly classified. Third, accuracy from the view component perspective ($Accuracy_{view}$) is 0.19, showing that 3 out of 16 view components are correctly classified. Finally, accuracy from the table component perspective ($Accuracy_{table}$) is 0.54, signifying that 7 out of 13 table components are correctly classified.

Because the existing approaches consider classes only, the performance of the proposed and existing approaches are first compared from the class component perspective. The $Accuracy_{class}$ of the proposed approach is 27% higher on average than that of the other approaches. Specifically, across all five web apps, the $Accuracy_{class}$ of the proposed approach consistently outperformed that of the MEM. Moreover, in four web apps (excluding JPetStore2), the $Accuracy_{class}$ of the proposed approach surpassed that of both the Bunch and the Mono2Micro. The reason the performance of the proposed approach is slightly lower than that of the Mono2Micro and the Bunch in JPetStore2 can be explained as follows. Web apps typically have domain objects that are used to store and manage the persistent data needed for specific domains. Consequently, the domain objects of a particular microservice are often used by other microservices requiring domain information. In other words, a domain object can be used more by microservices other than those creating and maintaining it. In such cases, the proposed approach has the limitation of misclassifying domain objects as services that use them more frequently. For example, the *Product* class in JPetStore2 is not accurately classified using the proposed approach. This class represents information on the animal breeds. Therefore, the *Product* objects should be created and maintained in the *Catalog Management* microservice which manages pet data in the web app. However, these objects are more frequently used in the *Cart Management* microservice than in the *Catalog Management* microservice. This occurs because the *Cart* object in the *Catalog Management* microservice often needs to access the data on pets that a user intends to purchase. Accordingly, the proposed approach classifies the *Product* class as a *Cart Management* microservice. This issue can be addressed in future works by refining the identification and classification of domain objects.

The performance of the proposed approach and existing approaches can be compared from the view component and the table component perspectives. The $Accuracy_{view}$ of the proposed approach is 41%-62% higher than that of the Mono2Micro which has a relatively good performance among the existing approaches. Furthermore, the $Accuracy_{table}$ of the proposed approach is 31%-67% higher than that of the Mono2Micro. As a result, it can be observed that the performance of the proposed approach from the view and table component perspectives is better than that of existing approaches.

For ShoppingApp, JSP is employed for views, and the business logic accessing tables is written in the JSP source code. These features pose a challenge for techniques that do not consider views, as they can not gather information on the business logic embedded in the JSP. However, our approach allows us to obtain better identification results by considering these views. Table 2 shows that the $Accuracy_{total}$ of the proposed approach is higher than that of the existing approaches by 24% to 40% in ShoppingApp.

For DayTrader, the $Accuracy_{class}$ and the $Accuracy_{view}$ of the proposed approach are low compared to those of the proposed approach for other web apps. The reasons for this are as follows: DayTrader contains use-cases and scenarios that are present in documents and codes but are not executed.

Consequently, some classes or views are not executed in

Table 3. Comparison with and without considering tables and views

Approach	Web app (Total number of components, Number of classes, Number of views, Number of tables)									
	JPetStore2 (73, 44, 16, 13)		JPetStore6 (53, 23, 17, 13)		PetClinic (32, 18, 7, 7)		ShoppingApp (37, 21, 12, 4)		DayTrader (131, 85, 40, 6)	
	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy
Without table & view	43, 35, 5, 3	0.59, **0.80**, 0.31, 0.23	29, 20, 5, 4	0.55, **0.87**, 0.29, 0.31	22, 16, 3, 3	0.69, **0.89**, 0.43, 0.43	23, 17, 5, 1	0.62, **0.81**, 0.42, 0.25	63, 50, 10, 3	0.48, **0.59**, 0.25, 0.50
Without view	49, 35, 3, 11	0.67, **0.80**, 0.19, **0.85**	35, 20, 4, 11	0.66, **0.87**, 0.24, **0.85**	21, 16, 1, 4	0.66, **0.89**, 0.14, **0.57**	24, 17, 4, 3	0.65, **0.81**, 0.33, **0.75**	66, 50, 10, 6	0.50, **0.59**, 0.25, **1.00**
Without table	53, 35, 13, 5	0.73, **0.80**, **0.81**, 0.38	40, 20, 15, 5	0.75, **0.87**, **0.88**, 0.38	25, 16, 7, 2	0.78, **0.89**, **1.00**, 0.29	28, 17, 10, 1	0.76, **0.81**, **0.83**, 0.25	79, 50, 27, 2	0.60, **0.59**, **0.68**, 0.33
Proposed approach	59, 35, 13, 11	**0.81, 0.80, 0.81, 0.85**	46, 20, 15, 11	**0.87, 0.87, 0.88, 0.85**	27, 16, 7, 4	**0.84, 0.89, 1.00, 0.57**	30, 17, 10, 3	**0.81, 0.81, 0.83, 0.75**	83, 50, 27, 6	**0.63, 0.59, 0.68, 1.00**

DayTrader, leading to the proposed approach being unable to classify these components accurately as microservices. If static analysis techniques such as the Bunch and the MEM are used, components related to unexecuted use-cases and scenarios can be identified. However, the Mono2Micro and our approach, which are both dynamic analysis techniques, can not identify the components mentioned above. Consequently, our approach exhibits a lower performance in the DayTrader experiment compared to experiments on other web apps. Nevertheless, in the DayTrader experiment, the $Accuracy_{total}$ obtained by the proposed approach is at least 18% higher than those of the baseline approaches, showing that our approach has better identification performance.

The performance described above, assessed in comparison with the ground truth identified by web app experts, indicates that the microservice apps identified using our approach align closely with those identified by experts from a functional perspective. In short, our identification technique can properly partition web apps by function, as intended by experts. The experimental results described above indicate that the proposed approach can be effectively used to identify microservices.

E. RQ2: Are Tables and Views Appropriately Allocated to Microservices?

It is crucial to assess whether tables and views are assigned appropriately to microservices when those microservices are identified using the proposed approach. To conduct this evaluation, experiments were conducted as described below.

1) Evaluation Method

The components of the five types of web apps were classified as microservices using the proposed approach. To determine whether the views and tables were appropriately considered using our technique, the following three groups were generated. One group was produced in which the classification results for tables and views were removed from the original classification results; another group was produced in which only the classification results for tables were removed; and a third group in which only classification results for views were removed. Subsequently, in these three groups, each component with removed classification result is randomly assigned to a microservice. An accuracy metric was employed to compare the original classification results with the classification results of the three groups.

2) Results and Analysis

The results are presented in Table 3. The configurations of the *Hit* column and the *Accuracy* columns in this table are the same as those listed in Table 2. It is noteworthy that the accuracies in Table 3 have the following characteristics. First, for the group without the classification results for views and tables, only views and tables are randomly classified as microservices. Therefore, the $Accuracy_{class}$ of this group is the same as that of the proposed approach. Second, for the group without the classification results for views, only the views are randomly classified. Thus, the $Accuracy_{class}$ and the $Accuracy_{table}$ of this group are the same as those of the proposed approach. Third, in the case of the group without the classification results for tables, only the tables are randomly classified. Therefore, the $Accuracy_{class}$ and the $Accuracy_{view}$ of this group are the same as those of the proposed approach. Overall, for all five web apps used in the experiment, the accuracies of the proposed approach are better than those of the three groups without the classification results for tables or views.

First, the performance of the proposed approach and the other three groups is compared from a total component perspective. The $Accuracy_{total}$ of the proposed approach is 3%-32% higher than that of the other three groups. For JPetStore6, the proposed approach shows significantly better performance compared to the group without the classification

results for tables and views, with a difference in the $Accuracy_{totoal}$ of 32%. However, for DayTrader, the proposed approach shows only slightly better performance compared to the group without the classification results for tables, with a difference in the $Accuracy_{total}$ of only 3%. As explained in the previous paragraph, the $Accuracy_{class}$ and $Accuracy_{view}$ of the group without the classification results for tables are the same as those of the proposed approach. Therefore, the difference in the $Accuracy_{total}$ between the proposed approach and the group without the classification results for tables depends on the proportion of tables among all components in the web app and the performance of the table classification. For DayTrader, the $Accuracy_{table}$ of the proposed approach is 100%, which is 67% higher than the $Accuracy_{table}$ of the group without the classification results for the tables. However, the percentage of tables in DayTrader is low (3%). In other words, even if all tables in DayTrader are accurately classified as microservices, the $Accuracy_{total}$ can only be improved by approximately 3%. Therefore, the difference in the $Accuracy_{total}$ between the proposed approach and the group without the classification results for the tables is not significant.

Second, the performance of the proposed approach and the other three groups might be compared from the class component perspective. However, this comparison is meaningless. As mentioned earlier, the $Accuracy_{class}$ of the proposed approach is the same as that of the other three groups for each subject's web apps.

Third, the performance of the proposed approach and the other three groups is compared from the view component perspective. The average percentage of views among all the components in the five web apps was 32%. A comparison between the group without the classification results for views and the proposed approach demonstrates superior results with the proposed approach. For JPetStore6, in which the percentage of views is high at 37%, the $Accuracy_{total}$ of the proposed approach is 21% higher than that of the group without the classification results for views. In addition, the $Accuracy_{view}$ of these two groups differs by 64%. However, for JPetStore2, in which the percentage of views is low at 27%, the $Accuracy_{total}$ of the proposed approach is 14% higher than that of the group without the classification results for views. In this case, the $Accuracy_{view}$ of these two groups differs by 62%. Overall, the $Accuracy_{view}$ of the proposed approach is 43-86% higher than that of the group without the classification results for view.

Finally, the performance of the proposed approach and the other three groups is compared from a table component perspective. The average percentage of tables among all the components in the five web apps is 11%. For JPetStore6, in which the percentage of tables is a large rate of 22%, when the group without the classification results for tables is compared with the proposed approach, the $Accuracy_{total}$ differs by up to 12% and the $Accuracy_{table}$ differs by up to 47%. However, for DayTrader, for which the percentage of tables is low at only 3%, a comparison between the proposed approach and the group without the classification results for tables shows that the $Accuracy_{total}$ of these two groups differs by only 3%. As mentioned earlier, because the proportion of tables in DayTrader is small, the $Accuracy_{total}$ is not significantly affected by the performance of table classification. However, the $Accuracy_{table}$ of the proposed approach is 66% higher than that of the group without the classification results for the tables. Overall, the $Accuracy_{table}$ of the proposed approach is 28%-67% higher than that of the group without the classification results for tables.

The experimental results described above demonstrate that the tables and views can be appropriately assigned to microservices using the proposed approach.

V. CONCLUSION

This paper introduces a novel approach for transforming monolithic web apps into microservices by utilizing use-cases as fundamental units. Each microservice identified using this method has a specific function. Additionally, in contrast to other methods for identifying microservices, this paper considers all three layers of web apps. As a result, our approach showed better performance than existing approaches.

In future work, we plan to verify the applicability of our approach to web apps developed using languages other than Java. In addition, we intend to further validate our approach by applying it to large-scale web apps.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2021R1F1A1048842). This study was supported by the Research Fund, 2021 of The Catholic University of Korea. This paper is a revised and expanded version of a Master's thesis [37] written by one of the authors.

REFERENCES

[1] H. M. Kienle and D. Distante, "Evolution of web systems," in *Evolving Software Systems*, Berlin, Germany: Springer, ch. 7, pp. 201-228, 2014. DOI: 10.1007/978-3-642-45398-4_7.

[2] M. d' Aquin, M. Sabou, and E. Motta, "Modularization: a key for the dynamic selection of relevant knowledge components," in *Proceeding of the Workshop on Modular Ontologies*, Athens: GA, pp. 1-14, 2006.

[3] S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert, "A survey of black-box modernization approaches for information

systems," in *Proceeding of the. International Conference on Software Maintenance*, San Jose: CA, pp. 173-183, 2000. DOI: 10.1109/ICSM.2000.883039.

[4] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, Cham, Switzerland: Springer, ch. 12, pp. 195-216, 2017. DOI: 10.1007/978-3-319-67425-4_12.

[5] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: intentions, strategies, and challenges," in *Proceeding of the International Conference on Software Maintenance and Evolution*, Cleveland: OH, pp. 481-490, 2019. DOI: 10.1109/ICSME.2019.00081.

[6] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116-116, 2015. DOI: 10.1109/MS.2015.11.

[7] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G. E. Boussaidi, G. Hecht, J. Privat, and Y. Guéhéneuc, "A taxonomy of service identification approaches for legacy software systems modernization," *Journal of Systems and Software*, vol. 173, no. 1 pp. 11086, 2021. DOI: 10.1016/j.jss.2020.110868.

[8] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," *IEEE Software*, vol. 35, no. 3, pp. 44-49, 2018. DOI: 10.1109/MS.2018.2141035.

[9] A. Levcovitz, R. Terra, and M. T. Valente, "Towards a technique for extracting microservices from monolithic enterprise systems," *arXiv:1605.03175*, 2016. DOI: 10.48550/arXiv.1605.03175.

[10] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting candidates of microservices from monolithic application code," in *Proceeding of the Asia Pacific Software Engineering Conferenc*e, Nara, Japan, pp. 571-580, 2018. DOI: 10.1109/APSEC.2018.00072.

[11] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *Proceeding of the International Conference on Web Service*, Honolulu: HI, pp. 524-531, 2017. DOI: 10.1109/ICWS.2017.61.

[12] N. Dmitry and S.S. Manfred, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24-27, 2014.

[13] D. Bajaj, A. Goel, and S. C. Gupta, "GreenMicro: identifying microservices from use cases in greenfield development," *IEEE Access*, vol. 10, pp. 67008-67018, 2022. DOI: 10.1109/ACCESS. 2022.3182495.

[14] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-oriented microservice extraction based on execution trace clustering," in *Proceeding of the International Conference on Web Services*, San Francisco: CA, pp. 211-218, 2018. DOI: 10.1109/ICWS.2018.00034.

[15] S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, "Identifying microservices using functional decomposition," in *Proceeding of the International Symposium on Dependable Software Engineering. Theories, Tools, and Applications*, Beijing, China, pp. 50-65, 2018. DOI: 10.1007/978-3-319-99933-3_4.

[16] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated microservice identification in legacy systems with functional and non-functional metrics," in *Proceeding of the International Conference on Software Architecture, Salvador*, Brazil, pp. 135-145, 2020. DOI: 10.1109/ICSA47634.2020.00021.

[17] L. Bao, C. Yin, W. He, J. Ge, and P. Chen, "Extracting reusable services from legacy object-oriented systems," in *Proceeding of the International Conference on Software Maintenance*, Timisoara, Romania, pp. 1-5, 2010. DOI: 10.1109/ICSM.2010.5609744.

[18] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2Micro: a practical and effective tool for decomposing monolithic Java applications to microservices," in *Proceeding of the ACM Joint Meeting of European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Athens, Greece, pp. 1214-1224, 2021. DOI: 10.1145/3468264.3473915.

[19] D. Taibi and K. Systä, "From monolithic systems to microservices: a decomposition framework based on process mining," in *Proceeding of the International Conference on Cloud Computing and Services Science*, Heraklion, Greece, pp. 153-164, 2019. DOI: 10.5220/ 0007755901530164.

[20] B. Kurniawan, *Java for the web with servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions*, Carmel, IN: Sams, 2002.

[21] C. D. Grosso, M. D. Penta, and I. G. R. de Guzman, "An approach for mining services in database oriented applications," in *Proceeding of the European Conference on Software Maintenance and Reengineering,* Amsterdam, Netherlands, pp. 287-296, 2007. DOI: 10.1109/CSMR.2007.11.

[22] S. H. Kim and J. Oh, "An effective reuse-based approach to automatic identification of microservices," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 27, no. 6, pp. 673-687, 2023. DOI: 10.6109/jkiice.2023.27.6.673.

[23] S. H. Kim, J. H. Shin, and J. Oh, "Utilizing web component structure for automatic microservices identification," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 27, no. 7, pp. 892-895, 2023. DOI: 10.6109/jkiice.2023.27.7.892.

[24] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of aspectJ," in *Proceeding of the European Conference on Object-Oriented Programming*, Budapest, Hungary, pp. 327-354, 2001. DOI: 10.1007/3-540-45337-7_18.

[25] B. Andrade, S. Santos, and A. R. Silva, "From monolith to microservices: static and dynamic analysis comparison," *arXiv:2204.11844*, 2022. DOI: 10.48550/arXiv.2204.11844.

[26] L. Meurice, C. Nagy, and A. Cleve, "Static analysis of dynamic database usage in java systems," in *Proceeding of the International Conference on Advanced Information System Engineering*, Ljubljana, Slovenia, pp. 491-506, 2016. DOI: 10.1007/978-3-319-39696-530.

[27] A. Cleve and J.-L. Hainaut, "Dynamic analysis of SQL statements for data-intensive applications reverse engineering," in *Proceeding of the Working Conference on Reverse Engineering*, Antwerp, Belgium, pp. 192-196, 2008. DOI: 10.1109/WCRE.2008.38.

[28] J. Oh, W. H. Ahn, and T. Kim, "Automatic extraction of dependencies between web components and database resources in java web applications," *Journal of Information and Communication Convergence Engineering*, vol. 17, no. 2, pp. 149-160, 2019.

[29] M. Stal, "Using architectural patterns and blueprints for service-oriented architecture," *IEEE Software*, vol. 23, no. 2, pp. 54-61, 2006. DOI: 10.1109/MS.2006.60.

[30] J. Oh, H. Ahn, and T. Kim, "Automatic extraction of component collaboration in java web applications by using servlet filters and wrappers," KIPS Transactions on Software and Data Engineering, vol. 6, no. 7, pp. 329-336, 2017.

[31] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48-50, 1956. DOI: 10.2307/ 2033241.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. Cambridge, MA: MIT Press, 2009.

[33] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the Bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193-208, 2006. DOI: 10.1109/TSE.2006.31.

[34] M, Melanie. *An introduction to genetic algorithms*. Cambridge, MA:

MIT press, 1998.
[35] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30-34, 1973. DOI: 10.1093/comjnl/16.1.30.
[36] M. Grandiniconsider, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv:2008.05756*, 2020. DOI: 10.48550/arXiv.2008.05756.
[37] D. Jung, "A use-case-based approach to transforming legacy web applications into microservices," M.S. dissertation, The Catholic University of Korea, Bucheon, Republic of Korea, 2023.

Si-Hyun Kim
received a B.S. degree in 2023 from the School of Computer Science and Information Engineering in the Catholic University of Korea. He is currently a M.S. student at the School of Computer Science and Information Engineering of the Catholic University of Korea. His current research interests include software engineering and web engineering.

Daeil Jung
received a B.S. degree in 2021 and a M.S. degree in 2023 from the School of Computer Science and Information Engineering in the Catholic University of Korea. His research interests include microservices, web engineering, and software engineering.

Norhayati Mohd Ali
received the PhD degree in computer science from the University of Auckland, New Zealand, in 2011. Previously, she received the bachelor's degree from the Universiti Teknologi Malaysia and the master's degree from the University of Southampton, United Kingdom. She is an associate professor and currently, the head of the Software Engineering and Information System Department, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. Her research interests include software engineering, model-driven engineering, design critics in software engineering, and software project management.

Abu Bakar Md Sultan
received the bachelor's degree in computer from Universiti Kebangsaan Malaysia, in 1993, and the master's degree in software engineering and the Ph.D. degree in artificial intelligence from Universiti Putra Malaysia (UPM).

Jaewon Oh
received his BSc, MSc, and PhD in Computer Science from Seoul National University, Korea in 1997, 1999, and 2004 respectively. He was a senior research engineer at Samsung Electronics Company from 2004 to 2007. Currently, he is a full professor at the School of Computer Science and Information Engineering of the Catholic University of Korea. His current interests include web engineering, software evolution, and software engineering.