Accredited Ranking SINTA 2
Decree of the Director General of Higher Education, Research, and Technology, No. 158/E/KPT/2021
Validity period from Volume 5 Number 2 of 2021 to Volume 10 Number 1 of 2026

Published online on: **http://jurnal.iaii.or.id**

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 7 No. 2 (2023) 263 - 270 ISSN Media Electronic: 2580-0760

Decomposing Monolithic to Microservices: Keyword Extraction and BFS Combination Method to Cluster Monolithic's Classes

Bintang Nuralamsyah[1], Siti Rochimah[2*]
[1,2]Informatics, Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember
[1]6025211031@mhs.its.ac.id, [2]siti@if.its.ac.id

Abstract

Microservices architecture is widely used because of the ease of maintaining its microservices, as opposed to encapsulating functionality in a monolithic, which may negatively impact the development process when the application continues to grow. The migration process from a monolithic architecture to microservices became necessary, but it often relies on the architect's intuition only, which may cost many resources. A method to assist developers in decomposing monolithic into microservices is proposed to address that problem. Unlike the existing methods that often rely on non-source code artifacts which may lead into inaccurate decomposition if the artifacts do not reflect the latest condition of the monolith, the proposed method relies on analyzing the application source code to produce a grouping recommendation for building microservices. By using specific keyword extraction followed by Breadth First Search traversal with certain rules, the proposed method decomposed the monolith's component into several cluster whose majority of cluster's members have uniform business domain. Based on the experiment, the proposed method got an 0.81 accuracy mean in grouping monolithic's components with similar business domain, higher than the existing decomposition method's score. Further research is recommended to be done to increase the availability of the proposed method.

Keywords: monolith decomposition; microservices; monolithic architecture; source code extraction; clustering

1. Introduction

Organizations must have robust business solutions that rely on technology in today's era. The need for technology-based solutions encourages software developers to design and build software products with various architectures to build effective and efficient software [1]. Monolithic Architecture is one type of software architecture that is widely used. Several well-known companies, such as Netflix, Amazon, and eBay, have implemented onolithic Architecture in their software. Monolithic Architecture encapsulates all functions into a single application unit. This architecture makes building applications easier to launch, test, and develop [2]. However, behind the advantages of monolithic architecture, some weaknesses make this architecture a double-edged sword. As it develops over time, the application will continue to grow both in terms of functionality and source code. Using monolithic architecture in these conditions will increase the burden on application developers. The characteristic of monolithic architecture, which encapsulates all things in one unit when applied in large-scale applications, will make the application very complex. Performing bug-fixing and feature additions to the software will become harder. The application becomes difficult to be maintained or developed [3]. Microservices architecture is becoming a choice of software architecture to overcome the weaknesses of monolithic architecture [2]. This architecture breaks up a monolith application into a collection of small services that interact with each other through a communication scheme between services. In addition to the convenience in the service maintenance process, microservices architecture also drives the adoption of DevOps and Cloud Computing in the development cycle [4], [5]. Microservices architecture is not a silver bullet[6]. It also occupies it's problems [7]. One of them is how to decompose applications with monolithic architecture into a microservices architecture. Utilizing microservices architecture on a small project is wasting resources [8]. Therefore, the average application projects are initially built using monolithic architecture. The need for decomposition exists when the application becomes quite large and hard to handle by a monolithic architecture. Migrating an application built with a monolithic architecture to a microservices architecture is not easy [9]. So far, the process of decomposition of applications with

Accepted: 15-01-2023 | Received in revised: 13-02-2023 | Published: 15-03-2023

monolithic architecture is carried out by relying on the software architect's intuition and experience [10]. Decomposing without a certain approach or process can cost many resources.

Some methods to decompose monolithic have been proposed before. Vresk et al. explained that a combination of verb-based and noun-based word in source code could be a criterion for decomposing an application [11]. Unfortunately, This method is a decomposition principle without implementation examples. Another attempt to decompose an application was done by Li et al. by proposing decomposing technique based on the dataflow model of the application. It transforms the dataflow diagram of the application into a modified dataflow, then splits it into some microservices candidates. In other words, it depends on the software dataflow artifact, which sometimes needs to be updated [2]. Mazlami et al. proposed another formal model to extract microservices from monolith [12]. It can decompose applications by analyzing their source code semantic similarity. It could perform better on an application with many business domains but has many similar source files. Another methodology that Mazlami proposed is based on contributors' commits histories. It splits the monolith application based on the contributor's commits pattern. The limitation of this method is the number of commits in VCS. Baresi et al. proposed a method to identify microservice using OpenAPI specification [13]. The method matched terms in the API specifications against a reference vocabulary. It relies much on well-defined interfaces with meaningful names. Selmadji et .al. proposed a decomposition method by combining the architect's suggestion and decomposition approach based on some quality function [14]. The decomposition quality depends on the software architect's comprehension of the software. application can be clustered by specific rules to create candidate recommendations for microservices.

In summary, many existing works rely on software architects' experience or software artifacts besides source codes. These dependencies lead to the possibility of using outdated software artifacts or inaccurate suggestions from the architect, which produce a less-accurate decomposition result. To address this problem, this paper proposed a decomposition method that relies only on software source codes to assist software decomposition.

The method analyzes the usage of some reserved keywords in the source code and uses them in the clustering process to generate microservices recommendations with minimal intervention from the architect to generate a microservices candidate recommendation. It comprises two main step and one optional step. The first and second step are extraction step and main clustering step, whether the optional step is extra clustering step.

The rest of this paper is structured as follows: Section 2 formally introduces the decomposition method, dataset, and evaluation method. Section 3 describes the result of the experiment. In section 4, the conclusion of this study is drawn and discussed potential future work and limitation.

2. Research Methods

2.1 Decomposition Method

The main idea of the proposed method is based on one of Unix's Philosophies which states that each service is responsible for performing a single business task. That statement also constructed the basis of Microservice architecture (MSA). In Domain Driven Design, a software development approach focusing on domain model development [15], "performing a single business task," is implemented by creating bounded contexts. Bounded contexts are boundaries created to separate different business concepts/business domains in software [16].

Components of software that reside in the same bounded context are assumed to have the same business domain between one and the others and are prohibited from having direct dependency or relation to components outside their bounded context. Components in software can be in various forms. In Object Oriented Programming, a component may be in the form of a class. Each class can have dependencies or relations to other classes, which is identified by usage of specific keywords on the source code. For example, in PHP, these keywords are “use” and “extends”. Different programming languages may have different keywords. The extracting step of the proposed method is done by analyzing information about the usage of the keywords that represented a relation or dependency between components. This kind of information is significant because components in the monolithic application can be clustered by specific rules to create candidate recommendations for microservices.

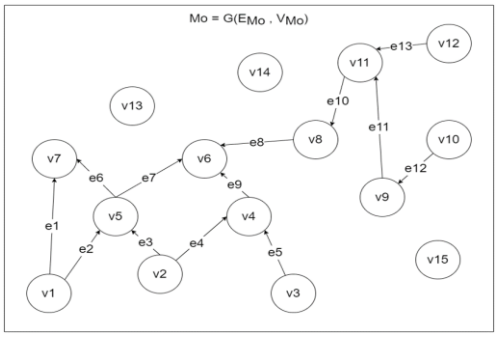

Figure 1. Example of Graph Representation of monolith Mo

A graph representing a monolith can be built using the component's relation as a base. The vertices represent

DOI: https://doi.org/10.29207/resti.v7i2.4866

component/class in the OOP paradigm, and the directed edges represent the existence of dependency from one component to the other. An edge from a vertex to the other vertices informs that a component is dependent on a particular component. For example, a sample monolith Mo with 15 components is represented in Figure 1. The source code for the sample can be accessed on the attached repository [17].

The graph was built by analyzing the relationship of every component that built the monolith Mo. On figure 1, some nodes do not have an arrow pointing to them. These nodes in this paper are called leaf nodes. On the contrary, nodes with no arrow coming out from them are defined as root nodes.

The clustering step of the proposed method is initialized by doing Bread First Traversal (BFS) on every leaf node with a stopping point last root node that has a path to the corresponding leaf node. For example, the BFS traversal on the leaf node v_2 visits these nodes consecutively: $v_2 \rightarrow v_5 \rightarrow v_4 \rightarrow v_7 \rightarrow v_6$

The method utilizes array to save list of nodes that were visited during each BFS as shown in Table 1. A graph with n-leaf nodes produces n-one-dimensional arrays because there are n different BFS traversal. These arrays are the seeds for microservices recommendation.

Table 1.The List of Arrays Produced by BFS

Leaf Node	Array
v_1	$[v_1, v_7, v_5]$
v_2	$[v_2, v_5, v_4, v_7, v_6]$
v_3	$[v_3, v_4, v_6]$
v_{10}	$[v_{10}, v_9, v_{11}, v_8, v_6]$
v_{12}	$[v_{12}, v_{11}, v_8, v_6]$
v_{13}	$[v_{13}]$
v_{14}	$[v_{14}]$
v_{15}	$[v_{15}]$

After all leaf nodes are traversed, the arrays are merged one with the other with specific rules. First, if they have n similar members, they are merged into one until no array has n similar members. This merging is based on a domain-driven design paradigm; a component should only depend on a component in the same bounded context.

Two arrays with n similar members are assumed to contain classes with similar business processes, so they should be grouped in the same microservices. The user determines the value of n. The default value equals one. one is selected as default, because in the exemplary implementation of domain-driven design, the component is only directly connected to the other component with the same business domain. Therefore, having one similar member is assumed to have a similar business process. Second, if there is no more array with n similar member, all arrays with one member only are merged.

The purpose of this process is to avoid the proposed method generating many microservice candidates with only one member, which is assumed may lead the developer to confusion. Third, an element of an array that exists in the other array must select one array randomly as its owner, so no element resides on two arrays. The arrays that have gone through the merging process are the microservices candidate recommendation for the developer.

Classes or components in the same array indicate they contain similar business processes and hence should be put in the same microservices. Therefore, the recommendation can help the developer know which monolithic component should be in the same microservice so the process of software comprehension, component mapping, and migration planning can be done faster as shown in Table 2.

Table 2. Final Result of Main Decomposing Process with $n = 2$

Cluster	Member
1	$[v_1, v_2, v_3, v_5, v_4, v_7, v_6]$
2	$[v_{10}, v_9, v_{11}, v_8, v_6, v_{12}]$
3	$[v_{13}, v_{14}, v_{15}]$

Table 3 are the pseudocodes of the extracting step. Line $2 - 6$ in `EXTRACT()` function defines the extraction process of each class file in the source code repository and determines what information is extracted from the class files.

Table 3. Pseudocode of Extraction Step

Pseudocode 1 Extraction Step	
1:	function EXTRACT(repository s)
2:	for file in s do
3:	class.classname ← GETCLASSNAME(file)
4:	class.references ← GETREFERENCES(file)
5:	classList.push(class)
6:	endfor
7:	Return classList
8:	end function

Table 4 shows the pseudocode of the main clustering process. `DECOMPOSE()` is a function that decomposes the monolith by using information from the extracting step.

Line 4 determines whether a class is a leaf node or not. If it is a leaf node, the algorithm starts the BFS traversal from the node and adds the visited nodes into array `cluster`. Lines 8-15 iterates over clusters and merged them if those clusters can be merged. Line 16 adds the special cluster into the list of the cluster. Function `canMerge()` takes argument n as the minimum threshold to determine whether two clusters can be merged. This algorithm returns a group of clusters that become the recommendation of the microservices.

There is a possibility that a class without any dependencies exists in the monolith, which means that the class will generate a one-member cluster and then merge into one big cluster. If the number of one-member clusters is too many, the particular cluster

DOI: https://doi.org/10.29207/resti.v7i2.4866

(combination of all clusters with one member) 's size will be enormous.

Table 4. Pseudocode of Main Clustering Step

```
Pseudocode 2 Primary Clustering Step
function DECOMPOSE(classList)
  clusterList = []
  for class in classList do
    if isLeaf(class) == true do
      clusterList.add(BFS(class))
  endfor
  specialCluster = []
  for cluster in clusterList do
    if canMerge(cluster, cluster+1, n) do
      MERGE(cluster, cluster+1)
      reset loop
    else if cluster.length == 1 do
      MERGE(specialCluster, cluster)
    endif
  endfor
  clusterList.push(specialCluster)
  return clusterList
end function
```

To reduce the size of the unique cluster, the proposed method is extracting another keyword, the namespace keyword. The namespace is a keyword that does not directly tell the dependencies of a class to another class. Nevertheless, this keyword can be used to help in grouping the one-member cluster. For each class that is a member of the unique cluster, the namespace is identified.

Table 5. Pseudocode of Extra Clustering Step

```
Pseudocode 3 Extra Clustering Step
function EXTRACLUSTER (clusterList)
  for class in clusterList["-1"] do
    namespace = class.namespace; specialclass = class
    for cluster in clusterList do
      for class in cluster do
        if namespace == class.namespace do
          cluster.push(specialclass)
          clusters[-1].remove(specialclass)
          found = True
        end if
      end for
    endfor
    if found == False do
      new_cluster =[specialclass]
      clusterList.push(new_cluster)
    end if
  end for
  clusterList.pop("-1")
  return clusterList
end function
```

Then, the method searchs in all clusters produced by the DECOMPOSE() function for a member with the same namespace as the current analyzed class. If there is a member with the same namespace, the class is then assigned to that cluster. However, if there is no member in all clusters with the same namespace, it then constructs its own cluster. In this paper, this kind of clustering is called the extra clustering step because it was done after the primary clustering step. This extra clustering step aims to improve the result of the decomposition method. The pseudocode is shown in Table 5.

2.2 Evaluation Method

A good form of monolith application decomposition is when each cluster generated by the process consists of classes that share the same business domain or are in the same bounded context. This argument is in line with the concept of microservice architecture, where each service on a microservice architecture is expected to have one specific responsibility in a bounded context [18]. In the grouping process carried out manually, each class on the monolith is analyzed. The analysis process is carried out by understanding the source code of the monolith application. From the results of the analysis, the business domain of each of these classes is determined. The process of clustering existing classes is carried out by grouping classes that have similar business domains. Through this process, a list of classes and their domains is generated. The list can then be used as ground truth to evaluate the decomposition results of the application by using the decomposition method. In this study, the evaluation was carried out by calculating the accuracy by matching the business domain resulting from manual analysis and the business domain resulting from the decomposition by automatic decomposition method. For the matching process to be carried out, each class in the decomposition result by the method must have a predicted domain. This method performs clustering, where clustering usually groups based on similarities between members without labels which in this context are the business domain of the class. Therefore, the authors use the *dominant domain* to help provide business domain predictions on each class in each cluster generated by decomposition by the method. The dominant domain is the business domain that has the most members in a cluster. For example, a clustering result from decomposition by the decomposition method, from now on referred to as cluster X, has five members, namely x_1, x_2, x_3, x_4, and x_5 shown in Table 6. By following ground truth, the actual business domain of all five-member sequentially transaction and authentication. Because business domain transaction owns one element more than authentication, the dominant domain of the current cluster is the transaction. So, all members of cluster X will be predicted as transaction business domains even though two of them have distinct business domains.

DOI: https://doi.org/10.29207/resti.v7i2.4866

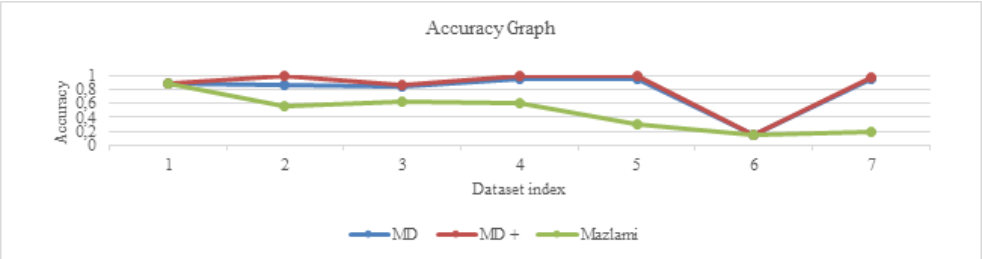

Figure 2. Accuracy Graph of The Proposed Method variants and Mazlami's Method

Table 6. Comparison of Actual and Predicted domain on Cluster X's member

Member	Actual Dom	Predicted Dom
x1	Authentication	Authentication
x2	Authentication	Authentication
x3	Authentication	Authentication
x4	Transaction	Authentication
x5	Transaction	Authentication

By using that dominant domain concept, the formula of accuracy that is used in this paper is formulated in Equation 1.

$$\text{Accuracy} = \frac{\sum_{i=0}^{j} f(c_i)}{n} \quad (1)$$

J is the number of clusters created, n is a number of classes in monolith application, ci is a a cluster created where $c_i \in C$, C is a set of clusters created during decomposition, and $f(c_i)$ is a function to calculate the number of dominant domain classes in a cluster which formulated by Equation 2.

$$f(c) = \max(nD_1, nD_2, \ldots, nD_k) \quad (2)$$

nD_x isa number of classes which have business domain D_x domain in cluster c, and k is a number of domains in monolith.

The greater the number of members with the same prediction domain as the actual domain, the higher the accuracy value. Clusters with all members having the same business domain will produce $\sum_{i=0}^{j} f(c_i)$ equals to n which lead to accuracy score equals to 1, the highest accuracy that can be achieved. Therefore, the more uniform the business domain of the cluster's members of each cluster produced, the greater the accuracy, which reflected better cluster quality. This evaluation compares the cluster quality generated by this paper method with the other method.

Seven datasets shown in Table 7 are used as ground truth sources in this experiment. Those seven datasets were web applications built by the PHP programming language and Laravel framework. Those datasets were chosen because, for the current method development, the author's focus is to decompose a PHP web application built with a monolithic architecture. Most of the datasets are taken from the Directorate of Technology and Information System Development of Institut Teknologi Sepuluh Nopember's git repository.

Table 7. Dataset Characteristic

No.	Dataset	LOC	Class	Domain's
1	Kayumas	2,990	34	3
2	Skill Passport	4,046	78	7
3	MyITS thesis	6,039	174	6
4	MyITS dorm	9731	197	5
5	MyITS Connect	35,194	322	8
6	MyITS	50,021	1038	16
7	MyITS	170,970	952	10

Class Number describes the number of classes that constructed the monolithic.

The domain's variance number is the number of the possible business domains on the corresponding dataset. For performance evaluation, the proposed methodology was compared to Mazlami Semantic Coupling Decomposition Algorithm [12]. Mazlami's algorithm was chosen because it uses the similar input as the proposed method and produces the same type of result (cluster of class). The experiments were done by decomposing those datasets using both algorithms. The evaluation is started with decomposing the dataset by the proposed method without the extra clustering step and n value equal to 1. After that, the proposed method decomposes the dataset with the extra clustering step and n value equal to 1. Finally, the dataset is decomposed by Mazlami's semantic coupling method. Because mazlami's method needs the number of clusters as an additional parameter, mazlami's method is run several times by varying the number of clusters. The minimum value for the number of clusters is the smallest total cluster created by the variance of the proposed method, which has been done before whereas the maximum value of the number of clusters is the highest total cluster created by the variance of the proposed method.

3. Results and Discussions

Figure 2 lists the evaluation result of each algorithm. For the easiness of writing, the proposed method without the extra clustering step is written MD, and the proposed method with an extra clustering step is written MD+. Figure 2 shows that the proposed method

DOI: https://doi.org/10.29207/resti.v7i2.4866

variances have better performances, in terms of accuracy, than the Mazlami's method. Either with or without additional clustering, the proposed method successfully delivers accuracy performance over Mazlami's on six of the seven available datasets. Mazlami's accuracy mean in this experiment is 0.47, with 0.25 as the standard deviation. Mazlami's method depends on the source code's semantic similarity score. Mazlami's method assumes that two components with similar syntax may have a similar business process. On the other side, most of the applications used as datasets come from an institute's repository.

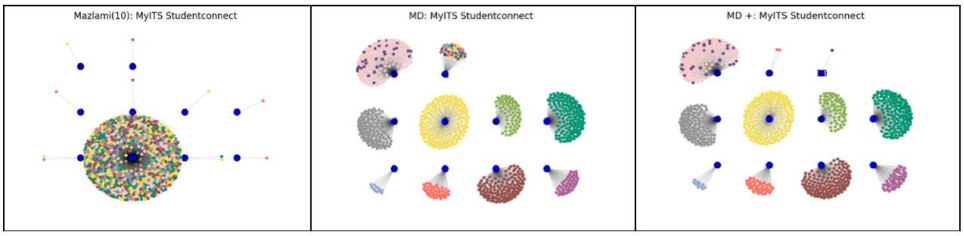

Figure 3. Decomposition Visualization on MyITS Studentconnect

They are built with some conventions the institute defines, such as variable and function naming rules and model and controller defining rules. These rules influenced the semantic similarity score, so two components written with the same convention will have similar semantics, even though they contain different business processes. Therefore, Mazlami's method is not suitable for this kind of application. T

he decomposition of Mazlami's method in this experiment creates one big cluster with various business domains. This kind of cluster is not preferable because the primary purpose of the decomposition is to split one enormous-size cluster called monolith into several smaller clusters called microservices. Each microservices is expected to have components with related business processes. The proposed method variants, MD and MD+, performed well on most datasets. The MD variant got a 0.79 accuracy mean with 0.26 as the standard deviation, while the MD+ variant accuracy was 0.84 with 0.28 as the standard deviation. 5 from 7 datasets were built by domain-driven design paradigm, which means the component source code writing follows the Domain Driven Design(DDD) rules. One of the DDD rules is that a component should not be dependent directly on a component with a different bounded context. Implementing those DDD rules on the source code caused most components to depend only on other components with similar business contexts, even though some components still violate these rules. Table 8 shows the number of components in each dataset which violated the DDD rules.

Table 8. Number of Violating Class in Dataset

Dataset	Violating Class
Kayumas	8
Skill Passport	0
MyITS thesis	2
MyITS dorm	0
MyITS Connect	0
MyITS Admission	20
MyITS Studentconnect	6

The Dataset with the highest number of violating classes is MyITS Admission. The combination of a high number of violating classes and a high number of total classes resulted in the creation of one enormous cluster that contains various domain businesses.

By analyzing the decomposition result[17], it is known that the giant cluster in MyITS Admission contained 884 components in the MD variant. Whereas the maximum number of components of a domain in MyITS Admission where only 122 components. It caused at least 762 components to be incorrectly predicted. It also happens in the MD+ variant, where 894 components got the wrong predicted domain. With this many wrong predicted components, it makes sense if MD and MD+ got low accuracy scores on MyITS Admission, even though they got 0.1 higher accuracy scores than Mazlami's method. MyITS StudentConnect is the largest Dataset in terms of LOC. The visualization of decomposition on this Dataset is shown in Figure 3. The blue dots represent the cluster's index, and the tiny colored dots represent classes or components. Small dots with the same color mean they have the same business domain. An edge from a small colored dot to a blue dot indicates that that class is a cluster member whose index was shown by the blue dot. The spatial placement does not represent anything; the main focus of the visualization is the relation between the blue dot with the small dots. From Figure 3 can be seen, Mazlami's method tends to build one big cluster, as described before. Whereas the MD and MD+ could split the components into several clusters, almost all clusters contain classes with the same business domain. One key difference in results between MD and MD+ is the existence of an extra clustering step. Without the clustering step, all arrays with one member component will be merged into one cluster. If the one-member arrays come from various domain businesses, the combined cluster will contain many different businesses process, as shown in the right-top cluster in the middle picture. Whereas only one domain was selected as the dominant domain; hence, classes with

DOI: https://doi.org/10.29207/resti.v7i2.4866

non-dominant domains in that cluster will be predicted wrong. The greater the number of classes predicted wrong, the lower the accuracy. This kind of situation did not happen in the MD+ variant because the MD+ variant mapped the classes without dependency into a cluster containing a component with the same namespace. In the author's habit of writing source code, classes with the same namespace usually contain similar business processes, even though this assumption still needs further investigation. Hence grouping the classes without dependency with this kind of approach improves the accuracy of the decomposition method in this context. However, further research may be needed, especially on namespace writing habits.

This experiment show that the MD and MD+ method are suitable for decomposing applications built with domain-driven design. With MD accuracy mean equals to 0.79 and MD+ accuracy mean equals to 0.84, it can be concluded that the proposed method got 0.81 in accuracy mean which means it decomposed the monolith well. Using a domain-driven design paradigm, the component is ruled to only be dependent on the component in the same bounded context. Therefore, the clusters of application with DDD generated by the proposed method tend to group components with the same business domain in one cluster. This kind of clustering is preferable because it helps developers understand which components of the application should be put in the same microservices later. For detailed report of the experiment please refer to the referenced repository [17].

4. Conclusion

This paper introduced a new approach to help developers decompose monolithic software into microservices. This approach analyzes information on reserved keyword usage that resides in the source code. Using source code as input, the proposed method in all variants could decompose the monolithic semi-automatically. By experiment, the proposed method got 0.81 on accuracy mean; compared to a similar existing method, the proposed method shows better results. The coupling of the component in the source code is the primary influencer of the decomposition result. The proposed method performs very well on source code built with the domain-driven design approach. The fact that this method was focused on PHP web applications with the Laravel framework built is one of the limitations of this work. This work focuses on analyzing keywords in the Laravel PHP framework, so currently, the scope of this method is only in applications built with the Laravel PHP framework.

Further research on other programming languages and other frameworks has the potential to expand the scope of the proposed method and is therefore considered advanced research. In addition, the additional clustering stages in this method result in more clusters, although their accuracy values are higher than the proposed method without additional clustering stages. Further research to assess the influence of the number of clusters and namespace writing habits on decomposition results can also be used as a further research direction. Another research direction is combining two or more existing methods with this proposed method. Various methods have been proposed before, even though they all have weaknesses. Combining those methods with this proposed method may increase the quality of software decomposition results.

Acknowledgment

This paper's experiment is supported by Directorate of Technology and Information System Development Institut Teknologi Sepuluh Nopember by providing Source code and developers to assist author in analyzing the dataset.

Reference

[1] F. Tapia, M. Á. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From Monolithic Systems to Microservices: A Comparative Study of Performance," *Appl. Sci.*, vol. 10, no. 17, Art. no. 17, Aug. 2020, doi: 10.3390/app10175797.

[2] S. Li *et al.*, "A dataflow-driven approach to identifying microservices from monolithic applications," *J. Syst. Softw.*, vol. 157, p. 110380, Nov. 2019, doi: 10.1016/j.jss.2019.07.008.

[3] K. Gos and W. Zabierowski, "The Comparison of Microservice and Monolithic Architecture," in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, Lviv, Ukraine, Apr. 2020, pp. 150–153. doi: 10.1109/MEMSTECH49584.2020.9109514.

[4] D. Trihinas, A. Tryfonos, M. D. Dikaiakos, and G. Pallis, "DevOps as a Service: Pushing the Boundaries of Microservice Adoption," *IEEE Internet Comput.*, vol. 22, no. 3, Art. no. 3, May 2018, doi: 10.1109/MIC.2018.032501519.

[5] A. Carrasco, B. van Bladel, and S. Demeyer, "Migrating towards microservices: migration and architecture smells," in *Proceedings of the 2nd International Workshop on Refactoring*, Montpellier France, Sep. 2018, pp. 1–6. doi: 10.1145/3242163.3242164.

[6] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, "Microservices: Migration of a Mission Critical System," *IEEE Trans. Serv. Comput.*, vol. 14, no. 5, Art. no. 5, Sep. 2021, doi: 10.1109/TSC.2018.2889087.

[7] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, "Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency," *IEEE Softw.*, vol. 35, no. 3, Art. no. 3, May 2018, doi: 10.1109/MS.2017.440134612.

[8] A. Singleton, "The Economics of Microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 16–20, Sep. 2016, doi: 10.1109/MCC.2016.109.

[9] V. Velepucha and P. Flores, "Monoliths to microservices - Migration Problems and Challenges: A SMS," in *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, Quito, Ecuador, Mar. 2021, pp. 135–142. doi: 10.1109/ICI2ST51859.2021.00027.

[10] S. Newman, *Monolith to microservices: evolutionary patterns to transform your monolith*, First edition. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, 2019.

DOI: https://doi.org/10.29207/resti.v7i2.4866

[11] T. Vresk and I. Cavrak, "Architecture of an interoperable IoT platform based on microservices," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, May 2016, pp. 1196–1201. doi: 10.1109/MIPRO.2016.7522321.

[12] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, Jun. 2017, pp. 524–531. doi: 10.1109/ICWS.2017.61.

[13] L. Baresi, M. Garriga, and A. De Renzis, "Microservices Identification Through Interface Analysis," in *Service-Oriented and Cloud Computing*, vol. 10465, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 19–33. doi: 10.1007/978-3-319-67262-5_2.

[14] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From Monolithic Architecture Style to Microservice one Based on a Semi-Automatic Approach," in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil, Mar. 2020, pp. 157–168. doi: 10.1109/ICSA47634.2020.00023.

[15] H. Vural and M. Koyuncu, "Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?," *IEEE Access*, vol. 9, pp. 32721–32733, 2021, doi: 10.1109/ACCESS.2021.3060895.

[16] P. Oukes, M. van Andel, E. Folmer, R. Bennett, and C. Lemmen, "Domain-Driven Design applied to land administration system development: Lessons from the Netherlands," *Land Use Policy*, vol. 104, p. 105379, May 2021, doi: 10.1016/j.landusepol.2021.105379.

[17] B. Nuralamsyah, "Monolith Mo." Accessed: Nov. 08, 2022. [Online]. Available: https://github.com/gslayer0/decomposing-monolith-to-microservices

[18] A. R. Sampaio, J. Rubin, I. Beschastnikh, and N. S. Rosa, "Improving microservice-based applications with runtime placement adaptation," *J. Internet Serv. Appl.*, vol. 10, no. 1, p. 4, Dec. 2019, doi: 10.1186/s13174-019-0104-0.

DOI: https://doi.org/10.29207/resti.v7i2.4866