Enhancement of Design Level Class Decomposition using Evaluation Process

Bayu Priyambadha[1], Tetsuro Katayama[2]
Interdisciplinary Graduate School of Agriculture and Engineering, University of Miyazaki, Miyazaki, Japan[1, 2]
Faculty of Computer Science, Universitas Brawijaya, Malang, Jawa Timur, Indonesia[1]

***Abstract*—Refactoring on the design level artifact such as the class diagram was already done using the threshold-based agglomerative hierarchical clustering method, specifically class decomposition. The approach produced a better cluster based on the label name similarity of attribute and method. But, some problems emerge from the experiment result. The negative Silhouettes element still exist in the cluster. And, there is an unusable cluster that only consists of one attribute element. This paper has proposed the evaluation process to optimize the result of clustering. This evaluation process is an additional process that aims to move the negative Silhouettes element to the other cluster. The movement is also to get the better value of element Silhouettes value. The evaluation process can produce a better result for clusters. The clusters produced from the evaluation process have higher Silhouettes values. The average Silhouettes value is increased by about 40%. Ultimately, the result shows no unusable cluster as mentioned in the previous research.**

Keywords—Refactoring; design level refactoring; software refactoring; class decomposition; software quality

I. Introduction

Refactoring the software design artifact is essential to maintain the design's internal structure [1]. Changes or alteration to the design artifact is easier than source code. The easiness is because of the original character of the model. The model is easy to change because the model is an abstract description of something more detailed. Refactoring at the design level means the refactoring activity is using the software design artifact as an object. The easiness of alteration and simulation of quality measurement using the specific metric is one of the benefits of refactoring software design artifacts. The software design artifact is a model bridging the requirement and implementation artifact and is the center of the software development process [2]. The changes will impact both sides, requirement and implementation artifact. In the case of software maintenance, refactoring activity is one way to decrease the maintenance cost [3], [4]. In the case of software development, the refactoring activity can be used as the evaluation process to maintain the internal quality of the design artifact before it is implemented into the source code. The design level refactoring increases the quality awareness of the design artifact as early as possible.

Shifting the object of refactoring activity to a higher level of abstraction has a specific problem. There is a limitation of information in the design artifact rather than the implementation artifact [5]. Therefore, excavation or mining and in-depth information analysis of the design artifacts are necessary [6]–[8]. Generally, the information on the design artifact is only written on the artifact. Sometimes, the information contains a hidden meaning that needs extra analysis to mean it. Natural language processing or semantic analysis is one approach that provides the functionality to gain the meaning of information [7]. Different from that, the source code level information clearly defines a specific element. The software developer can easily use it as data to analyze and measure quality, for example, the number of operand or operators in the source code to measure the complexity of the source code. The developer also can easily know the relation between attribute and method by reading the internal source code. They can figure it out by looking at the assigning value statement to the specific attribute.

Refactoring activity begins from the existence of the smell in the software artifacts. The smell is the indicator that there is something wrong in it. The quality of the artifacts decayed because of the smell. Finding the smell in the artifact is the first activity before refactoring itself. Researchers have already researched the smell detection process in the software artifacts. Smell detection mostly uses the source code as an object, known as code smell detection. Nowadays, the design of smell detection has started to emerge [9]–[11]. The terms and characteristics between code and design smells are different. The differentiation is based on the object, and the information lies in it. But, the previous research tried to use the code smell term and characteristics to find the smell in the design artifacts [8]. As a result, the Blob smell is detected using the class diagram information. Semantic analysis was used to determine the relation between class elements based on the name labels to enrich the class diagram information.

Blob smell is one of the lacks of internal structure quality indicators. It indicates the greedy process of the class. One class has a lot of process in it, whereas the other class nearby is only the data provider. The blob class monopolizes data processing provided by the nearby classes [8]. This phenomenon can happen due to software changes or the developer's lack of clean architecture theory. The clean architecture theory explains that the class must comply with the Single Responsibility Principle [12]. During the development or evolution cycle, the class has to have only one reason to change. The reason to change is related to the process or functionality of the specific class. If the class has more than one function or manipulates many operations, it will be the candidate that there is much reason to change it during the software cycle.

Furthermore, the blob class in the software system will increase the maintenance cost because it affects the class's

understandability [13]. The refactoring activity can resolve the problems of the Blob class. One of the refactoring processes to solve the blob class problem is class decomposition. Mostly, a source code became the field of the class decomposition process. Much research has been conducted using source code information on the class decomposition process. Shifting the decomposition process to the design level is interesting due to the possibility of decreasing the cost of the change, easiness of change simulation, and early quality awareness.

Class decomposition is the process of separating one class into many classes. The decomposition is based on the specific characteristics defined before the separation process. Many researchers proposed the class decomposition mechanism at the source code or design level. Most of the class decomposition mechanism approach uses the clustering process [13]–[19], in which the elements in the class are separate based on each element's closeness characteristic. This method aims to make the separation process result following the Single Responsibility Principle concept. The Threshold-based Agglomerative Hierarchical Clustering was tried to implement on both source code [19] and design level artifacts [5]. Each clustering process is based on the metrics generated from each source code and design artifacts. The design level class decomposition on the class diagram uses two metrics, syntactic (syn) and semantic (sem) [5]. Both metrics are calculated by considering syntactic and semantic closeness from the element's label name. Using the closeness of syntactic and semantic of the label name, the Threshold-based Agglomerative Hierarchical Clustering created a more compact cluster compared to the result of clustering on the source code level. The compactness of clusters was observed from the value of the Silhouettes value of every cluster. However, the decomposition results still show the elements with a negative Silhouette value. A negative Silhouette value indicates that an element's distance from the others in its cluster is large. The negative Silhouette elements are considered the worst in the relation with the concept of single responsibility principle. It is important to enhance the element placement mechanism of the negative element. Additionally, some clusters are considered unable to implement because, in case implemented as a class, it will instantiate objects that cannot interact with each other. A cluster with only one element, especially if the element includes a private modifier, is deemed worthless or useless. As a result, it is seen to be critical to incorporate the modifier aspect in the decomposition process.

The validity of the class decomposition's result is important. It is related to the class's applicability when implemented in the real case or source code. The existence of negative elements in the resulting cluster and a single private element in one cluster is a big problem for the applicability of the class. This condition requires in-depth attention, especially to validate the result of the decomposition process. The basic validation mechanism is to move the specific element from the origin cluster to the other cluster. The moving mechanism aims to put the specific element (negative element) to the other cluster to get a better Silhouette value. The other problem is the existence of the private single-member cluster. It also decreases the applicability of the class when it is implemented into the software's source code. In the previous approach, the clustering process is based on the two metrics syn and sem. The addition of other metrics is important to solve the unusable class.

This research is conducted to propose the validation mechanism to solve previous research's problems. The basic idea is to move the elements in the cluster that are not well-positioned. The new metric is proposed to increase every cluster's placement accuracy and compactness. All descriptions of the proposed algorithm of the validation mechanism and the experiment are organized as follows. Section II summarizes the state of the arts of the class decomposition approach. Then continue the description of the class usability and compactness of the class in the decomposition process in Section III. Section IV and V explain the proposed algorithm and the research scenarios. Section VI describes the result and discussion. Then the last is the conclusion and future work in Section VII.

II. Related Work

Many researchers published methods for class decomposition based on a specific type of smell. The research has two object studies, source code, and class diagram. The following section summarizes the history of research in the area of class decomposition.

A. Class Decomposition on Source Code

Bavota et al. presented a number of methods that could be used to decompose classes at the level of source code. Bavota's research history used the two-step decomposition techniques and MaxFlow-MinCut algorithms to extract classes [14]–[16]. The research involved considering both structural and semantic characteristics of the class. There are three metrics used: Structural Similarity between Methods (SSM), Call-based Interaction between Methods (CIM), and Conceptual Similarity between Methods (CSM). According to a study, transitive closure was calculated using metrics based on the values of distance between class elements. The other method uses the graph to represent the relatedness between elements and the weight to represent the closeness between elements. The transitive closure is able to split a Blob class into more than two classes, which is a significant improvement over the MaxFlow-MinCut approach. Furthermore, it can automatically determine how many classes should be extracted from a Blob.

A discussion of metric-based refactoring opportunities identification for object-oriented software systems is presented in an article by Isong Bassey et al. [20]. They conducted a thorough analysis of sixteen (16) primary studies in order to identify the state of the practice in ROI. The purpose of this article is to summarize all existing refactoring opportunities. The analysis was divided into three categories: structural, semantic, and structural and semantic. Using metrics to identify refactoring opportunities is the focus of this paper. Al Dallal's structural approach and Bavota's structural and semantic approaches previously published elsewhere are summarized in this paper.

According to Wang Ying et al., weighted clustering is automatically used to refactor software [13]. This article focuses on class-level refactoring. A network is considered to be a representation of the relationship of dependencies between

methods (as nodes). There are three matrices that illustrate the relationships between methods, (i) attribute sharing (Sharing Attribute Weight/ SAW), (ii) method invocation (Method Invocation Weight/ MIW), and (iii) functional coupling (Functional Coupling Weight/ FCW). A combination of three matrices as well as semantic similarity weights (SSW) is used to compute edge weights. Thus, the most advantageous cluster with the appropriate weight is selected. Wang compares his method with Bavota and Fokaefs. Wang's approach improves cohesion and coupling without affecting the code's behavior. Furthermore, it improves code's understandability, flexibility, reusability, and maintainability.

Mohamed Hamdi discusses the Agglomerative Hierarchical Clustering (AHC) method for class decomposition [19]. The decomposition occurs until classes have a single responsibility iteratively. One of the main challenges is terminating the decomposition process. They define the threshold concept for determining the endpoint during the decomposition process. There are six matrices: Internal Attribute Sharing (IAS), Internal Direct Call Dependency (IDC), Internal Indirect Call Dependency (IIC), Internal Method Sharing (IMS), and External Indirect Call Dependency (EIC). In this case, the weighted AHC results are more beneficial. This approach appeared to be a solution to the problem of the limited number of classes resulting from the decomposition process and the termination state.

B. Model-Driven Software Engineering

Model-Driven Software Engineering (MDSE) uses a software model as the primary artifact of software development [2]. Compared to the implementation artifact (source code), the software model is closer to the problem domain. The model transformation is the heart of the MDSE since the MDSE aims to generate the source code from the models. On the other hand, there is another approach to the development of software called Code-centric Development (CcD). A comparison study between MDSE and CcD has already been done for over a decade. From the review paper by Domingo et al., many researchers have been evaluating the benefit of the MDSE [21]. Some works said that MDSE decreases development time (up to 89%) relative to Code-centric Development (CcD). The other works suggest that the MDSE is suitable for academic exercise. Furthermore, the other works assert that MDSE is also suitable for inexperienced developers. Finally, Domingo et al., based on their review of MDSE, conclude that the MDSE is suitable for academic exercise and inexperienced developers [21].

C. Class Decomposition on Class Diagram

The class decomposition process is shifted to the design level artifact taking into account the ease of change and quality measurement. A similarity score is calculated between the class's elements (attributes and methods) used in the decomposition process based on the metrics that are derived from the information found in the class diagram. There are two approaches to determining the similarity rates between elements of a class: syntactic and semantic analysis. Thus, the two approaches evaluate the similarity of sentences based on their similarity in terms of syntax and meaning. Those metrics are syn and sem. The following formulas are defined for the metrics [5].

$$syn = \begin{cases} 1, & similar\ type > 0 \\ 0, & similar\ type \leq 0 \end{cases} \quad (1)$$

and,

$$sem = \frac{2.wi.|w_1 \cap w_2| + ws.(|s(w_1,w_2)| + |s(w_2,w_1)|)}{|w_1| + |w_2|} \quad (2)$$

where $s(w_1, w_2)$ or $s(w_2, w_1)$ is the number of words that have a synonym relationship between two labels, and $wi = 1$ and $ws = 0.75$ [22]. The closeness or similarity between class elements is calculated using the following formula.

$$Sim(e1, e2) = \frac{syn + sem}{2} \quad (3)$$

The class decomposition process uses the Threshold-based AHC that is used the similarity formula to calculate the closeness between class elements. Based on the previous decomposition result, the static and dynamic threshold AHC clusters are more compact than Hamdi's approach. The compactness of the clusters is measured using the Silhouette value. Based on the results, it is evident that there are certain advantages to be gained, but there are also some shortcomings as well. Decomposition results still show elements with negative Silhouette values. When the Silhouette value is negative, it indicates that the current element is far from the other elements in the cluster. In other words, negative Silhouette elements are considered to be the worst. Negative elements need to be improved in their movement mechanism. Moreover, some clusters are considered unable to implement because their objects may not be able to collaborate. It is considered useless to have a cluster with only one element, especially if the element has a private modifier. This is why the modifier aspect must be included in the decomposition process. Avoiding useless clusters is essential.

III. Proposed Approach

A. Scope of Study

The research focuses on our previous research results using the same dataset as the previous experiment. Two formulas will be proposed to solve the previous problems. Those formulas will be focused on overcoming the negative Silhouette and useless cluster [5] by combining Class Usability ($CUsability$) and Silhouettes value ($s(i)$). The combination of two metrics are used to evaluate the cluster after the clustering process. The whole evaluation process will be proposed as an evaluation algorithm. This study also uses classes that are not problematic to gain other insights in this study.

B. Problem Accomplishment

The previous research's result mentioned that there were two problems found. The first is the negative Silhouette element, and the second is the cluster that is predicted to be unusable. That is why the evaluation process must consider two aspects: the Silhouette value and the usability of the class

(that will be quantified in the form of a metric). The following formula calculates the Eval value ($Eval$).

$$Eval = a.s(i) + b.CUsability \quad (4)$$

$s(i)$ shows the Silhouette value, and $CUsability$shows the class usability value. a and b is the weight that describes the proportion of each value to the evaluation value. $CUsability$ has measured the usability of the class by considering the number of public methods that existed in the cluster. The class is usable if it at least consists of one public method. In other words, if the class has a public method, the class will be able to collaborate with the others (useful). The $CUsability$ is calculated using the following formula.

$$CUsability = \begin{cases} 0 & , mpub = 0 \\ 1 & , mpub \geq 1 \end{cases} \quad (5)$$

The $mpub$ is the number of public methods in the class candidate (in the cluster). The silhouette value is calculated using the following formula [23].

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \quad (6)$$

Where,

- $a(i)$ = the average dissimilarity of i to all other objects of A, then,
- $d(i, K)$ = the average dissimilarity of i to all object cluster K, when $K \in Cluster \ and \ K \neq A$,
- $b(i) = MIN(d(i, K)), K \neq A$.

The proposed evaluation algorithm has the main process of selecting the negative element and then moving it to the other cluster by comparing the Eval value before and after moving. The algorithm will be appended to the previous algorithm as the evaluation process.

C. Preliminary Experiment

Before the algorithm is confirmed, a preliminary experiment is conducted to ensure the performance of the evaluation process. The preliminary experiment uses one study case from the Landfill dataset, Class Transfer (Blob class from HSQLDB). The preliminary experiment is an early evaluation of the proposed algorithm (implementation of the $Eval$ formula) that uses a combination of weights a and b. In the case of Class Transfer, using a combination of weights with the value a bigger than b in the $Eval$ formula, the process was always run because the negative element always existed. As a result, the process of evaluation is never stopped. Based on this result, it was tried to print out the difference of $Eval$ value (before and after moving) every iteration and draw it into the line graph to show the trend. Fig. 1 shows the line graph of $Eval$ value differences in the case Class Transfer using weights $a = 0.9$ and $b = 0.1$. The trend shows that the values are shifted alternately in the mid to late iterations. It seems that the specific element was moved and moved back to the same cluster because the value of Silhouettes of that particular element is always negative in both clusters (origin and destination cluster).

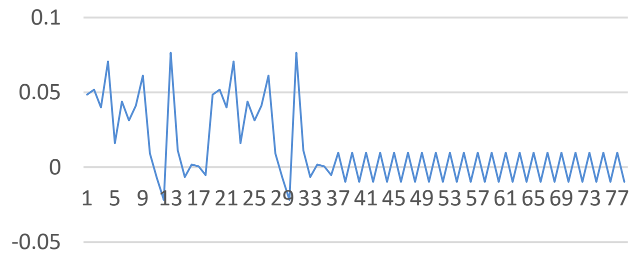

Fig. 1. Eval Value Differences of Class Transfer.

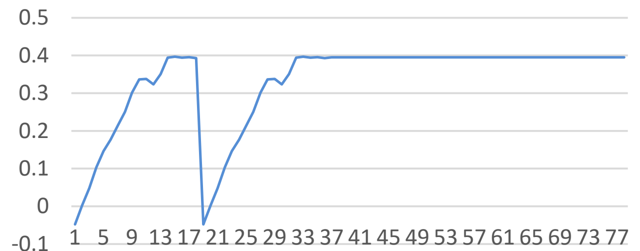

Fig. 2. The Average of Eval Value.

Fig. 1 shows the graph of the Eval value in every iteration. In the middle of the graph, Fig. 1, the data show a pattern that causes the unstoppable process. Even though it shows the pattern, the data seems unstable (continually moving from positive to negative). So, it needs to calculate to get a more stable value. Starting from iteration number 37, the Eval value between before and after moving is 0.00977 and -0.00977. Then, it tried to use the average formula to get a more stable value.

Fig. 2 shows the result after the values are averaged. The graph shows the flat value starting from iteration number 37, and it is easier to use as a termination condition for the algorithm for Class Transfer. The flat value of Class Transfer is about 0.4. The value of 0.4 cannot be used in the other study cases. It is only suitable for Class Transfer. Therefore, it is possible to be different from the other study cases. The other calculation is necessary to find a universal value to get the stopping condition.

D. Stopping Condition of Algorithm

The stopping condition in the decomposition process was the new problem that emerged in the preliminary experiment. One formula that can be used to find the pattern is by calculating the average Eval value between pre and post-movement to the other cluster. The following formula represents how the average of $Eval$ can be calculated.

$$AvgEval_n = \frac{Eval_n + Eval_{n-1}}{2} \quad (7)$$

Where,

- n is the number of decomposition iterations,
- $Eval_{n-1}$ is Eval value before moving to the other cluster,
- $Eval_n$ is Eval value after moving to the other cluster.

The $AvgEval$ for every iteration is represented in a line graph in Fig. 2. The easiest way to find the stopping condition is to calculate the differences of $AvgEval$ between two iterations using the following formula.

$$AvgDiff = AvgEval_n - AvgEval_{n-1} \quad (8)$$

Then the stop condition is represented as follows.

$$StopCondition = \begin{cases} AvgDiff = 0 & ,true \\ AvgDiff \neq 0 & ,false \end{cases} \quad (9)$$

Where,

- $AvgEval_n$ is Average Eval value from iteration number n,
- $AvgEval_{n-1}$ is Average Eval value from iteration number $n-1$.

The main idea of the stopping condition is to find zero (0) differences of $AvgEval$ between iterations. If the differences of $AvgEval$ are zero (0), then it means that there is no increment of Eval value even if the specific element is moved to the other cluster. Then the last position of the cluster will be chosen as the best solution. Fig. 3 shows the line graph of $AvgDiff$ as the representation of the differences of $AvgEval$ before and after movement.

E. Proposed Algorithm

The algorithm is proposed to answer the problems that emphasize the previous class decomposition approach. The new algorithm is the representation of an additional process on the class decomposition. The evaluation algorithm is described in Fig. 4. In the design level class decomposition research, the decomposition process is done by the Threshold-based Hierarchical Agglomerative Clustering. First, two metrics are used to calculate the closeness between elements in the class decomposition process, syn, and sem. Then, the process continues to the evaluation process. That is aim is to evaluate the placement of every element. This process is focused on the element that has the negative silhouette value. The evaluation process's main idea is to move the negative element to the other cluster to get a better silhouette value. The evaluation process is an iteration process that considers the increment of silhouette value and stopping condition that is defined in the previous section.

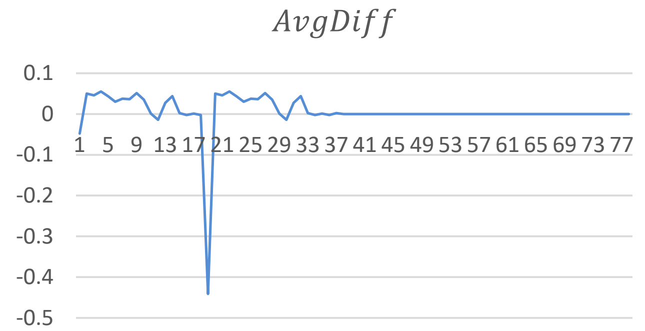

Fig. 3. The Line Graph $AvgDiff$

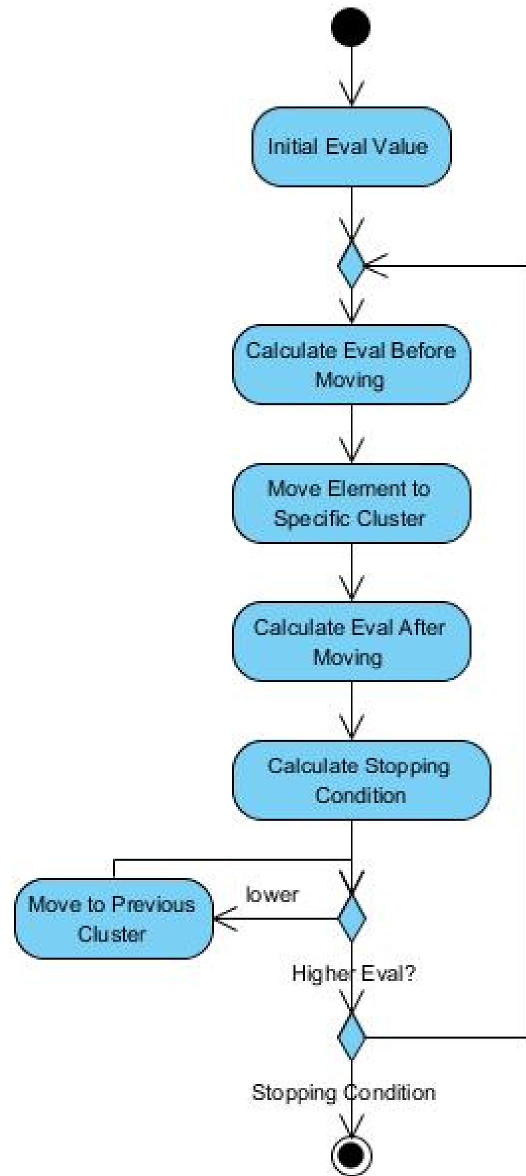

Fig. 4. The Evaluation Algorithm

IV. The Experiment Scenario

The class decomposition experiment uses ten study cases taken from the open-source Java application. The test cases are classes in several open-source Java application that is indicated as Blob smell according to the Landfill smell dataset [24]. Table I shows the list of test cases. All test cases will be decomposed using the threshold-based hierarchical agglomerative clustering and static and dynamic thresholds. Then evaluate using the proposed approach in various combinations of weights. The weights are set to start from a=0.1 and b=0.9 until a=0.9 and b=0.1, with the increment and decrement of 0.1.

The combination of weights has aimed to find the best composition of weights in the class decomposition process based on the compactness and usability of clusters.

TABLE I. List of Test Cases

No.	Class Name	Application
1.	AudioFile	aTunes
2.	JDBCBench	HSQLDB
3.	Interpreter	jEdit
4.	SVGOutputFormat	jHotDraw
5.	Transfer	HSQLDB
6.	Import	agroUML
7.	StringConverter	HSQLDB
8.	RipCdDialog	aTunes
9.	DefaultDrawingViewTransferHandle	jHotDraw
10.	MDIApplication	jHotDraw

TABLE II. The Result of Static Threshold Decomposition

Class	Threshold	a=0.1;b=0.9	a=0.2;b=0.8	a=0.3;b=0.7	a=0.4;b=0.6	a=0.5;b=0.5	a=0.6;b=0.4	a=0.7;b=0.3	a=0.8;b=0.2	a=0.9;b=0.1
AudioFile	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.932	0.864	0.796	0.728	0.661	0.593	0.525	0.457	0.389
	Silhouettes	0.322	0.322	0.322	0.322	0.322	0.322	0.322	0.322	0.322
JDBCBench	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.935	0.871	0.807	0.743	0.679	0.615	0.551	0.487	0.423
	Silhouettes	0.359	0.359	0.359	0.359	0.359	0.359	0.359	0.359	0.359
Interpreter	**Clusters**	1	1	1	1	1	1	2	2	2
	Eval	0.928	0.856	0.784	0.712	0.64	0.569	0.436	0.372	0.309
	Silhouettes	0.281	0.281	0.281	0.281	0.281	0.281	0.246	0.246	0.246
SVGOutputFormat	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.919	0.839	0.759	0.679	0.599	0.519	0.439	0.359	0.279
	Silhouettes	0.199	0.199	0.199	0.199	0.199	0.199	0.199	0.199	0.199
Transfer	**Clusters**	1	1	1	1	1	2	2	2	2
	Eval	0.929	0.859	0.788	0.718	0.648	0.418	0.266	0.225	0.258
	Silhouettes	0.296	0.296	0.296	0.296	0.296	0.216	0.179	0.228	0.291
Import	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.917	0.835	0.753	0.671	0.588	0.506	0.424	0.342	0.26
	Silhouettes	0.177	0.177	0.177	0.177	0.177	0.177	0.177	0.177	0.177
StringConverter	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.952	0.904	0.856	0.808	0.76	0.712	0.664	0.616	0.569
	Silhouettes	0.521	0.521	0.521	0.521	0.521	0.521	0.521	0.521	0.521
RipCdDialog	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.876	0.807	0.739	0.67	0.602	0.534	0.443	0.388	0.292
	Silhouettes	0.26	0.26	0.26	0.26	0.26	0.26	0.276	0.276	0.238
DefaultDrawingView TransferHandle	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.93	0.86	0.791	0.721	0.652	0.582	0.513	0.443	0.374
	Silhouettes	0.304	0.304	0.304	0.304	0.304	0.304	0.304	0.304	0.304
MDIApplication	**Clusters**	2	2	2	2	2	2	2	2	2
	Eval	0.927	0.854	0.782	0.709	0.637	0.564	0.492	0.419	0.347
	Silhouettes	0.274	0.274	0.274	0.274	0.274	0.274	0.274	0.274	0.274

TABLE III. The Result of Dynamic Threshold Decomposition

Class	Threshold	a=0.1;b=0.9	a=0.2;b=0.8	a=0.3;b=0.7	a=0.4;b=0.6	a=0.5;b=0.5	a=0.6;b=0.4	a=0.7;b=0.3	a=0.8;b=0.2	a=0.9;b=0.1
AudioFile	**Clusters**	7	7	7	7	7	7	7	9	7
	Eval	0.951	0.903	0.855	0.807	0.759	0.711	0.663	0.404	0.567
	Silhouettes	0.519	0.519	0.519	0.519	0.519	0.519	0.519	0.28	0.519
JDBCBench	**Clusters**	2	2	2	2	2	6	6	6	7
	Eval	0.935	0.871	0.807	0.742	0.678	0.361	0.266	0.239	0.266
	Silhouettes	0.357	0.357	0.357	0.357	0.357	0.016	0.185	0.23	0.292
Interpreter	**Clusters**	1	1	1	1	6	5	11	9	6
	Eval	0.928	0.856	0.758	0.712	0.413	0.402	0.372	0.352	0.293
	Silhouettes	0.281	0.281	0.281	0.281	-0.013	0.138	0.154	0.31	0.268
SVGOutputFormat	**Clusters**	5	5	5	5	10	9	9	8	6
	Eval	0.936	0.873	0.81	0.746	0.46	0.418	0.345	0.4	0.44
	Silhouettes	0.366	0.366	0.366	0.366	0.029	0.104	0.158	0.305	0.378
Transfer	**Clusters**	2	2	1	1	7	12	14	12	14
	Eval	0.902	0.829	0.788	0.718	0.461	0.259	0.246	0.288	0.331

	Silhouettes	0.249	0.249	0.296	0.296	-9.682	0.119	0.282	0.382	0.377
Import	**Clusters**	2	2	2	2	3	5	6	7	7
	Eval	0.862	0.843	0.765	0.687	0.472	0.368	0.392	0.376	0.39
	Silhouettes	0.227	0.219	0.219	0.219	0.078	0.214	0.474	0.403	0.403
StringConverter	**Clusters**	3	3	3	3	3	3	3	3	3
	Eval	0.942	0.885	0.827	0.77	0.712	0.655	0.597	0.54	0.482
	Silhouettes	0.425	0.425	0.425	0.425	0.425	0.425	0.425	0.425	0.425
RipCdDialog	**Clusters**	3	3	3	3	3	4	5	5	7
	Eval	0.927	0.855	0.783	0.711	0.638	0.495	0.465	0.418	0.409
	Silhouettes	0.277	0.277	0.277	0.277	0.277	0.307	0.307	0.314	0.362
DefaultDrawingView TransferHandle	**Clusters**	3	3	3	3	3	3	3	3	3
	Eval	0.922	0.845	0.768	0.69	0.613	0.536	0.458	0.381	0.304
	Silhouettes	0.227	0.227	0.227	0.227	0.227	0.227	0.227	0.227	0.227
MDIApplication	**Clusters**	5	5	5	5	5	5	5	6	5
	Eval	0.923	0.846	0.77	0.693	0.616	0.54	0.463	0.375	0.31
	Silhouettes	0.233	0.233	0.233	0.233	0.233	0.233	0.233	0.239	0.233

V. Result and Discussion

A. Result of the Experiment

The proposed algorithm was implemented in the prototype applications. Ten study cases of the Blob class are ready to use to ensure the new approach's final result. All of the classes were decomposed using a prototype application that was already updated using a new algorithm. Every result using the static and dynamic threshold decomposition is described in the following tables (Table II for the static and Table III for the dynamic threshold).

B. Compared to the Previous Approach

In the previous paper, two study cases were used, one of which is MDIApplication class. The result of decomposition using the previous algorithm on MDIApplication (using $a = 0.5$ and $b = 0.5$) is as described in Table IV and V.

In the case of the Silhouette value, using a new approach (after adding the validation process using the $Eval$ value), the result is shown as case number 10 (Tables II and III). There are increments of Silhouettes value of both static and dynamic threshold decomposition. The static threshold increased from 0.08 to 0.274, and the dynamic threshold increased from 0.15 to 0.233. Even though the dynamic threshold has a lower Silhouettes value, the dynamic threshold produces more clusters that match the purpose of single responsibility principles. More clusters are produced using a dynamic threshold.

The other result compared to the previous approach is the useless class. The problem emerged according to the cluster that only has one element (Table V), and the element is private (cluster number 2). The element name is scrollPane which has -0.27 of Silhouette. After updating the algorithm using the evaluation process ($Eval$ value), the result of decomposition is shown as follows (Table VI). The scrollPane element is moved to cluster number five, together with the other element. No clusters are considered unusable after updating the algorithm using the evaluation process.

C. Discussion

The previous section shows the experiment result after updating the algorithm using the evaluation process. Tables II and III show the detail of every combination of weight and express every case based on the cluster, $Eval$ value, and Silhouettes value. The result is different from one case to the other. For example, six cases using the dynamic threshold AHC produced a better value of Silhouettes than the static threshold AHC. Those cases are AudioFile, Interpreter, SVGOutputFormat, Import, and RipCDDialog. The rest of the cases are better using the static threshold AHC. The Silhouettes value is used as the consideration because the cluster requirement results in the high compactness of elements based on the similarity of syntax and semantics.

Higher Silhouettes also show the similarity of the cluster's context to produce the single responsibility class. With the use of the evaluation process, the result of Silhouettes can be increased by at least 40% of Silhouettes. This result shows that the evaluation process can place the elements more precisely by considering the value of the class usability and the Silhouettes. Most of the results show that the combination of weight (a and b) that can produce the best cluster is a higher portion than b.

TABLE IV. The Static Decomposition (Previous Approach)

Cluster	Elements	Silhouettes Index
1	parentFrame	-0.12
	MDIApplication	-0.03
	desktopPane	0.01
	Show	-0.41
	isSharingToolsAmongViews	-0.01
	Hide	-0.39
	serialVersionUID	-0.03
	scrollPane	0.03
	Prefs	0.00
2	createFileMenu	0.30
	Init	0.01
	getComponent	0.06
	createViewActionMap	0.31
	Configure	0.05
	createModelActionMap	0.15
	toolBarActions	-0.01
	createViewMenu	0.30
	updateViewTitle	0.32
	createHelpMenu	0.34
	createWindowMenu	0.30
	initLookAndFeel	0.11
	wrapDesktopPane	0.04
	createMenuBar	0.21
	createEditMenu	0.32
	Launch	0.05
Average Silhouettes		**0.08**

TABLE V. THE DYNAMIC DECOMPOSITION (PREVIOUS APPROACH)

Cluster	Elements	Silhouettes Index
1	isSharingToolsAmongViews Prefs	-0.12 -0.08
2	scrollPane	-0.27
3	parentFrame desktopPane	0.00 0.03
4	MDIApplication serialVersionUID	0.27 0.22
5	Show Hide	-0.19 -0.12
6	getComponent	-0.51
7	Launch	-0.62
8	createFileMenu Init initLookAndFeel createMenuBar	-0.84 -0.39 -0.49 -0.58
9	updateViewTitle Configure	0.28 0.08
10	createViewMenu createHelpMenu createWindowMenu createEditMenu	0.78 0.89 0.75 0.73
11	wrapDesktopPane toolBarActions	0.95 0.98
12	createViewActionMap createModelActionMap	1.00 1.00
Average Silhouettes		**0.15**

TABLE VI. RESULT OF DECOMPOSITION USING EVALUATION PROCESS ($a = 0.5; b = 0.5$)

Cluster	Elements	Modifier	Silhouettes
1	MDIApplication serialVersionUID isSharingToolsAmongViews	publicMethod private publicMethod	0.083 0.128 0.066
2	init initLookAndFeel	publicMethod protectedMethod	0.31 0.149
3	updateViewTitle configure launch show hide	protectedMethod publicMethod publicMethod publicMethod publicMethod	0.073 0.203 0.128 0.087 0.053
4	createViewMenu createHelpMenu createWindowMenu createEditMenu createFileMenu createMenuBar createViewActionMap createModelActionMap parentFrame	publicMethod publicMethod publicMethod publicMethod publicMethod protectedMethod protectedMethod protectedMethod private	0.532 0.59 0.527 0.483 0.497 0.326 0.572 0.282 0.03
5	wrapDesktopPane toolBarActions getComponent desktopPane scrollPane prefs	protectedMethod private publicMethod private private private	0.331 0.159 0.059 0.091 0.035 0.023
Average Silhouettes			**0.233**

TABLE VII. ELEMENT'S CHARACTER OF STUDY CASES

Class	A	B	C	D
AudioFile	39	9	30	Dynamic
JDBCBench	33	21	12	Static
Interpreter	65	20	45	Dynamic
SVGOutputFormat	61	9	52	Dynamic
Transfer	80	50	30	Dynamic
Import	30	13	17	Dynamic
StringConverter	16	1	15	Static
RipCdDialog	36	15	21	Dynamic
DefaultDrawingViewTransferHandle	15	2	13	Static
MDIApplication	25	6	19	Static

A: Total element; B: Attribute Element; C: Method Element; D: Approach

TABLE VIII. CORRELATION RESULT

No.	Pair Data	*p*-value
1.	Element - Approach	0.0134
2.	Attribute Element - Approach	0.1645
3.	Method Element - Approach	0.0247

In the specific number, $a \geq 0.7$ is suitable to produce a better cluster in both static and dynamic threshold AHC. Six cases are good using a dynamic threshold, and four cases using a static threshold. This result raises curiosity about whether the class decomposition uses static or dynamic.

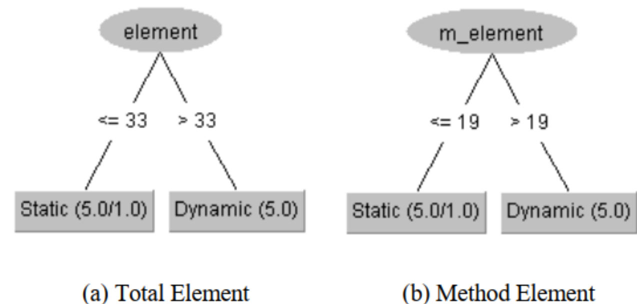

Fig. 5. The Tree Visualization of Tree-Based Classification Analysis

Based on the existing study cases in this research, every studied case is detailed into specific characteristics of class that relate to the element. For example, the number of total elements, the number of method elements, and the number of attribute elements are counted to find a correlation with the type of approach. Table VII shows the detail of the class based on the element.

The correlation between each character to the approach that is used on the class decomposition is counted using a statistical approach. There are three data pairs; the result is shown in Table VIII. Two pairs of data have significant differences. It is determined based on the result of the p-value of each pair. The total element (No. 1) and method element (No. 3) has a p-value lower than 0.05, and the attribute element (No. 2) is higher than 0.05. The total element and the number of method elements are related to the type of approach used in the class decomposition process.

The threshold number that can be used as the decision point for each total element and method element is also interesting. The data of element characteristics were analyzed using the tree-based classification method (Fig. 5). Furthermore, the total element and method element can indicate when the static or dynamic threshold should be used in the class decomposition process. The tree visualization shows the threshold number for each characteristic. The total element and method element have the threshold numbers 33 and 19. If the number of each characteristic is lower than the number, then the static threshold AHC is better and vice versa. This classification analysis result has an accuracy of about 80%.

The statistical and the threshold number analysis is only suitable for the current scope of the experiment. It needs more study cases to make the result acceptable to the larger scope of the experiment.

VI. Conclusion and Future Work

The class decomposition in the level of design is worth doing to support the concept of model-driven software engineering. The optimization of the design level Threshold-based Agglomerative Hierarchical Clustering (AHC) experiment has been done by adding an evaluation process. The evaluation process aims to move the specific element with negative Silhouettes value in every cluster to the other better cluster. The evaluation process is able to increase the average Silhouettes of the cluster compared to the previous approach. The increment of Silhouettes has averaged about up to 40%. The evaluation process is also able to solve the unusable cluster, as mentioned in the previous approach result.

This research experiment takes ten study cases from the Landfill smell dataset. All data consists of Blob smell. Most of the good result of decomposition is using the Dynamic Threshold AHC. Six study cases are good using the dynamic threshold, and four study cases are good using the static threshold. And the best result is produced by using the higher portion of Silhouettes (a) in the Eval formula.

The results were analyzed using a statistical approach to get more valuable information about the result. Three variables are related to the class: the number of total elements, method, and attribute elements. The total element and method element affect the use of the approach (static or dynamic threshold) to get a better result of decomposition. In the scope of the experiment, both the total element and method element have the threshold numbers 33 and 19. A smaller number than the threshold will use static, and a larger will use dynamic threshold.

The design-level class decomposition research is important to be continued in the future. The future plan aims to increase the optimization of the result of decomposition. In this experiment, the evaluation process is a separate process that is in sequence with the previous approach. Merging the algorithms became a consideration for future computational improvements. The increment of study cases number is worth increasing the algorithm's usability. Implementing the decomposition process at the source code level is worth doing in the future. The impact of changes in real implementation is important to study.

References

[1] M. Fowler et al., "Refactoring Improving the Design of Existing Code Second Edition," Second Ed. United State of America: Pearson Education - Wesley, 2019.

[2] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice." Morgan & Claypool, 2012.

[3] A. Yamashita and L. Moonen, "Exploring the impact of inter-smell relations on software maintainability: An empirical study," in Proceedings - International Conference on Software Engineering, 2013.

[4] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," Empir. Softw. Eng., vol. 23, no. 3, pp. 1188–1221, Jun. 2018.

[5] B. Priyambadha and T. Katayama, "Design Level Class Decomposition using the Threshold-based Hierarchical Agglomerative Clustering," Int. J. Adv. Comput. Sci. Appl., vol. 13, no. 3, pp. 57–64, 2022.

[6] B. Priyambadha and T. Katayama, "Tree-based keyword search algorithm over the visual paradigm's class diagram XML to abstracting class information," 2020 IEEE 9th Glob. Conf. Consum. Electron. GCCE 2020, pp. 280–284, 2020.

[7] B. Priyambadha, T. Katayama, Y. Kita, H. Yamaba, K. Aburada, and N. Okazaki, "Utilizing the similarity meaning of label in class cohesion calculation," J. Robot. Netw. Artif. Life, vol. 7, no. 4, pp. 270–274, 2021.

[8] B. Priyambadha, T. Katayama, Y. Kita, H. Yamaba, K. Aburada, and N. Okazaki, "The Seven Information Features of Class for Blob and Feature Envy Smell Detection in a Class Diagram," 2021 Int. Conf. Artif. Life Robot., pp. 348–351, 2021.

[9] K. Alkharabsheh, Y. Crespo, E. Manso, and J. A. Taboada, "Software Design Smell Detection: a systematic mapping study," Softw. Qual. J., vol. 27, no. 3, pp. 1069–1148, 2019.

[10] B. K. Sidhu, K. Singh, and N. Sharma, "A Catalogue of Model Smells and Refactoring Operations for Object-Oriented Software," Proc. Int. Conf. Inven. Commun. Comput. Technol. ICICCT 2018, pp. 313–319, 2018.

[11] B. Kaur Sidhu, "Model Smells In Uml Class Diagrams," Int. J. Enhanc. Res. Manag. Comput. Appl., vol. 5, pp. 2319–7471, 2016.

[12] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design. 2017.

[13] Y. Wang, H. Yu, Z. Zhu, W. Zhang, and Y. Zhao, "Automatic Software Refactoring via Weighted Clustering in Method-Level Networks," IEEE Trans. Softw. Eng., vol. 44, no. 3, pp. 202–236, 2018.

[14] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "A two-step technique for extract class refactoring," ASE'10 - Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng., pp. 151–154, 2010.

[15] G. Bavota, A. De Lucia, and R. Oliveto, "Identifying Extract Class refactoring opportunities using structural and semantic cohesion measures," J. Syst. Softw., vol. 84, no. 3, pp. 397–414, 2011.

[16] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "Automating extract class refactoring: an improved method and its evaluation," Empir. Softw. Eng., vol. 19, no. 6, pp. 1617–1664, 2014.

[17] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 93–101, 2009.

[18] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Identification and application of Extract Class refactorings in object-oriented systems," J. Syst. Softw., vol. 85, no. 10, pp. 2241–2260, 2012.

[19] M. Hamdi, R. Pethe, A. S. Chetty, and D. K. Kim, "Threshold-driven class decomposition," Proc. - Int. Comput. Softw. Appl. Conf., vol. 1, pp. 884–887, 2019.

[20] I. Bassey, N. Dladlu, and B. Ele, "Object-Oriented Code Metric-Based Refactoring Opportunities Identification Approaches: Analysis," Proc. - 4th Int. Conf. Appl. Comput. Inf. Technol. 3rd Int. Conf. Comput. Sci. Appl. Informatics, 1st Int. Conf. Big Data, Cloud Comput. Data Sci., pp. 67–74, 2017.

[21] Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Evaluating the Benefits of Model-Driven Development," Adv. Inf. Syst. Eng., no. June

2021, pp. 353–367, 2020.

[22] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," Inf. Syst., vol. 36, no. 2, pp. 498–516, 2011.

[23] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," J. Comput. Appl. Math., vol. 20, pp. 53–65, 1987.

[24] F. Palomba et al., "Landfill: An Open Dataset of Code Smells with Public Evaluation," 2015 IEEE/ACM 12th Work. Conf. Min. Softw. Repos., pp. 482–485, 2015.