Design Level Class Decomposition using the Threshold-based Hierarchical Agglomerative Clustering

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 activity is essential to maintain the quality of a software's internal structure. It decays as the impact of software changes and evolution. Class decomposition is one of the refactoring processes in maintaining internal quality. Mostly, the refactoring process is done at the level of source code. Shifting from source code level to design level is necessary as a quick step to refactoring and close to the requirement. The design artifact has a higher abstraction level than the source code and has limited information. The challenge is to define new metrics needed in class decomposition using the design artifact's information. Syntactic and semantic information from the design artifact provides valuable data for the decomposition process. Class decomposition can be done at the level of design artifact (class diagram) using syntactic and semantic information. The dynamic threshold-based Hierarchical Agglomerative Clustering produces a more specific cluster that is considered to produce a single responsibility class.**

Keywords—Refactoring; design level refactoring; software refactoring; hierarchical clustering; class decomposition

I. INTRODUCTION

Refactoring alters software's internal structure without changing the external behavior [1]. The primary purpose of the refactoring process is to preserve the quality of internal structure as the impact of change implementation in the evolution cycle of software. The quality of software's internal structure may decay during evolution. Many research and tools exist to expose the mostly get the structural decay as an impact of the software evolution [2]. The reduced quality of software's internal structure impacts the next changes. Decreasing the internal structure also decreases the maintainability of software and increases the effort of the next changes [3], [4]. Therefore, refactoring is recommended to solve the structural decay and to avoid the high cost that the developer must pay during the software changes process.

There are many options for refactoring the software artifact. Researchers already report and give guidance to refactoring based on the specific smells. One of the refactoring activities is the extract class. The class can be fat of functionality as an impact of the changes. The extract class decomposes the class due to class growth as changes happened during the evolution. According to the guidelines, the class must handle clear responsibility or function. Therefore, the extract class helps to maintain the class stay in clear functionality. Many research reports the methodology for class decomposition at the source code level [5]–[14]. Mostly was done by using the clustering of the class elements based on various points of view and done at the source code level.

Hamdi et al. proposed the class decomposition methodology and named it Threshold-driven Class Decomposition based on the Agglomerative Hierarchical Clustering (AHC) algorithm [11]. The class decomposition uses several metrics that can only be easily calculated at the source code level. For example, direct and indirect call dependency, internal and external call dependency, and attribute sharing. The main result in the research of Hamdi is applying the threshold to HAC to determine the termination point in the process of decomposition. The result of Hamdi's algorithm sounds more promising in the case of termination state than the other approach.

Now-a-days, researchers have changed the refactoring object and shifted it to the design level. Shift to the design is considered necessary due to the easiness of model transformation. Model as an object of transformation is like a bridge between software artifacts (act integrally). It is bridging between the requirement and the implementation artifact. The model refinement is close to the requirement and implementation artifact. It will have an impact on both sides. It can also be used for software evolution and implementation in code. Vertical and horizontal model transformations are both possible. Vertical transformations are used when the source and target models have different levels of abstraction. On the other side, a horizontal transformation occurs when the source and target models have the same abstraction. The source model transformation does not impact the target model's behavior [15].

The class diagram is one of the design or model artifacts. The class diagram is more abstract than the source code. According to the limited information, working with the higher model level has a big challenge. The lower level of abstraction of the model has more detailed information than the higher level of the model. The similarity metric was used to do decomposition using Hamdi's approach. The metric is calculated based on the information in the source code. Using Hamdi's metric is difficult if the experiment's object is changed to the class diagram. The similarity value between elements is the mandatory requirement for clustering or decomposing using a hierarchical clustering algorithm. Shifting the experiment's object to the class diagram gives the consequences to define the new similarity metric based on the information from the class diagram. One outcome of this research is determining the new similarity metrics calculated using all of the data or information from the class diagram. The metrics have a function to measure the distance between elements. In the case of class, it is not only the methods but also the attributes. The similarity metrics are considered the class's syntactic and semantic information in the class diagram. Besides shifting the experiment's object, the decomposition process only focused on the Blob class in the class diagram.

The rest of the paper is organized as follows. Section 2 summarizes the state of the arts of class decomposition approach. Section 3 describes the class usability and compactness of the class in the decomposition process. Sections 4 and 5 explain the proposed algorithm and the research scenarios. Section 6 describes the result and discussion. Then the last is the conclusion and future work in Section 7.

II. Related Work

Several researchers are researching class decomposition as the refactoring activity. The class decomposition is used to maintain the class stay in the clear functionality. The development of research on class decomposition is as follows.

A Two-Step Technique for Extract Class Refactoring by Bavota et al. told about the extract class's approach based on the responsibility [5]. The object study is source code. This experiment considers the structural and semantic information inner the class. Specifically, this uses the metrics to measure the structural and semantic similarity. The metrics are Structural Similarity between Methods (SSM), Call-based Interaction between Methods (CIM), and Conceptual Similarity between Methods (CSM). The chains extraction (The Transitive Closure), as the proposed approach, is obtained by computing the transitive closure of the method-by-method matrix. The value of the matrix is calculated based on the combination of three metrics. Then, the approach is used to extract the class using the threshold minCohesion and minLength. minCohesion is the similarity value between methods, and minLength is the minimum chain length in the graph. By using both thresholds, the class is split into several classes.

The following paper from Bavota et al. discussed the identification of extract class refactoring opportunities using structural and semantic cohesion metrics [6]. In Bavota's refactoring process, the class partition process uses the MaxFlow-MinCut algorithm. Bavota implemented the algorithm as follows. In particular, let c be the class to be refactored and $M(c) = \{m_1, m_2, \dots, m_n\}$ be the set of its methods (including the constructor and static methods). The first main process in this approach is defining the graph showing the relationships between methods. The complete graph is defined as G_M. A set of weighted edges connects all the class's pairs of methods. The weight of each edge is represented by the value of the relatedness rate of a pair of methods. The weight of edges is computed by considering the Structural Similarity between Methods (SSM), Call-based Dependence between Methods (CDM), and Conceptual Similarity between Methods (CSM). Once the graph is computed (weighted), a MaxFlow-MinCut algorithm is used to obtain a partition of the original class.

The next paper of Bavota et al. discussed the usefulness of using class structural and semantic information to extract class [7]. The structural and semantic are measured using SSM, CDM, and CSM. The final conclusion is that using a combination of structural and semantic information is worth doing extract class. This paper also said that the transitive closure approach is better than the MaxFlow-MinCut approach. Also, Bavota compares the transitive closure with the other approach, for example, Fokaefs et al., that uses a hierarchical clustering to extract class refactoring [10]. The transitive closure can split a Blob class into more than two classes, overcoming the MaxFlow-MinCut approach. And, it can automatically define the number of classes that should be extracted from a Blob class.

Isong Bassey et al. talk about the metric-based refactoring opportunities identification (ROI) for object-oriented software systems [14]. They carried out a comprehensive analysis on sixteen (16) primary studies to identify the state of the practice in ROI. This paper is summarized all refactoring opportunities that already existed before 2016. They separated analysis into three groups: the structural, the semantic, and the structural and semantic. This paper focuses on the source of information that can be used to identify the refactoring opportunities using the matrices. This paper summarized several research experiments, such as Al Dallal for the structural approach, Bavota and Al Dallal for the structural and semantic approach already published in several papers. All papers use a metric-based analysis approach. The same review is also already done by Al Dallal [16].

Wang Ying et al. talk about automatic software refactoring using weighted clustering [12]. This paper focuses on class-level refactoring. They consider the dependency relation between methods (as nodes) described as a network. Three matrices explain 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). Not only the tree matrices but also Semantic Similarity Weight (SSW) is used to calculate the weight of the edge. The most beneficial cluster with the specific weight is chosen as the solution. The result is compared with the experiment conducted by Bavota and Fokaefs. Wang only compares the algorithm with Bavota and Fokaefs in terms of the effectiveness of clustering. Wang's approach can resolve cohesion and coupling problems without changing the code's external behavior. And, it can help to improve the understandability, flexibility, reusability, and maintainability of code.

Mohammed Hamdi talks about the class decomposition method using the Hierarchical Agglomerative Clustering (HAC) [11]. The decomposition is iterative until classes have a single responsibility. The main problem is considering the difficulty of terminating the decomposition process. This paper defines the notion of threshold to determine the termination point in the decomposition process. This paper explains that class responsibility is identified using method similarity based on internal and external class relationships. There are six matrices, Internal Attribute Sharing (IAS), Internal Direct Call dependency (IDC), Internal Indirect Call dependency (IIC), Internal Method Sharing (IMS), External Indirect Call dependency (EIC), and External Call Dependency (ECD). By using the weighted AHC, the result is considered better. This approach looked like solving the problems of limitation of a number of resulting classes and the termination state of the decomposition process. The research conducted by Hamdi is the latest research that raises the problems in previous research.

III. Syntactic and Semantic Metrics

Shifting to the design artifact, especially the class diagram, brings out the new challenge of defining new metrics. The metrics are used to calculate the similarity score between the class's elements (attribute and method) used in the decomposition process. The metrics are calculated based on the information in the class diagram. There are two approaches to define the similarity rate between class elements: syntactic and semantic analysis. It means that the two approaches measure the similarity score using the similarity of syntax and meaning.

The class diagram is one artifact that shows the relation of objects in the software system. It shows the inner structure of every class and the relation between them. When reading the class diagram, we got pure in the model level consisting of text and notation. It is not easy to collect information based on the image of the class diagram. However, by converting it into the XML file, all the information of the class diagram is easier to collect. [17], [18]. Extracting the information from the XML file uses the text-based extraction method. Therefore, syntactic and semantic analysis is appropriate to calculate the similarity metric from the XML class diagram.

A. *The Type References Similarity (Syntactic)*

Syntactic analysis means the analysis based on the actual syntax that lies on the class diagram. This approach is inspired by Al Dallal et al. [19] in their proposed cohesion metrics. In this approach, the class elements are considered a relationship if they have the same type. The type similarity is based on the type of attributes and methods (references variable or return value). The value will be 0 or 1. 0 means there is no relation, and one, there is a relation between method and attribute.

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

All the attribute and method data types are collected and then mapped into the relation matrix between them in every class.

B. *The Meaning Similarity (Semantic) of the Label*

Semantic analysis is the meaning analysis between attribute and method. This analysis considers the meaning of every label in the class (related to the class elements)—for example, the name of a method, attribute, and method's parameters. The label names are split into some words to make it easier to get the meaning. The semantic similarity is calculated based on the closest meaning between words with a value between 0 and 1. The higher value means the close of meaning. Semantic similarity can be calculated using the following formula.

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

Previously, the formula was used to calculate the semantic similarity between process names by Dijkman et al. [19]. From that formula, w_1 and w_2 are the word collections from every compared label. The words are extracted from the attributes or methods labels. The Dijkman's formula is considered appropriate to use in the case of class element's label. For an example of the splitting process, there is a label named "transcriptType." Then the "transcriptType" is split into "transcript" and "Type." Sometimes, the label of attributes or methods is written without using capital characters as a beginning of words. If not possible to split appropriately, then the splitting process is done by comparing the longest fragment of a word with the dictionary. If it exists in the dictionary, then it will be separated from the label. $s(w_1, w_2)$ or $s(w_2, w_1)$ is the number of words that have a synonym relationship between two labels. The synonymity of words is based on the calculation of relatedness by considering the depths of two words in the WordNet taxonomy (Wu-Palmer) [20]. A couple of words above 0.5 is considered a synonym. wi and ws are the weight that is defined for a similar word and the word that has semantic similarities (synonym). Dijkman defines the value of $wi = 1$ and $ws = 0.75$ [19].

C. *Elements Similarity Metric*

The element similarity metric is the combination of both syntactic and semantic metrics. The formula to calculate the similarity between method and attribute is described as follows. syn and sem are syntactic and semantic similarity scores respectively. Then, $e1$ and $e2$ are elements in the class. It can be the attributes or methods of the class. The whole formula for the element similarity metric is described as follows.

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

This formula is used to define the similarity matrix that will be an input to the algorithm for decomposing the class.

IV. Threshold-based Agglomerative Hierarchical Clustering Algorithm

Threshold-based agglomerative hierarchical clustering is divided into static threshold and dynamic threshold hierarchical clustering. The difference between static and dynamic is the definition of the threshold. The static approach defines the threshold value at the beginning of the decomposition process. It is done only one time. The dynamic approach calculates the threshold in every cycle of the decomposition process. The threshold is adjusted based on the matrix changes in every step [11]. This research uses Hamdi's algorithm but is implemented at the design level. The similarity matrix considers syntactic and semantic aspects of the element's label. The decomposition is based on the similarity matrix composed using the formula (3) value. Later, the similarity matrix is used to compose the dissimilarity matrix to validate the decomposition result. Fig. 1. shows the dynamic threshold for threshold-based agglomerative hierarchical clustering. The static and dynamic threshold differences are located at the calculated threshold process. The calculation threshold is done once in the static approach and done in every process cycle in the dynamic approach. The process is run recursively and implemented in the prototype application.

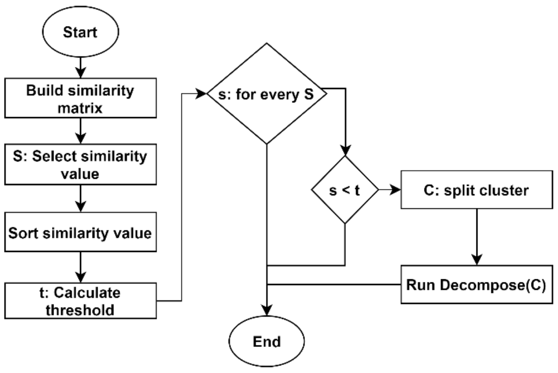

Fig. 1. Dynamic Threshold-based Agglomerative Hierarchical Clustering [11].

V. Research Scenarios

This chapter explains the scenarios of the experiment using new metrics from the class diagram and implements it using the threshold-based AHC by Hamdi. The scenario explains the dataset, tools, and validation process.

This experiment focuses on shifting from the source code to the design level. The proposed metric is implemented using two cases. First is the same case as Hamdi's paper (Transcript class), then called case 1. The Transcript class is regenerated as a class diagram using the Visual Paradigm and converted into XML files. It has been modified at the data type of the parameters variable using the types often used for those variables. The class Transcript is described as Fig. 2.

Second is the other case based on the smell dataset from the Landfill dataset [21]. Case 2 is the MDIApplication that is taken from the jHotDraw application. The MDIApplication is considered as Blob class based on the Landfill dataset. The MDIApplication is shown in the following Fig. 3.

The threshold-based agglomerative hierarchical clustering algorithm proposed by Hamdi et al. is implemented into the prototype application. This application is considered important due to the time of the calculation process. Also, the prototype application is built for the accuracy of the calculation. It also implements the tree-based keyword search algorithm useful for information collected over the XML class diagram [17]. The calculation of similarity meaning between words (semantic) is using the WordNet library for Java to support the semantic score between labels using formula (2).

The class diagram from the example cases (Fig. 2 and Fig. 3) is converted into an XML file then extracted using the prototype application. Then, the prototype application will automatically generate the similarity matrix based on the formula (3). Table I shows the value of similarity between elements (attributes and methods) of the Transcript class. Before that, the labels of every element are named using the prefix "a" for attribute, and "m" is for a method then followed by a number as elements index. Finally, the decomposition process is done using the threshold-based AHC (both static and dynamic). And the result of the experiment will be compared with the result of class decomposition at the source code level. Table II shows Hamdi’s similarity matrix of the Transcript class.

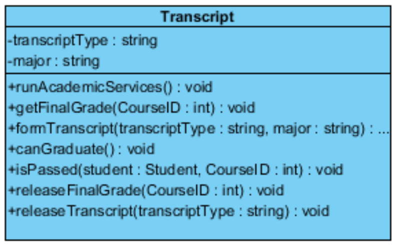
Transcript
-transcriptType : string
-major : string
+runAcademicServices() : void
+getFinalGrade(CourseID : int) : void
+formTranscript(transcriptType : string, major : string) : ...
+canGraduate() : void
+isPassed(student : Student, CourseID : int) : void
+releaseFinalGrade(CourseID : int) : void
+releaseTranscript(transcriptType : string) : void

Fig. 2. Transcript Class.

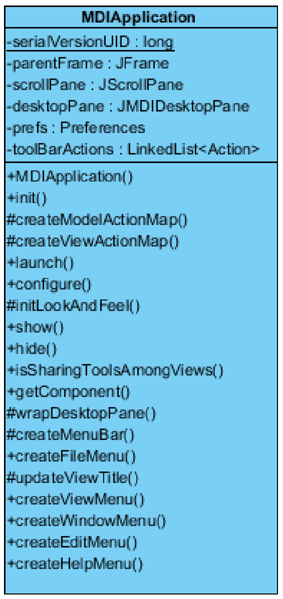

Fig. 3. MDIApplication Class from the jHotDraw.

TABLE I. THE SIMILARITY MATRIX OF TRANSCRIPT CLASS (DESIGN LEVEL)

	a1	a2	m1	m2	m3	m4	m5	m6	m7
a1		0.59	0.15	0.23	0.65	0.28	0.16	0.28	0.93
a2			0.15	0.09	0.58	0.19	0.05	0.14	0.65
m1				0.92	0.63	0.73	0.73	0.92	0.75
m2					0.71	0.78	1.00	1.00	0.88
m3						0.65	0.59	0.71	0.79
m4							0.71	0.83	0.80
m5								1.00	0.73
m6									0.94
m7									

TABLE II. THE SIMILARITY MATRIX OF TRANSCRIPT CLASS (PREVIOUS APPROACH) [11]

	m4	m5	m6	m7	m8	m9	m10
m4		0.25	0.33	0.25	0.25	0.32	0.25
m5			0.18	0.18	0.18	0.19	0.18
m6				0.08	0.08	0.26	0.2
m7					0.22	0.12	0.05
m8						0.05	0.05
m9							0.19
m10							

The validation uses two approaches. The first approach is by comparing the compactness rate using the Silhouettes index of cluster result. And the second approach is the conformance rate between the source code and the class diagram level's result. The compactness rate of every cluster resulting from the decomposition process will calculate using the Silhouettes index. Silhouettes index is the method that can be used to validate the consistency data in the cluster [22]. Then, the deep analysis of the result according to the applicability of the class is also considered important.

VI. Result and Discussion

A. *Result of Experiment*

The two metrics (syn and sem) are used to calculate the similarity between class elements then the similarity matrix is composed for every case. Finally, the similarity matrix from the two cases is used for the decomposition process using the prototype application. Case 1 is the decomposition process using class Transcript at the design level. The result of decomposition is differentiated based on the approach (static and dynamic threshold). First, the static threshold decomposition result is shown in Table III. Then, the result of the dynamic decomposition is shown in Table IV.

Table III shows the result of the decomposition process using the static threshold. It also shows the result of the Silhouettes index for every element inner the cluster. There are two clusters resulting from the static threshold decomposition process. The average of the Silhouettes index for all elements is 0.24.

Table IV shows the result of decomposition using the dynamic threshold. It is shown that the number of clusters resulting from the decomposition process is four clusters. Every element in the clusters has the Silhouettes index score that expresses the validity of the position of current elements in the specific cluster. If the score is close to 1, it is considered in the better cluster. Otherwise, it is considered the worse cluster. The average of Silhouettes from all elements is 0.35 using the dynamic threshold decomposition process.

Case 2 is taken from the jHotDraw application. The class MDIApplication will be decomposed due to the Blob indicators that are shown in the Landfill dataset. Same with the first case, the second case also threatened using the same experiment. The result of the static threshold decomposition is described in the following Table V.

There are two clusters resulting from the static threshold decomposition process. Every cluster represented the class after decomposition. Every element of the class has a Silhouettes index score representing the specific cluster's validity position. Many elements have negative of Silhouettes index. The average of Silhouettes for all elements is 0.08.

The dynamic decomposition result using case number two is described in Table VI. There are 12 clusters, and every element of each cluster is calculated. There are also many negative Silhouettes scores. The average Silhouettes of all elements is 0.15. This is because many clusters only have one element.

TABLE III. The Static Threshold Decomposition (Case 1)

Cluster	Elements	Silhouettes Index
1	canGraduate	-0.52
	releaseTranscript	-0.31
	runAcademicServices	-0.72
	formTranscript	-0.03
	transcriptType	0.38
	major	0.37
2	isPassed	1.00
	releaseFinalGrade	1.00
	getFinalGrade	1.00
Average Silhouettes		**0.24**

TABLE IV. The Dynamic Decomposition (Case 1)

Cluster	Elements	Silhouettes Index
1	formTranscript	-0.34
	major	-0.11
2	canGraduate	-0.18
	runAcademicServices	-0.48
3	releaseTranscript	0.50
	transcriptType	0.80
4	isPassed	1.00
	releaseFinalGrade	1.00
	getFinalGrade	1.00
Average Silhouettes		**0.35**

TABLE V. The Static Decomposition (Case 2)

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 VI. THE DYNAMIC DECOMPOSITION (CASE 2)

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

B. Compared with the Previous Approach

The result of the experiment using new metrics (syn and sem) at the design artifact shows different from the result of the previous approach. The previous approach uses six metrics at the source code level to calculate the similarity between class elements. The decomposition result using the previous approach will be assessed using the Silhouette index, then compared with the result of the current experiment. This comparison will focus on the Silhouette index value of decomposition using case 1.

The calculation of the Silhouette index of previous is based on the dissimilarity matrix calculated from the similarity matrix. The Silhouette index comparison of the previous and proposed approaches using case 1 is shown in Table VII. Based on the result shown in Table VII, the average of Silhouette using the previous approach is shown little different from the current approach. In the previous approach, using the static threshold, the average of the Silhouette index is 0.05. The dynamic threshold approach produces the average Silhouette as -0.02. The current approach for static and dynamic are 0.24 and 0.35, respectively. The experiment is used the prototype application to apply the decomposition using the previous approach's similarity matrix (Table II).

The previous experiment uses the six metrics to calculate the similarity between the class's elements at the source code level. The current experiment is done at the design level using two metrics gathered from the class diagram. The comparison previous and proposed approach is to know how the conformance rate between each other.

Tables VIII and IX show the clustering result using each approach (previous and proposed). Using different metrics and objects of study, the previous and proposed approach results in the same number of clusters but different elements. The conformance is calculated by dividing the number of conformed elements at both results by the number of all elements. For example, for the static threshold AHC, four elements conform at both sides of the result. m4 and m6 are located at the same cluster (also m5 and m8). The rest element, m7, m9, and m10, are considered not to conform. It is similar to the dynamic threshold AHC. Four elements conform at both sides (m10, m7, m9, and m6). The conformance rate for both results is $\frac{4}{7} = 0.5714$. The proposed experiment uses two metrics from the class diagram to do the class decomposition results 0.5714 conformance rate.

C. Discussion

Tables III, IV, V, and VI show the result of the experiment, not only the number of the cluster but also the Silhouettes index score for every element. The Silhouettes index shows the validity of every element placed in a specific cluster. A higher score is better for Silhouettes.

TABLE VII. THE AVERAGE SILHOUETTE INDEX COMPARISON (CASE 1)

	Previous Approach		Current Approach	
	Static	Dynamic	Static	Dynamic
Case 1	0.05	-0.02	0.24	0.35

TABLE VIII. THE AVERAGE SILHOUETTE INDEX OF STATIC THRESHOLD (CASE 1)

Static Threshold			
Cluster	Previous	Cluster	Proposed
1	m5	1	m7
	m7		m10
	m8		m4
	m10		m6
2	m4	2	m8
	m6		m9
	m9		m5

TABLE IX. THE AVERAGE SILHOUETTE INDEX OF DYNAMIC THRESHOLD (CASE 1)

Dynamic Threshold			
Cluster	Previous	Cluster	Proposed
1	m5	1	m6
	m10	2	m7
2	m7		m4
	m8	3	m10
3	m9	4	m8
4	m4		m9
	m6		m5

The differences and the trend of the Silhouettes index between static and dynamic threshold AHC are interesting to discuss. Case 1 shows that the average of Silhouettes of static threshold is 0.25. And, the dynamic threshold has the average of Silhouettes is 0.35. In case 2 from jHotDraw, the static threshold shows that the average Silhouettes are 0.08 and 0.15 for the dynamic threshold. The dynamic threshold AHC produces the higher Silhouettes index score from the two cases. The dynamic threshold AHC is better than the static threshold AHC for those two cases in the design phase.

The comparison with the previous approach's average Silhouette shows a different trend. The previous approach shows that the static threshold AHC has a better value of the Silhouette index than a dynamic threshold. It is shown that the value of static and dynamic are 0.05 and -0.02, respectively. In this case, the value of the similarity matrix used in the experiment to decompose the class is taking an important position. The use of six metrics to calculate the similarity value and then decomposed using the static and dynamic threshold AHC is slightly lower than the use of two metrics (proposed experiment). The compactness of each cluster resulting from the decomposition process depends on the metrics used. Using two metrics, it shows the conformance rate of about 0.5714. It means that four elements conformed to each other (previous and proposed approach). With the 0.5714 conformance rate, the proposed approach’s result has a better average of Silhouettes. But, this result cannot be used to justify which one is better. The decomposition result's correctness might be able to be found by deep analysis at the implementation level after the decomposition is finished.

The other angle of results shows that there are many elements with negative Silhouettes. It means that the elements are placed in the worse cluster. Silhouettes also show the density of every element in every cluster. The higher Silhouettes score shows the distance between elements inner the specific cluster is close to each other and far from the other cluster. High differentiation of distance between clusters is better for clustering results.

The decomposition of class using static and dynamic threshold AHC leaves the existence of many elements with a negative score of the Silhouettes index problem. To overcome this problem is needed to move the element with negative Silhouettes to the other cluster by comparing the Silhouettes score before and after movement. The better Silhouettes score will be chosen to increase the validity of the cluster.

The other point that is interesting to discuss is the result of clustering. The number of clusters between static and dynamic thresholds is always increased. The dynamic threshold produces more clusters than static. The static threshold produces less number of clusters, and the other side consists of a bigger element inner of every cluster and vice versa for the dynamic threshold AHC. Hamdi et al. said that the static threshold is suitable for fine-grained decomposition, and the dynamic is suitable for coarse-grained decomposition [11]. Based on the definition, the fine-grained will produce smaller objects.

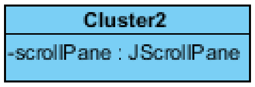

Fig. 4. Cluster Number Two.

Fine-grained produces more objects than coarse-grained decomposition. But it differs from Hamdi's result in the class diagram and uses the two metrics (syn and sem) for distance similarity calculation. In this experiment, using dynamic threshold AHC creates more clusters with smaller elements. And the static threshold produces a smaller number of clusters with a bigger number of elements in every cluster. It can happen because of the threshold. Different from the static, the dynamic threshold alters the threshold in every cycle of the decomposition process to find the separation limit. That is why the dynamic threshold has higher opportunities to create a new cluster in every decomposition cycle.

In the class decomposition process, the main purpose is to decompose the class by distinguishing the functionality of the class. The dynamic threshold AHC can find the single functionality inner the decomposed class. Using syntactic and semantic metrics in the class diagram, the dynamic threshold AHC is more distinctive than the static threshold to do decomposition. The dynamic threshold that is calculated on every cycle process makes the decomposition done in fine-grained size.

The dynamic threshold AHC results in more clusters than static, but the number of elements in every cluster is smaller than the static threshold. Some clusters only consist of one element, for example, cluster number two resulting from dynamic threshold decomposition of case 2 (see Table VI). The element in cluster number two is only scrollPane, and it is an attribute. From the detailed class in Fig. 3, the scrollPane has a private modifier. This will be the next interesting problem to discuss. Cluster number two will be one class with only one attribute and is private. Fig. 4 shows how cluster number two is realized into the class.

Class from cluster number two is doubtful to use. There is only one attribute, and it has a private modifier. The object from the class of cluster number two will be unable to access. The other word, the object will not collaborate with the other objects in the software system. The main purpose of object in the software system is to do sub-function to fulfill at least one of the software functionality. Ideally, the attribute is private to match the theory of information hiding. But, usually, at least there is one method that has a public modifier. The method has functioned as the object's boundary to access the data or process provided by the object. If there is no public method, then it will be an unuseful object. The movement process must solve this condition. For the class that only has private elements, the element must be moved to the other more valid cluster by comparing the validity or compactness rate of the element.

VII. Conclusion and Future Work

The refactoring can be done by using the design artifact. This paper shows the new metrics from the class diagram to do the decomposition of class using the class diagram. The metrics are syn and sem that measure using analysis of syntax and meaning. The metric uses the information gathered from the class diagram to calculate the similarity matrix. The decomposition process uses the algorithm from the previous approach but is implemented in the class diagram as a design-level artifact. The decomposition result shows a few points of conclusion.

The first conclusion is the differences between the previous and the current decomposition result. The current result of decomposition shows that the clusters resulting from the static and dynamic threshold AHC are more compact than the previous approach's result. It is validated using the Silhouette index to measure the compactness of the clusters.

Both approaches produce the same number of clusters, whether using the static or dynamic threshold AHC. But, some of the elements are different between previous and proposed approaches. The conformance rate of both (previous and proposed) approaches is 0,5714, with the proposed approach result showing higher compactness.

In the proposed experiment, there is a trend in the Silhouette index value of the proposed experiment's static and dynamic threshold result. The dynamic threshold is higher than static in the Silhouette value in both cases. Dynamic threshold AHC produces a more compact cluster than the static. The dynamic threshold AHC also produces more number of a cluster than the static threshold. On achieving single responsibility principles, the dynamic threshold AHC's result shows more specific than static because the result of the cluster consists of a lower number of elements but a higher Silhouette index as a measurement of compactness.

The result shows the advantages that can be obtained and shows that there are still shortcomings. The decomposition result still shows the elements that have the negative Silhouette value. The negative Silhouette value shows that the distances of the current element are far from the other elements in the same cluster. The other word, the negative Silhouette elements are considered the worse place. The enhancement for the moving mechanism of the negative element is considered important.

The result also shows that some clusters are considered unable to implement because the cluster may produce objects that cannot collaborate with others. The cluster that only has one element, specifically if the element has a private modifier, is considered a useless cluster. From this fact, it is considered important to include the modifier aspect to do the decomposition process. It is important to avoid the emergence of useless clusters.

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] F. A. Fontana, P. Braione, and M. Zanoni, "Automatic detection of bad smells in code: An experimental assessment," J. Object Technol., vol. 11, no. 2, 2012.

[3] A. Yamashita and L. Moonen, "Exploring the impact of inter-smell relations on software maintainability: An empirical study," Proc. 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, 2018.

[5] 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.

[6] 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, Mar. 2011.

[7] G. Bavota, "Using structural and semantic information to support software refactoring," Proc. - Int. Conf. Softw. Eng., pp. 1479–1482, 2012.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] N. Anquetil, A. Etien, G. Andreo, and S. Ducasse, "Decomposing God Classes at Siemens," Proc. - 2019 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2019, pp. 169–180, 2019.

[14] 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.

[15] M. Misbhauddin and M. Alshayeb, "UML model refactoring: a systematic literature review," Empir. Softw. Eng., vol. 20, no. 1, pp. 206–251, 2013.

[16] J. Al Dallal, "Identifying refactoring opportunities in object-oriented code: A systematic literature review," Inf. Softw. Technol., vol. 58, pp. 231–249, 2015.

[17] 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.

[18] 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.

[19] 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.

[20] J. B. Gao, B. W. Zhang, and X. H. Chen, "A WordNet-based semantic similarity measurement combining edge-counting and information content theory," Eng. Appl. Artif. Intell., vol. 39, pp. 80–88, 2015.

[21] F. Palomba et al., "Landfill: An open dataset of code smells with public evaluation," IEEE Int. Work. Conf. Min. Softw. Repos., vol. 2015-Augus, pp. 482–485, 2015.

[22] 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.