Automatic Class Decomposition using Clustering

Maen Hammad, Rua'a Hasan Banat
Department of Software Engineering
The Hashemite University
Zarqa, Jordan
{mhammad@hu.edu.jo, ruaa.hassan92@gmail.com}

2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C) | 978-1-6654-3910-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSA-C52384.2021.00019

***Abstract*—This paper presents an automated technique for architectural decomposition of classes using K-Means clustering. The goal is to help designers in grouping related classes into subsystems or packages. The proposed technique utilizes the K-Means clustering algorithm to cluster a set of java classes into disjoint groups. The technique analyzes source code of classes to represent classes as sets of identifiers. Then, the K-Means clustering technique is applied on these identifiers. The proposed technique has been evaluated on a number of packages from an open source project. Results showed that the proposed technique can achieve good results when the system architecture is composed of packages with large number of classes.**

Keywords— Software Design, Classes Decomposition, K-Means Clustering.

I. INTRODUCTION

Software design is an essential phase in object oriented development. During this phase, developers work on the classes that were generated after analyzing user's requirements. One of the main activities of the design phase is class decomposition. Designers have to group large number of classes into packages or subsystems. Each package should have a number of related or collaborated classes that perform similar or specific tasks. Good class decomposition leads to better and high quality design.

During object oriented analysis phase, large number of classes is generated. These classes are refined and revisited during the design phase. Then, the decomposition step is performed. Initially, we may have all system's classes are grouped in one large package. These classes could be hundreds. This large package need to be decomposed into smaller ones. We used the term decomposition to refer to the process of decomposing large number of classes into smaller packages. As a result, each package should have a small number of related classes. Also, during reengineering activities, maintainers may face a very large package with many unrelated classes. This package may need to be decomposed to smaller packages.

The problem that we are addressing is how to help designers or maintainers in their classes decomposition task. Manual classes decomposition consumes time and effort of designers. Large number of classes has to be decomposed into packages. Moreover, these packages should have coherent and related classes. One possible solution of this problem is to provide designers with an automated tool to decompose large number of classes into disjoint sets. These sets can be candidates for packages that satisfy the cohesion design principle.

This paper presents an automated technique, supported by a tool, to perform class decomposition. The input of the tool is a number of classes that need to be decomposed into a number of packages. The output of the tool is a number of groups of related classes. Each group of classes can be used to form a package or a subsystem. The technique is based on clustering. The K-Means clustering technique is utilized to cluster classes based on their identifiers. The names of attributes, methods, packages, invoked methods and used classes are used in the clustering. Our premise is that the identifiers of classes can play a key role in identifying related and collaborative classes that should be grouped together.

There are many clustering algorithms that can be used. Most of the research in this area utilizes different variations from the Agglomerative Hierarchal Clustering (AHC) algorithm. We want to show that utilizing the K-means algorithm can give good results too. K-means clustering is easy to apply and has efficient execution time. We also used a simple similarity criterion. A simple text analysis is applied on the source code of classes to extract their identifiers. The identifiers are then used for applying the clustering.

The proposed technique has been applied to cluster a number of classes from an open source project. Then, a comparison is made between the generated clusters and the original decomposition of classes in the project. The preliminary results showed that the proposed technique can be used to cluster group of classes into disjoint clusters. The research contributions of this paper are summarized as follows:

- An automated technique to support classes decomposition activities.
- An automated tool to help in applying the proposed technique.
- A pilot case study on a real open source project to discuss the results of the proposed technique.

The remainder of the paper is organized as follows. Section 2 discusses the related work in the area. The proposed technique is detailed in Section 3. A case study with the evaluation of the results is discussed in Section 4. Section 5 concludes the paper with our future work.

II. RELATED WORK

Wiggerts [1] presented an overview of clustering algorithms that may be used in systems re-modularization towards object oriented structure. Lung et al. [2] demonstrated how clustering methods can be applied to software partitioning, recovery, restructuring, and decoupling. Mitchell and Mancoridis [3] used the Bunch clustering system to produce subsystems decomposition by partitioning a graph of the entities as classes) and their relations as function calls in the source code. Chetta [4] proposed a technique to recover the architecture of a large software project from its source code. His technique is based on the clustering algorithm K-means. The clustering is performed upon the dependencies graph created from classes, their interactions and their hierarchical organization. Our approach is simpler. We use

978-1-6654-3910-7/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSA-C52384.2021.00019

only the identifiers in the classes to feed the clustering algorithm.

Xu et al. [5] proposed a program restructuring approach using clustering techniques at the function level. They mainly used the hierarchical agglomerative algorithms. The approach helps software designers make a decision why and how to restructure an existing program and give them heuristic restructuring advice. Hierarchical clustering has been also used by Maqbool et al. [6] for software architecture recovery. Lung et al. [7] used hierarchical agglomerative clustering methods for decomposition of requirements into subsystems. According to Cui et al. [8], applying agglomerative hierarchical clustering algorithms to component identification has difficulties in producing perfectly and satisfactory results

Also, Alkhalid et al. [9] used clustering techniques for software architecture decomposition. They used K-nearest neighbor algorithm and two hierarchical agglomerative clustering methods. They used clustering techniques to compute the similarity for a number of components based on attributes.

Mancoridis et al. [10] proposed a novel method for modularizing a software system. They aimed to provide a good cluster in terms of high cohesion and low coupling. They essentially use hill-climbing and genetic algorithms. They produce the module dependency graph based on the source code, then applied cluster technique on the resulting graph. They used the Hierarchical clustering algorithm. Sartipi and Kontogiannis [11] proposed a clustering framework for recovering the architecture of a software system. The technique measures the association between the files or packages. The clustering technique extracts the maximum association between the system entities.

Clustering techniques are used in solving the problem in source code when classes may become very large, complex and less cohesive. Fokaefs et al. [12] proposed a class decomposition method using a HAC algorithm. They used the clustering technique to identify cohesive groups of entities. They used dependency information from the source code to calculate distances between class members. Another work is done by Liu et al. [13]. They proposed an adaptive fuzzy clustering technique based on clustering algorithms. The proposed technique studies distance, density and distribution of data instances. The proposed technique grouped software component instances into cohesive clusters based on the similarity of components. A different probabilistic approach for software systems partitioning is presented in [14]. The approach used lexical information extracted from java classes. Garcia et al. [15] presented a machine learning-based technique for recovering an architectural view that contains components and connectors. In their approach, they recover software concerns to help identify components and connectors.

In summary, our technique differs from the related work in the area by applying the K-means algorithm for class decomposition with simple similarity criteria. No graph dependencies nor sophisticated code parsing is needed. Only the identifiers in the classes are extracted with simple text analysis.

III. METHODOLOGY

The proposed technique is an automated process. It starts by analyzing the textual contents of classes. The analysis aims to extract the identifiers of the class. In the next step, a term matrix is generated for all classes based on their identifiers. Finally, the K-means clustering is applied on the generated term matrix. The result will be disjoint clusters for classes. Each cluster is considered as a package or subsystem of related classes. The main steps of the proposed technique are detailed in the following subsections.

A. Tokenization

All classes under consideration are processed one by one. Each class is textually analyzed to extract all its tokens. The tokenization is performed based on the operators of the programming language. The extracted tokens for each class are saved in a set that corresponds to that class.

B. Extraction of Identifiers

In this step, the set of tokens for each class is searched to filter out the programming language's reserved words, digits and operators. Tokens that have only one letter are also filtered out. These tokens are usually local variables that are not useful for clustering. After the filtering step, the remaining tokens are the identifiers of the class. The goal is keep only the names of used libraries, attributes, variables and methods. All these identifiers appear in the class scope that will be used in the clustering step.

C. Generation of Term Matrix

A terms matrix is generated from all the sets of identifiers. The horizontal access of this matrix consists of the names of all identifiers from all sets. The vertical access represents the names of all sets (i.e. classes). The values of the cells are either zero or one. The cell value depends on the existence of the identifier in the class. This term matrix is used in the clustering step.

D. Clustering

The K-means clustering is widely used in data mining as a type of unsupervised learning technique. It starts with a group of randomly selected center point (centroids) for every cluster, and then performs iterative calculations to optimize the positions of these centroids. We performed the K-means clustering using the generated term matrix from the previous step. The main issue in K-means method is the determining of the number of clusters (K) before the clustering starts. This issue can be solved later by different techniques that are used to predict the number of clusters. This issue is out the scope of this paper.

For example, Figure 1 shows the java class UmlNotationTextEvent. After performing the tokenization and the extraction of identifiers steps, the set of identifiers is shown in Figure 2. As shown in the figure, all operators and java reserved words were filter out.

After extracting the identifiers of all classes, a term matrix is automatically generated. The term matrix shows the existence of a specific identifier in all classes under consideration. Table 1 shows a snapshot from the term matrix of the case study discussed in Section 4. The Classes column lists all classes that need to be clustered. The Terms columns show all unique identifiers extracted from all classes. The 1/0 values of the cells correspond to the existence or absence of identifiers in classes.

```
class UmlNotationTextEvent extend
                    NotationTextEvent {
      private final String text;
      private final boolean underlined;
      private final boolean italic;
      private final boolean bold;
    UmlNotationTextEvent (  final String text,
            final boolean    italic,  final boolean
            underlined, final boolean bold ) {
        this.text = text;
        this.italic = italic;
        this.underlined = underlined;
        this.bold = bold;  }
    public String getText () {  return text;  }
    public boolean isUnderlined(  ) { return underlined;  }
    public boolean isItalic(   ) {  return italic; }
    public boolean isBold(   ) { return bold;  }
    }
```

Fig. 1. Source code for a java class.

UmlNotationTextEvent; NotationTextEvent; text; underlined; italic; bold; getText; isUnderlined; isItalic; isBold

Fig. 2. The set of Identifiers for the class in Figure 1.

TABLE I. A SNAPSHOT FROM A TERM MATRIX EXAMPLE

Classes	Terms		
	owner	paint	Param
DeploymentDiagramFactory	1.0	0.0	0.0
DeploymentDiagramGraphMode	0.0	0.0	0.0
FigNode	1.0	1.0	0.0
FigPort	1.0	0.0	0.0
UMLDeploymentDiagram	1.0	0.0	1.0
DeploymentDiagramModule	0.0	0.0	0.0
DeploymentDiagramPropPanelFactory	0.0	0.0	0.0

E. Supporting Automation

A tool has been implemented to automatically realize and apply the proposed technique. The input of the tool is a specific number of java classes. The tool performs; tokenization, extraction of identifiers and the generation of the terms matrix. Finally, the clustering step is done by the Weka tool based on the generated terms matrix.

IV. CASE STUDY AND EVALUATION

The proposed technique has been applied and evaluated using a real case study. Six packages were randomly selected from the UML modeling tool ArgoUML. It is an open source java project (http://argouml.tigris.org/). The total number of classes in the selected six packages was 34. Table 2 lists the names of classes with their packages.

For evaluation purposes, we deleted the package statement from the source code of each class. Our goal is to apply the proposed clustering technique on all the 34 classes. Then, a comparison is made between the classes of original packages and the classes of generated clusters.

TABLE II. NAMES OF EXTRACTED CLASSES WITH THEIR PACKAGES FROM THE ARGOUML OPEN SOURCE PROJECT

Package Name	Names of classes
Depdiagram	DeploymentDiagramFactory; DeploymentDiagramGraphMode; FigNode; FigPort; UMLDeploymentDiagram
Deployment	DeploymentDiagramModule; DeploymentDiagramPropPanelFactory
Notation	NameUmlNotation; NotatedItem; NotationLanguage; NotationManage; NotationModule; NotationText; NotationTextEvent; NotationType; StereotypeUmlNotation; UmlNotationLanguage; UmlNotationTextEvent
State	StateDiagramModule; StateDiagramPropPanelFactory
Ststediagram	FigCircleState; FigEntryPoint; FigExitPoint; FigRegion; FigVertex; StateDiagramFactory; StateDiagramGraphModel; UMLStateDiagram
Transformer	EventTransformer; SimpleStateTransformer; Transformer; TransformerAction; TransformerManager; TransformerModule

K-Means clustering was applied on the 34 classes. The number of clusters (K) was set to five before the clustering was started. As we mentioned before, the predefined K value is necessary to the K-Means clustering algorithm. The scope of this paper is focused on showing that K-Means clustering can give good results in class decomposition. For our future work, we plan to predict the number of clusters from the terms matrix. In this case, the number of clusters is automatically predicted and fed to the clustering tool.

As a result, five clusters with IDs: 0, 1, 2, 3, and 4 were generated by the K-means clustering algorithm. Cluster with ID=2, was the largest with 17 classes. The smallest cluster had only one class (ID=4). The generated clusters were compared to the original packages. The comparison is done based on matching the classes in clusters with the classes in packages. Based on the matching, we tied each cluster with a package. Assuming that packages are correctly decomposed, clustering should generate clusters that match packages.

Table 3 shows the evaluation results of matching the classes in their packages and the classes in the generated clusters. The first column lists the names of the original packages. The distribution of their classes over the generated clusters is shown in the second column “No. of clusters”. It shows the number of clusters that contains all the classes of the package. The cluster ID that best matches the package is shown in the third column.

TABLE III. COMPARISON RESULTS BETWEEN ORIGINAL PACKAGES AND CLUSTERS

Packages	# of clusters	Matched Cluster ID	% of matched classes
Depdiagram	3	0	2/5 (40%)
Deployment	2	None	0/2 (0%)
Notation	3	2	9/11 (81%)
State	2	None	0/2 (0%)
Ststediagram	3	1	4/8 (50%)
Transformer	3	3	2/6 (33%)

The last column shows the matching percentages of the classes in packages with classes in the matched clusters. The matching is determined based on the common classes between packages and clusters. The Matching Percentage (MP) between Cluster C and Package P is calculated as follows:

MP = number of common classes between P and C / number of classes in P

The matching percentage is an indication for the correctness of the resulted clusters in comparison with the original packages. For example, the first row in Table 3 shows the results of package Depdiagram. The closet cluster that matches this package is the one with ID=0. The number of common classes between the package Depdiagram and this cluster is two. The matching percentage is 2/5= 40%. In other words, we can say that 40% of the classes in package Depdiagram were clustered together correctly in one cluster.

Also shown in Table 3 that nine of the eleven classes of package Notation were clustered together in the cluster with ID=2. The matching percentage is 81%. It is the highest matching percentage for all packages. It is important to note that the Notation package has the largest number of classes. Package Ststediagram has eight classes which is the next largest package. Its matching percentage is 50%. On the other hand, packages Deployment and State had no matched clusters. The sizes of these packages are two classes per package.

In conclusion, we noticed that the matching percentage depends on the number of classes in packages. Packages with large number of classes have better chances to be automatically grouped by the proposed clustering technique. Systems that have very small number of classes per packages may have negative effect on the clustering results. It is important to mention that the conclusions and observations are based on the pilot case study investigated in this paper. Further investigation is needed in the future to generalize these results.

V. CONCLUSIONS AND FUTURE WORK

An automated technique has been proposed to help designers in their class decomposition activities. It is shown that decomposition of large number of classes can be achieved by clustering them based on their identifiers and by using K-means clustering. Evaluation results showed that the technique can achieve better results when the software architecture has more classes that need to be grouped in a package. Our future work aims to predict the number of clusters (K) from the terms matrix. Larger datasets are in preparation for applying comprehensive and detailed evaluation for further investigations. We also plan to include more information to enhance the clustering results. One possible addition is the Coupling between Objects (CBO) metric.

REFERENCES

[1] Wiggerts, Theo A. "Using clustering algorithms in legacy systems remodularization." In *the the Fourth Working Conference on Reverse Engineering*. IEEE, 1997.

[2] Lung, Chung-Horng, Marzia Zaman, and Amit Nandi. "Applications of clustering techniques to software partitioning, recovery and restructuring." *Journal of Systems and Software, Vo.* 73, No. 2, 2004, pp. 227-244.

[3] Mitchell, Brian S., and Spiros Mancoridis. "On the automatic modularization of software systems using the bunch tool." *IEEE Transactions on Software Engineering,* Vo. 32, No. 3, 2006, pp. 193-208.

[4] Chetta, Alessandro. "Architecture recovery using partitioning clustering technique." Thesis, POLITECNICO DI MILANO (2016).

[5] Xu, Xia, Chung-Horng Lung, Marzia Zaman, and Anand Srinivasan. "Program restructuring through clustering techniques." In *the Fourth IEEE International Workshop on Source Code Analysis and Manipulation, 2004,* pp. 75-84.

[6] Maqbool, Onaiza, and Haroon Babri. "Hierarchical clustering for software architecture recovery." *IEEE Transactions on Software Engineering* Vol. 33, No. 11 (2007), pp. 759-780.

[7] Lung, Chung-Horng, Xia Xu, and Marzia Zaman. "Software architecture decomposition using attributes." *International Journal of Software Engineering and Knowledge Engineering, Vol.* 17, No. 5, 2007, pp. 599-613.

[8] Cui, J. F., and Chae, H. S. "Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems", *Information and Software technology*, Vol. *53, No.* 6, 2011, pp. 601-614.

[9] Alkhalid, Abdulaziz, Chung-Horng Lung, and Samuel Ajila. "Software architecture decomposition using adaptive K-nearest neighbor algorithm." In *the 26th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE),* 2013

[10] Mancoridis, Spiros, Brian S. Mitchell, Chris Rorres, Y. Chen, and Emden R. Gansner. "Using automatic clustering to produce high-level system organizations of source code." In the 6th International Workshop on Program Comprehension (IWPC'98), 1998, pp. 45-52.

[11] Sartipi, Kamran, and Kostas Kontogiannis. "Component clustering based on maximal association." In *the Eighth Working Conference on Reverse Engineering*. 2001.

[12] Fokaefs, Marios, Nikolaos Tsantalis, Alexander Chatzigeorgiou, and Jorg Sander. "Decomposing object-oriented class modules using an agglomerative clustering technique." *In the 2009 IEEE International Conference on Software Maintenance(ICSM'09)*, 2009, pp. 93-101.

[13] Liu, Duo, Chung-Horng Lung, and Samuel A. Ajila. "Adaptive clustering techniques for software components and architecture." In *the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), 2015*.

[14] Corazza, Anna, Sergio Di Martino, and Giuseppe Scanniello. "A probabilistic based approach towards software system clustering." In *the 14th European Conference on Software Maintenance and Reengineering*, pp. 88-96. IEEE, 2010.

[15] Garcia, Joshua, Daniel Popescu, Chris Mattmann, Nenad Medvidovic, and Yuanfang Cai. *"Enhancing architectural recovery using concerns."* In *the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 552-555. IEEE, 2011.*