STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LII**, Number 2, 2007

A STUDY ON CLUSTERING BASED RESTRUCTURING OF OBJECT-ORIENTED SOFTWARE SYSTEMS

ISTVÁN GERGELY CZIBULA AND GABRIELA SERBAN

Abstract. The structure of a software system has a major impact on its maintainability. *Refactoring* is an activity performed through the entire lifecycle of a software system in order to keep the software structure clean and easy to maintain. We have previously introduced in [3] a clustering approach for identifying refactorings in order to improve the structure of software systems. The aim of this paper is to make a comparative analysis on several clustering algorithms (developed based on the approach from [3]) which can be used in order to recondition the class structure of a software system. Based on this analysis, we highlight the advantages of determining refactorings of object-oriented software systems using clustering.

1. Introduction

The structure of a software system has a major impact on the maintainability of the system. This structure is the subject of many changes during the systems lifecycle. Improper implementations of these changes imply structure degradation that leads to costly maintenance.

A continuous improvement of the software systems structure can be made using *refactoring*, that assures a clean and easy to maintain software structure.

In [6] Fowler defines refactoring as "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs". Refactoring is viewed as a way to improve the design of the code after it has been written. Software developers have to identify parts of code having a negative impact on the system's maintainability, and to apply appropriate refactorings in order to remove the so called "bad-smells" [1].

Received by the editors: October 20, 2007.

2000 *Mathematics Subject Classification.* 68N99, 62H30.

1998 *CR Categories and Descriptors.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement –*Restructuring, reverse engineering, and reengineering*; I.5.3 [**Computing Methodologies**]: Pattern Recognition – *Clustering*.

We have previously introduced in [3] a clustering approach for identifying refactorings in order to improve the structure of software systems. For this purpose, a clustering algorithm named *kRED* was introduced. To our knowledge, there is no approach in the literature that uses clustering in order to improve the class structure of a software system, excepting the approach introduced in [3]. The existing clustering approaches handle methods decomposition ([15]) or system decomposition into subsystems [10].

We have improved the approach from [3] by developing several clustering algorithms that can be used to identify the refactorings needed in order to recondition the class structure of an object-oriented software system: *HAC* [14], *PAMRED* [11], *HARED* [2], *HARS* [12] and *PARS* [13]. These algorithms are based on the on the idea of *partitional* and *hierarchical* clustering.

The rest of the paper is structured as follows. The main aspects related to clustering, to the approach for determining refactorings using clustering [3] and to the clustering algorithms previously developed are presented in Section 2. The comparative study between the clustering algorithms for identifying refactorings of object-oriented software systems is made in Section 3. An experiment on a real software system is reported in Section 4. Some conclusions and further work are given in Section 5.

2. Background

2.1. **Clustering.** *Clustering* [8], also known as unsupervised classification, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects, $\mathcal{O}$. The measure used for discriminating objects can be any *metric* or *semi-metric* function $d : \mathcal{O} \times \mathcal{O} \longrightarrow \Re$, called *distance*. A large collection of clustering algorithms is available in the literature ([8]). Most clustering algorithms are based on two popular techniques known as *partitional* and *hierarchical* clustering.

2.2. **A Clustering Approach for Refactorings Determination - *CARD*.** In this subsection we briefly describe the clustering approach (*CARD*) that was previously introduced in [3] in order to find adequate refactorings to improve the structure of software systems. *CARD* approach consists of three steps:

- **Data collection** - The existing software system is analyzed in order to extract from it the relevant entities: classes, methods, attributes and the existing relationships between them.
- **Grouping** - The set of entities extracted at the previous step are regrouped in clusters using a clustering algorithm. The goal of this step is to obtain an improved structure of the existing software system.

- **Refactorings extraction** - The newly obtained software structure is compared with the original software structure in order to provide a list of refactorings which transform the original structure into an improved one.

As described above, at the **Grouping** step of *CARD*, the software system S has to be re-grouped. This re-grouping can be viewed as a ***partition*** of S. We mention that a software system S is viewed in [3] as a set $S = \{s_1, s_2, ..., s_n\}$, where $s_i, 1 \leq i \leq n$ is an *entity* from the system (it can be an *application class*, a *method* from a class or an *attribute* from a class). In our clustering approach, the objects to be clustered are the entities from the software system S. Our focus is to group similar entities from S in order to obtain high cohesive groups (clusters).

2.3. **Clustering Algorithms for Refactorings Determination.** We have developed several clustering algorithms that can be used in the *Grouping* step of *CARD* in order to find an improved structure of a software system: *kRED* [3], *HAC* [14], *PAMRED* [11], *HARED* [2], *HARS* [12] and *PARS* [13].

In order to apply a clustering method for refactorings extraction, a distance function between the entities from a software system has to be defined. This distance has to express the idea of cohesion between the entities from the software system.

We have defined in [3] a modality to compute the dissimilarity degree *diss* between any two entities from the software system S. *diss* is a semi-metric and expresses the distance between the entities from the software system and emphasizes the idea of cohesion. The dissimilarity degree *diss* highlights the concept of cohesion, i.e., entities with low distances are cohesive, whereas entities with higher distances are less cohesive.

In developing our clustering algorithms, we have used two approaches:

- The first approach is to use a vector space model based clustering. We have defined a vector space that characterizes the entities from S and, based on *diss*, we have used distance metrics (*Euclidian distance*, *Manhattan distance*, *Hamming distance*) in order to express the dissimilarity between the entities from the software system. In this direction we have introduced two vector space model based clustering algorithms *kRED* and *HAC* that can be used in the *Grouping* step of *CARD* in order to obtain an improved structure of a software system.
- The second approach is to use only the distance between the entities from S given by the semi-metric *diss*. In this direction we have introduced four clustering algorithms *PAMRED*, *HARED*, *HARS*, and *PARS* that can be used in the *Grouping* step of *CARD* in order to obtain an improved structure of a software system.

Another classification of the developed algorithms is based on the clustering method used:

- *kRED*, *PAMRED*, and *PARS* are *partitional* clustering algorithms;
- *HAC*, *HARED* and *HARS* are *hierarchical* clustering algorithms.

3. Comparative analysis

In order to comparatively analyze the proposed clustering algorithms, we consider as case study the open source software JHotDraw, version 5.1 ([7]). It is a Java GUI framework for technical and structured graphics, developed by Erich Gamma and Thomas Eggenschwiler, as a design exercise for using design patterns. It consists of **173** classes, **1375** methods and **475** attributes. The reason for choosing JHotDraw as a case study is that it is well-known as a good example for the use of design patterns and as a good design.

Our focus is to test the accuracy of our approach on JHotDraw, i.e., how accurate are the results obtained after applying the algorithms in comparison with the current design of JHotDraw.

For evaluation, we use two measures:

- **Accuracy of classes recovery - ACC.** *ACC* defines the degree to which the partition obtained by the clustering algorithm is similar to the initial structure of the analyzed software system.
- **Precision of entities discovery - PREC.** *PREC* defines the percentage of entities (methods and attributes) from the software system that were correctly (in comparison with the current design of the system) discovered in the partition reported by a clustering algorithm.

The evaluation of the results obtained by applying the above defined algorithms on JHotDraw case study are made using the following characteristics:

- *ACC* measure that has to be maximized;
- *PREC* measure that has to maximized;
- the running time of the algorithm that has to minimized.

All these algorithms provide better results than the approaches existing in the literature in the field of refactoring. Table 1 gives the comparative results.

A graphical representation of the results illustrated in Table 1 is given in Figure 1.

Based on the results presented in Table 1 and Figure 1, we can conclude that *PARS* algorithm provides the best results.

4. Case Study

As shown in Section 3, from the analyzed clustering algorithms for identifying refactorings, *PARS* algorithm provides the best results. That is why

Algorithm	ACC	PREC	Running time (min.)
kRED	**0.9829**	**0.9966**	**5**
HAC	**0.9899**	**0.9945**	**6**
PAMRED	**0.9939**	**0.9994**	**1.5**
HARED	**0.974**	**0.9978**	**3.5**
HARS	**0.974**	**0.9978**	**3.68**
PARS	***1***	***1***	***1.5***

TABLE 1. Comparative results.

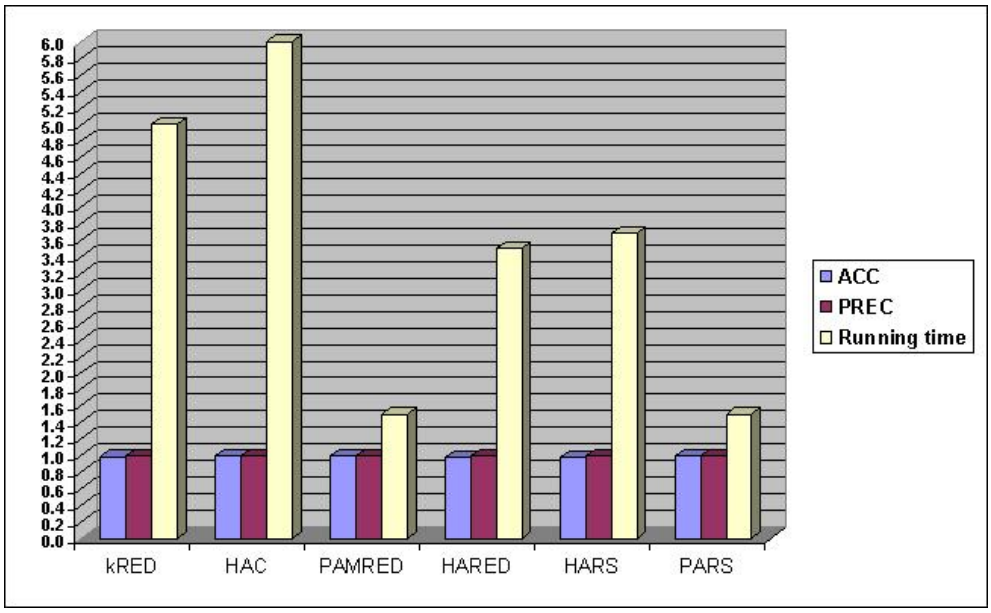

FIGURE 1. The comparative results.

in this section we present a real software system as a case study for evaluating *PARS* algorithm. It is DICOM (*Digital Imaging and Communications in Medicine*) [5] and HL7 (*Health Level 7*) [9] compliant PACS (*Picture Archiving and Communications System*) system, facilitating the medical images management, offering quick access to radiological images, and making the diagnosing process easier.

The analyzed application is a large distributed system, consisting of several subsystems in form of stand-alone and web-based applications. We have applied *PARS* algorithm on one of the subsystems from this application.

The analyzed subsystem is a stand-alone Java application used by physicians in order to interpret radiological images. The application fetch clinical images from an image server (using DICOM protocol), display them, and offer various tools to manage radiological images.

Even if the application is currently used, it also continuously evolves in order to satisfy change requirements and to provide better user experience based on feedback. That is why, the developers are often faced with the need of structural and conceptual changes.

The analyzed application consists of **1015** classes, **8639** methods and **4457** attributes.

After applying *PARS* algorithm, a total of 90 refactorings have been suggested: 10 *Move Attribute* refactorings, 78 *Move Method* refactorings, and 2 *Inline Class* refactoring.

The obtained results have been analyzed by the developers of the application and the following conclusions were made:

- 28.8% from the refactorings identified by *PARS* were accepted by the developers as useful in order to improve the system.
- 21.1% from the refactorings were acceptable for the developers, but they concluded that these refactorings are not necessary in the current stage of the project.
- 50.1% from the refactorings were strongly rejected by the developers.

Analyzing the obtained results, based on the feedback provided by the developers, we have concluded the following:

- *PARS* successfully identified smart GUI anti-patterns (parts of software were the presentation layer contains bussiness logic), misplaced constants (constants used only on a subtree of a class hierarchy, but defined in some base class). These kind of weaknesses can be discovered only if the developer manually inspects all the classes, or if a bug (related to the misplaced bussiness logic) arises. That is why automatic detection by *PARS* of these kind of weaknesses can prevent system failure or other kind of bugs and also save a lot of manual work.
- A large number of miss-identified refactorings are due to technical issues: the use of Java anonymous inner classes, introspection, the use of dynamic proxies. These kind of technical aspects appear frequently in projects developed in Java. In order to correctly deal with these aspects, we have to improve only the **Data collection** step of our approach, without modifying the *PARS* algorithm.
- Another cause of miss-identified refactorings is due to the fact that the *distance* used for discriminating entities in the clustering process take into account only two aspects of a good design: *low coupling* and

high cohesion. It would be also important to consider other principles related to an improved design, like: *Single Responsibility Principle*, *Open-Closed Principle*, *Interface Segregation Principle*, *Common Closure Principle* [4], etc.

- Our approach is currently implemented as a stand-alone application: the user provides the .jar files containing the classes of the analyzed software system and our application displays the suggested refactorings. The developers have suggested that it would be preferable to integrate our tool existing IDE (as a plugin), instead of a stand-alone application.

5. Conclusions and Future Work

We have presented in this paper a comparative analysis of several clustering algorithms that we have previously developed, algorithms which can be used in order to recondition the class structure of a software system. As a conclusion, the advantages of our approach for determining refactorings using clustering are:

- it can deal with various types of refactorings;
- it can be applied for large software systems;
- it can offer support to software developers for identifying ill-structured software modules.

Further work can be done in the following directions:

- To study the applicability of other learning techniques in order to improve software systems design.
- To use other search based approaches in order to determine refactorings that improve the design of a software system.
- To develop a tool (as a plugin for Eclipse) that is based on *CARD*.
- To apply our approach in order to transform non object-oriented software into object-oriented systems.

References

[1] William J. Brown, Raphael C. Malveau, III Hays W. McCormick, and Thomas J. Mowbray, *Antipatterns: refactoring software, architectures, and projects in crisis*, John Wiley & Sons, Inc., New York, NY, USA, 1998.

[2] I.G. Czibula and Gabriela Serban, *A Hierarchical Clustering Algorithm for Software Systems Design Improvement*, KEPT 2007: Proccedings of the First International Conference on Knowledge Engineering: Principles and Techniques, 2007, pp. 316–323.

[3] Istvan G. Czibula and Gabriela Serban, *Improving Systems Design using a Clustering Approach*, IJCSNS International Journal of Computer Science and Network Security **6** (2006), no. 12, 40–49.

[4] Tom DeMarco, *Structured analysis and system specification*, Addison-Wesley Longman Publishing Co., Inc., Prentice Hall, 1979.
[5] *Digital Imaging and COmmunications in Medicine*. at Web: http://medical.nema.org/.
[6] M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
[7] E. Gamma, *JHotDraw Project*. http://sourceforge.net/projects/jhotdraw.
[8] Jiawei Han, *Data mining: Concepts and techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
[9] *Health Level 7*. at Web: www.hl7.org/.
[10] Chung-Horng Lung, *Software Architecture Recovery and Restructuring through Clustering Techniques*, ISAW '98: Proceedings of the Third International Workshop on Software Architecture, 1998, pp. 101–104.
[11] Gabriela Serban and Istvan G. Czibula, *A New Clustering Approach for Systems Designs Improvement*, SETP-07: Proceedings of the International Conference on Software Engineering Theory and Practice, 2007, pp. 47–54.
[12] Istvan G. Czibula and Gabriela Serban, *Hierarchical Clustering for Software Systems Restructuring*, INFOCOMP Journal of Computer Science, Brasil (2007), to be published.
[13] Gabriela Serban and Istvan G. Czibula, *Restructuring software systems using clustering*, ISCIS 2007: Proceedings of the 22nd International Symposium on Computer and Information Sciences, 2007, pp. to be published.
[14] Istvan G. Czibula and Gabriela Serban, *Software systems design improvement using hierarchical clustering*, SERP 2007: Proceedings of SERP '07, 2007, pp. 229–235.
[15] Xia Xu, Chung-Horng Lung, Marzia Zaman, and Anand Srinivasan, *Program Restructuring through Clustering Techniques*, SCAM '04: Proceedings of the Source Code Analysis and Manipulation, Fourth IEEE International Workshop, 2004, pp. 75–84.

Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
E-mail address: istvanc@cs.ubbcluj.ro

Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro