computers

Article

Extract Class Refactoring Based on Cohesion and Coupling: A Greedy Approach

Musaad Alzahrani

Department of Computer Science, Albaha University, Albaha 65799, Saudi Arabia; malzahr@bu.edu.sa

Abstract: A large class with many responsibilities is a design flaw that commonly occurs in real-world object-oriented systems during their lifespan. Such a class tends to be more difficult to comprehend, test, and change. Extract class refactoring (ECR) is the technique that is used to address this design flaw by trying to extract a set of smaller classes with better quality from the large class. Unfortunately, ECR is a costly process that takes great time and effort when it is conducted completely by hand. Thus, many approaches have been introduced in the literature that tried to automatically suggest the best of classes that can be extracted from a large class. However, most of these approaches focus on improving the cohesion of the extracted classes yet neglect the coupling between them which can lead to the extraction of highly coupled classes. Therefore, this paper proposes a novel approach that considers the combination of the cohesion and coupling to identify the set of classes that can be extracted from a large class. The proposed approach was empirically evaluated based on real-world Blobs taken from two open-source object-oriented systems. The results of the empirical evaluation revealed that the proposed approach is potentially useful and leads to improvement in the overall quality.

Keywords: software quality; software refactoring; software metrics

Citation: Alzahrani, M. Extract Class Refactoring Based on Cohesion and Coupling: A Greedy Approach. *Computers* **2022**, *11*, 123. https://doi.org/10.3390/computers11080123

Academic Editor: Dae-Kyoo Kim

Received: 24 July 2022
Accepted: 12 August 2022
Published: 16 August 2022

1. Introduction

Cohesion and coupling are two of the most important aspects of software quality. Many empirical studies in the literature have proven the influence of the cohesion and coupling on software complexity, maintainability, and testability [1–5]. Highly cohesive and loose coupled software units take less effort to change and test because cohesive units should ideally have a single responsibility and one reason to change and loosely coupled units should ideally have very limited change propagation. Generally, cohesion refers to the degree of internal dependency between the elements of a software unit whereas coupling refers to the degree of external dependency between the elements of two software units. In object-oriented systems, software practitioners put tremendous effort into modularizing the systems into a set of classes that have high cohesion and low coupling. Unfortunately, real-world systems have to undergo cycles of maintenance and evaluation during their lifespan in order to meet the changing business domain requirements [6]. Consequently, the design quality of the systems starts to decline gradually and the systems become very rigid and fragile to changes [7].

Software refactoring is a preventive solution that aims to improve the design quality to make the systems easier to change. It is the process of changing the internal structures of the systems while preserving their external behaviors [8]. Martin in [8] described a wide range of refactoring techniques that can be applied to tackle specific design flaws (also known as bad smells). One of the design flaws that commonly occurs during the maintenance and evaluation activities in object-oriented systems is known as "God Class" or "Blob" which refers to a large class that has low cohesion and many methods. Extract class refactoring (ECR) is the refactoring technique used to address God Class design flaw.

It refers to the activities of partitioning a large class with many responsibilities into smaller classes such that each class has one responsibility [8].

Refactoring is a time- and effort-consuming process [9]. Therefore, researchers have striven hard to introduce approaches that aim to automate the implementation of different kinds of refactoring techniques including ECR (e.g., [10–13]). Unfortunately, most of the previously proposed ECR approaches focus on cohesion as a quality indicator to evaluate and choose the possible classes that can be extracted from the Blob in question. Considering only the cohesion of the extracted classes and neglecting the coupling between them can result in classes that are highly cohesive yet tightly coupled because increasing the cohesion of the extracted class can lead to increasing the coupling between them. Therefore, ECR can be thought of as an optimization problem that can be solved by finding an optimal balance between the cohesion and coupling of the extracted classes [14].

This paper proposes a novel approach that considers the cohesion and coupling when performing ECR. The proposed approach evaluates the possible solutions of ECR based on the combination of the cohesion and coupling for the set of classes that can be extracted from a given Blob. The purpose of the proposed approach is to identify the set of extracted classes with the best overall quality. The contribution of this paper can be summarized as follows:

- Defining a combined measure of the cohesion and coupling for a set of classes. The goal of this measure is to evaluate the overall quality (in terms of cohesion and coupling) of a set of classes that can be extracted from a Blob when conducting ECR.
- Introducing a set of algorithms that automatically suggest the set of classes that can be extracted from a given Blob.
- Conducting an empirical evaluation of the proposed approach based on Blobs taken from two real-world systems.

The rest of the paper is organized as the following. Section 2 presents a summary and discussion of a set of related studies. Section 3 presents in details the proposed approach. Section 4 presents the empirical evaluation of the proposed approach. Section 5 gives the conclusion of the study and future work.

2. Related Work

There is a common belief among the research community that software refactoring is beneficial and leads to improvement in software quality. Therefore, software refactoring has been the focus of a great deal of research, which has resulted in the development of many potential approaches to assist and automate the process. The following sections discuss the impact of software refactoring on software quality and provide a review and summary of several methods that are related to the proposed approach.

2.1. The Impact of Software Refactoring on Software Quality

Software maintenance is the longest and most expensive phase in the software development life cycle [15]. Many inevitable changes occur to internal structures of software systems during the maintenance phase which usually lead to negative impacts on the internal quality of the systems [16]. Software refactoring is believed to be the solution that addresses the degradation of software quality resulting from the changes that occurred during the maintenance activities. Conceptually, software refactoring improves software quality. In practice, however, software refactoring can have a positive and negative impact on software quality [17,18]. Therefore, many studies in the literature have investigated the influence of refactoring activities on software quality (e.g., see [15,17,19–22]).

cKaur and Kaur [19,20] conducted an experiment on an open source Java library to investigate the impact of code refactoring on the software quality. Using the Eclipse plugin JDeodorant, they identified two bad smells in the considered Java library. Before applying the suitable refactoring techniques to remove the identified bad smells, they measured the complexity of the library. Then they measured it again after applying the refactoring techniques. The results of the measurement showed that the complexity decreased after the application of the refactoring techniques.

The authors of [21] examined different variations of the Hill Climb algorithm with aim of identifying an optimum sequence of refactoring techniques that leads to the best improvement in the maintainability. They conducted their experiments on a Java system that needed several types of refactoring techniques. The optimum sequence of refactoring techniques that led to the highest increase in the maintainability value of the system was identified by the Steepest-Ascent Hill-Climbing algorithm.

Shatnawi and Wei conducted [17] an empirical study on two open source systems to examine the effect of several types of refactoring activities on four software external quality attributes namely: reusability, flexibility, extendibility, and effectiveness. They adopted the model in [23] to indicate the values of the considered external quality attributes using several object-oriented metrics such as coupling, cohesion, and inheritance. The findings of the study showed that most of the refactoring techniques led to improvement in the considered quality attributes. However, some refactoring activities had a negative impact on software quality. Similar findings were identified in a recent study [18]. Therefore, the maintenance team who is responsible for carrying our refactoring operations on a system should use a multi-attribute model to assess the impact of code refactoring on the overall quality of the system.

Al Dallal in [24] constructed a set of prediction models that can automatically identify Extract Subclass Refactoring opportunities. He empirically evaluated the constructed models based on classes selected from six open-source Java systems. The results showed that identified Extract Subclass Refactoring opportunities led to improvement in the cohesion, coupling, and size of the selected classes.

Researchers in [25] conducted an empirical study to examine the refactoring techniques that will likely cause faults. They used the refactoring tool Ref-Finder [26] to automatically identify different types of refactoring opportunities in three open source Java systems. Then they applied the SZZ algorithm [27] to identify whether changes in the source code resulting from performing the refactoring opportunities detected by the tool Ref-Finder would induce faults in the considered systems. The results indicated that most of the refactoring types were safe to apply. However, hierarchy refactoring operations such as Pull Attribute and Pull Method were more likely to cause bugs in the systems. Therefore, these kinds of refactoring can have a negative impact on the software quality unless a thorough testing and code inspection are performed after applying them.

Canfora et al. [28] experimentally studied the impact of the refactoring on the software complexity. They examined the refactoring-related changes that occurred during a specific interval in the lifespan of four open-source software systems. The results showed that the number of files that needed to be changed during maintenance activities in the considered systems usually decreased after performing refactoring operations which lead to lower complexity of the systems.

Gatrell and Counsell [29] studied the effect of refactoring operations on software maintainability. They selected 7489 classes from a large commercial system implemented in C# and studied their change history over a period of 12 months that was divided into intervals of 3 or 4 months. The researchers then identified the refactoring operations that occurred in the selected classes during the middle intervals and investigated their impact on the change and fault frequency of the classes after the middle intervals. The results indicated that classes that underwent some refactoring activities were less change and fault prone.

Researchers in [30] qualitatively analyzed the impact of refactoring operations on Windows 7 over a period of 3 months. They identified the refactoring operations by mining the history log of windows 7. The findings showed the most frequently refactored modules during the development of Windows 7 were the modules that had a high rate of test coverage, complexity, and inter-module dependencies. The complexities and inter-module dependencies of the refactored modules slightly decreased compared to other modules. However, the size of the modules increased after performing the refactoring. Therefore, the researchers suggested that software project managers and developers should conduct a multi-aspect assessment to better observe the impact of refactoring operations on software quality.

2.2. Relevant ECR Techniques

In an early study in the field of software refactoring [31], researchers developed a visualization tool that can recognize the parts of a system that might need to undergo one of four types of refactoring techniques namely: Extract Class, Move Method, Move Attribute, and Inline Class refactoring. The refactoring changes in the system under consideration can be identified with the assistance of this tool due to the use of structural dependency metrics. Marinescu [32] developed a technique that he named the detection strategy to assist software developers in discovering software modules (e.g., classes) that have a specific problem in their design, such as a class that is responsible for a large number of tasks. The strategy recommends writing metrics-based rules that are able to immediately identify software modules that are influenced by a deficiency in the design. There are four stages involved in the process of developing metrics-based rules to find errors in the design. Firstly, the manifestations of the design defects are identified. For instance, God Class is characterized by a high level of class complexity, a poor level of class coherence, and dependency on data from other classes. Secondly, a metric that is suitable for predicting each symptom is selected. For example, LCOM2 [33] is the measurement that is used to measure cohesion. Thirdly, a filtering technique, such as a less-than filter, is used for each metric to identify the symptom. For example, a value of LCOM2 greater than 10 is deemed to indicate poor class cohesiveness. In the last stage of the process, you will need to correlate the symptoms by using AND/OR operators. An opportunity for refactoring will present itself as soon as a defect in the design has been identified. A similar study [34] presented a metric-based detection approach for a collection of bad smells, one of which was the God Class. In order to identify God Classes, the authors relied on size and cohesiveness measures. They offered four different kinds of refactoring procedures, one of which was ECR, to eliminate the design smells caused by large classes.

A technique for the ECR was suggested by Fokaefs et al. [10], and it makes use of an agglomerative clustering algorithm that is determined by the Jaccard index between the methods of the class that is going to be refactored. Comparing two class methods using the Jaccard index takes benefit of their structural similarities. There is a larger potential that two methods would be included in the same cluster if their Jaccard index is higher. The clusters produced as a consequence show the possible classes that can be extracted from the class in question. In a similar manner, an extension tool of the JDeodorant Eclipse plugin that can discover chances for ECR in God Classes was developed by Fokaefs et al. [35]. A software developer can choose one of the opportunities for refactoring from those the tool offers, and the tool will then implement it automatically. The tool uses structural dependency metrics to generate classes that have greater cohesiveness than the class that will be refactored.

The approach introduced by Bavota et al. [11] separates the class to be refactored into two classes that have greater cohesion than the original class. The approach exploits the semantic and structural dependencies that exist between the methods of the class. The class that has to be refactored is first represented as a weighted graph, with nodes representing individual methods and edges representing the degree to which those methods structurally and semantically depend on each other. The Max Flow-Min Cut technique [36] is then used in this approach to divide the weighted graph into two weighted subgraphs that are representative of the two classes that can be extracted from the class to be refactored. In addition, Bavota et al. [12] presented a different approach that can automatically break down the class that will be refactored into two or more classes. Similar to their previous work, the class under consideration is modeled as a weighted graph. However, they utilize in this work a two-step clustering algorithm instead of the Max Flow-Min Cut. This algorithm separates the edges of the graph with light weights to partition the graph into a collection of subgraphs that reflect the classes that can be extracted from the original class. A similar method to automate the refactoring was proposed by the authors of [37]. This technique can automatically pin down three different refactoring possibilities: Move Method Refactoring, Move Field Refactoring, and ECR. These opportunities can be utilized to improve the quality of the design.

An ECR technique was presented by Akash et al. [38] that makes use of topic modeling. In this approach, each method included within a class is considered to be a document, and a topic distribution is produced for each class by making use of the Latent Dirichlet Allocation model [39]. Then the cosine similarity between the topic distributions of the two methods is utilized to determine their semantic similarity. However, a drawback of this technique is that to train the topic model, there has to be a significant volume of textual data. In most cases, the textual data of a class (such as identifiers and comments) will not be sufficient to properly train the topic model. As a result, the learned topic distribution may not be enough to adequately represent the semantics of a method included in a class.

Different from most of the studies in the literature, researchers in [13,40,41] proposed to exploit the clients of the class to be refactored to identify ECR opportunities. The clients of the class were used to measure the dependency between the methods of the class such that methods that were used by common clients were considered to have a dependency on each other. Then each set of methods that had strong client-based dependency were suggested to be extracted into a separate class.

The broad research on ECR is a valuable resource. However, it is evident that the majority of previous research only focuses on class cohesion when considering ECR. The main drawback observed here is improving only the cohesion of the extracted classes can lead to a decline in their overall quality because increasing the cohesion often comes at the price of the coupling. This paper, however, introduces a new approach that considers both the cohesion and coupling to identify better ECR opportunities.

3. The Proposed Approach

This section presents a set of mathematical definitions and notations that are used in the proposed approach. In addition, it presents the proposed algorithms for ECR and gives a worked example to demonstrate how to apply the presented approach on a given class.

3.1. Mathematical Definitions and Notations

Definition 1 (Methods)**.** *A class c contains of a set of methods that define the behaviour of the class and it is formally given as:* $M(c) = \{m_1, m_2, \ldots, m_n\}$ *where n is the number of methods in the class c.*

Definition 2 (Attributes)**.** *A class c contains a set of attributes that define the state of the class and that are shared by the methods of the class:* $A(c) = \{a_1, a_2, \ldots, a_k\}$ *where k is the number of attributes in the class c.*

Definition 3 (Dependency between two methods)**.** *Similar to [12,42], the dependency between two methods is defined as the number of attributes that are used in common by the two methods divided by the total number of attributes used by the two methods. It is formally given by the following equation:*

$$d(m_i, m_j) = \begin{cases} \frac{|U_i \cap U_j|}{|U_i \cup U_j|} & \textit{iff } |U_i \cup U_j| > 0, \\ 0 & \textit{Otherwise;} \end{cases} \tag{1}$$

where U_i and U_j are the set of attributes used by method m_i and m_j, respectively.

The value of the dependency between two methods ranges from 0 to 1 where a value of 0 means the two methods have no dependency (i.e., they do not use any attribute in common) and a value of 1 means the two methods have full dependency (i.e., the two methods use the same set of attributes).

Definition 4 (Cohesion)**.** *As suggested in [43,44], the cohesion of a class is defined as the total summation of the dependencies between each pair of methods in the class divided by the total number of pairs of methods in the class. The cohesion is formally defined as follows:*

$$cohesion(c) = \begin{cases} \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d(m_i,m_j)}{\frac{n\times(n-1)}{2}} & \textit{iff } n > 1, \\ 0 & \textit{Otherwise;} \end{cases} = \begin{cases} \frac{2\times\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d(m_i,m_j)}{n^2-n} & \textit{iff } n > 1, \\ 0 & \textit{Otherwise;} \end{cases} \tag{2}$$

where $\{m_i,\ m_j\} \in$ *M(c) and n is the number of methods in the class c.*

The range of $cohesion(c)$ is within the closed interval [0,1]. The larger the value of $cohesion(c)$, the higher the cohesion and better the quality of the class and vice versa.

Definition 5 (Coupling)**.** *The coupling between two classes is defined as the summation of the total dependencies between the methods of the first class and the methods of the second class divided by the multiplication between the number of methods in the first class and the number of methods in the second class. It is formally given as the following:*

$$coupling(c_1, c_2) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{k} d(x_i, y_j)}{n \times k} \tag{3}$$

where $x_i \in M(c_1)$ *and* $y_j \in M(c_2)$*; and n and k are the number of methods in the class* c_1 *and* c_2*, respectively.*

The value of $coupling(c_1,\ c_2)$ falls within the closed interval [0,1]. The smaller the value of $coupling(c_1,\ c_2)$, the lower the coupling between the two classes and the better the quality of the classes and vice versa.

Definition 6 (Combined value for the cohesion and coupling for a set of classes)**.** *The combined value for the cohesion and coupling for a set of classes is defined as the value of the average cohesion of the extracted classes minus the average coupling between the extracted classes. It is formally defined as follows:*

$$\beta(S) = \begin{cases} \frac{\sum_{i=1}^{p} cohesion(c_i)}{p} - \frac{2\times\sum_{i=1}^{p-1}\sum_{j=i+1}^{p} coupling(c_i,c_j)}{p^2-p} & \textit{iff } p > 1, \\ cohesion(c_i) & \textit{Otherwise;} \end{cases} \tag{4}$$

where $S = \{c_1, c_2, \ldots c_p\}$ *is the set of the extracted classes and p is the number of classes in the set.*

The value of $\beta(S)$ is used to evaluate the overall quality (in terms of cohesion and coupling) of the extracted classes suggested by the proposed approach. $\beta(S)$ ranges from -1 to +1. The larger the value of $\beta(S)$, the better the overall quality of the extracted classes because a large value of it indicates that the extracted classes have high cohesion and low coupling between them which is the main goal of performing ECR. On the other hand, a small value of $\beta(S)$ indicates lower quality of the extracted classes because it means either the extracted classes have low cohesion or have high coupling between them. Smaller values of $\beta(S)$ can result from performing ECR on a class that is highly cohesive (i.e., the methods of the class have high dependencies between each other) because the extracted classes may have higher cohesion than the original class, but they will have high coupling between them. In such a case, it might be better to leave the class as is without performing ECR. In addition, a small value of $\beta(S)$ can occur when performing ECR on a class that has very low or no dependencies between most of its methods because the extracted classes will have low cohesion. For such a class, other types of refactoring might be more suitable than ECR. The value of $\beta(S)$ is used in the proposed approach to choose the set of classes that can be extracted from the original class.

3.2. The ECR Algorithms

The goal of ECR is to extract classes from the original class that have high cohesion and low coupling between each other. Therefore, ECR can be considered as an optimization problem in which the aim is to maximize the cohesion of the extracted classes while minimizing the coupling between them as much as possible. To this aim, a greedy approach is proposed to address the ECR problem. A greedy approach is an algorithmic technique that can be used to tackle optimization problems. Although this technique is not always guaranteed to find the optimal solution, it has been used to optimally solve a wide range of optimization problems such as Huffman coding and fractional knapsack problems [36]. Greedy algorithms solve a problem by selecting the best (greedy) choice at the moment that solves part of the problem and reduces its size without considering the final optimal solution of the whole problem. Figure 1 gives a process overview of the proposed ECR approach. The details of the process are better explained in Algorithm 1. The algorithm takes as an input the class to undergo ECR and outputs a set of classes suggested to be extracted from the input class. The algorithm initially takes each method in the input class and put it into a separate class and adds all resulting classes into a set S. Then the algorithm makes a greedy choice by merging the two classes in the set S that have the highest coupling, (see Algorithm 2), compared to the coupling between any other two classes in the set. This choice will reduce the size of the problem (i.e., $|S|$) by one. The algorithm repeats the previous step until the size of the problem become 2 which means there are only 2 classes in the set S. The algorithm stores the classes in the set S and their $\beta(S)$ value, see Equation (4), after each merger of two classes. The algorithm finally selects the set of the classes that have the highest $\beta(S)$ value as the suggested classes that can be extracted from the input class. To avoid the extraction of small classes that have only one method, the algorithm merges any class in the selected set that has only one method with another class with which the resulting class will have the highest $\beta(S)$ value (see Algorithm 3).

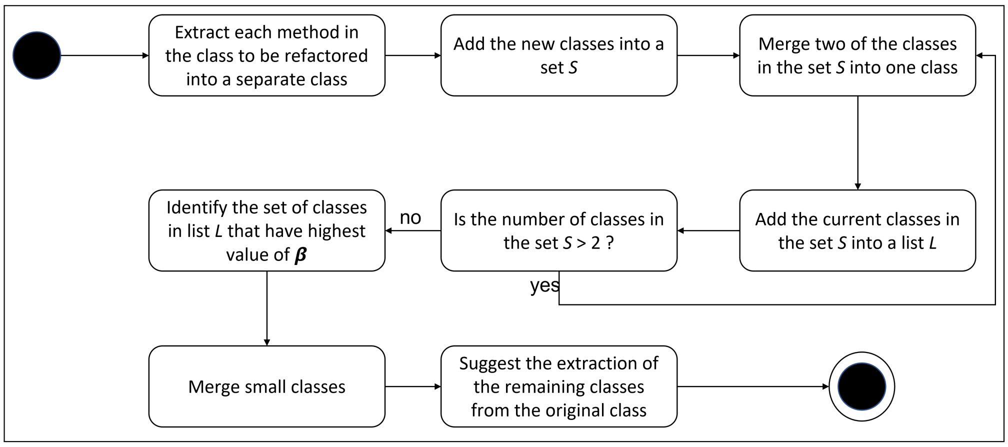
Extract each method in the class to be refactored into a separate class
Add the new classes into a set S
Merge two of the classes in the set S into one class
Add the current classes in the set S into a list L
Is the number of classes in the set S > 2 ?
no
yes
Identify the set of classes in list L that have highest value of β
Merge small classes
Suggest the extraction of the remaining classes from the original class

Figure 1. Process overview of the proposed ECR approach.

Algorithm 1: extractClassRefactoring(A)

```
Input: the class A to undergo ECR
Output: a set of suggested classes to be extracted from the input class.
1  begin
2      $S$ = { };
3      SuggestedExtractedClasses = [ ]; // a list of sets. Each set contains different
         classes that can be extracted from the input class.
4      $\beta$ = [ ]; // a list of float values. The value in $\beta$[i] is the value of
         $\beta$(SuggestedExtractedClasses[i]), see Equation (4).
5      Extract each method in the input class A into a separate class and add the class
         to the set $S$; // the number of classes in the set $S$ will be equal to the number
         of methods in the input class A.
6      while $|S| > 2$ do
7          S = mergeTwoClasses($S$); // a greedy choice, see Algorithm 2.
8          add the current classes in $S$ to the list SuggestedExtractedClasses;
9          add $\beta(S)$ to the list $\beta$; // see Equation (4).
10     end
11     max = $-\infty$;
12     j = 1;
13     for $i$ = 1 to the length of the list $\beta$ do
14         if $\beta[i] \geq max$ then
15             max = $\beta$[i];
16             j = i;
17         end
18     end
19     return mergeSmallExtractedClasses(SuggestedExtractedClasses[j]); //see
         Algorithm 3.
20 end
```

Algorithm 2: mergeTwoClasses(S)

```
Input: a set of classes.
Output: the input set after merging two classes.
1  begin
2      max = $-\infty$;
3      mergedClasses = {};
4      for each unordered pair of classes $\{c_i, c_j\} \in S$ do
5          if $coupling(c_i, c_j) \geq max$ then
6              mergedClasses = $\{c_i, c_j\}$;
7              max = $coupling(c_i, c_j)$;
8          end
9      end
10     merge the two classes in the set mergedClasses into one class R;
11     add R to $S$;
12     return $S - mergedClasses$; //remove the merged classes from $S$.
13 end
```

Algorithm 3: mergeSmallExtractedClasses(S)

```
Input: a set of classes.
Output: the input set after merging small classes.
begin
    max = $-\infty$;
    mergedClasses = {};
    for each class $c_i \in S$ such that $|M(c_i)| < 2$ do
        for each class $c_j \in S - \{c_i\}$ do
            merge the two classes $\{c_i, c_j\}$ into one temporary class t;
            if $\beta(S \cup \{t\} - \{c_i, c_j\}) \geq max$ then
                mergedClasses = $\{c_i, c_j\}$;
                max = $\beta(S \cup \{t\} - \{c_i, c_j\})$;
            end
        end
        merge the two classes in the set mergedClasses into one class R;
        add R to $S$;
        max = $-\infty$;
    end
    return $S - mergedClasses$; //remove the merged classes from $S$.
end
```

3.3. Worked Example

This section presents an example for the purpose of illustrating how the proposed ECR technique is used. In this example, the proposed technique is applied to a hypothetical class consisting of 10 methods and 12 attributes. Table 1 shows the methods-by-attributes Z matrix that models the hypothetical class where the rows of the matrix represent the methods of the class and the columns represent the attributes of the class. The value of the entry $Z[i][j]$ shows whether the method i uses the attribute j where a value of 1 means the method i uses the attribute j and a value of 0 means the method i does not use the attribute j. The dependency between each pair of methods in the class is modeled as a weighted undirected graph (Figure 2) where the nodes of the graph represent the methods of the class and the weights of the edges of the graph represent the dependencies between the methods of the class. The missing edges between nodes mean the methods represented by the nodes have 0 dependencies. The dependency between the two methods is calculated using Equation (1). For example, the weight of the edge$\{m_1, m_3\}$ is equal to 0.4, which is coming from calculating the dependency between the two methods as follows:

$$d(m_1,\ m_3) = \frac{|U_1 \cap U_3|}{|U_1 \cup U_3|} = \frac{2}{5} = 0.4$$

where U_1 and U_2 are extracted from the matrix Z:

$$U_1 = \{a_1, a_2, a_3, a_9\}$$

$$U_3 = \{a_1, a_2, a_4\}$$

Figure 3 shows colored graphs that represent the classes in the set S during the execution of the proposed ECR algorithm (see Algorithm 1). Each color represents a separate class and methods (nodes) that have the same color belong to the same class. The value of $\beta(S)$ is given under each graph. Part (A) of Figure 3 shows the classes in the set S after executing lines from 2 to 5 in Algorithm 1. As it can be seen, each method belongs to a different class (i.e., each method has a different color) after executing line 5 of the algorithm. Parts (B) to (I) in Figure 3 represent the classes in S after each time two classes in the set are merged into one class which is implemented in lines 6 to 10 of the algorithm.

Finally, and after the execution of lines 11 to 19, the algorithm suggests the set of classes represented in the colored graph depicted in part (H) of Figure 3 to be extracted from the input hypothetical class because they have the highest $\beta(S)$ value (i.e., 0.33) compared to the other sets of classes. The suggested extracted classes are the following:

$$c_1 = \{m_1, m_2, m_3, m_4\}$$

$$c_2 = \{m_5, m_6, m_7\}$$

$$c_3 = \{m_8, m_9, m_{10}\}$$

The cohesion of c_1, c_2, and c_3 are 0.31, 0.47, and 0.47, respectively, whereas the cohesion of input class is 0.17. As it can be noticed the cohesion of the extracted classes improved significantly compared to the cohesion of the input class and most importantly the $\beta(S)$ value of the set of the extracted classes is larger than β (the cohesion) of the input class. Thus, it can be claimed that the extracted classes have better overall quality than the input class.

Table 1. The methods-by-attributes matrix that represents the hypothetical class in the worked example.

	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$	$\mathbf{a_4}$	$\mathbf{a_5}$	$\mathbf{a_6}$	$\mathbf{a_7}$	$\mathbf{a_8}$	$\mathbf{a_9}$	$\mathbf{a_{10}}$	$\mathbf{a_{11}}$	$\mathbf{a_{12}}$
$\mathbf{m_1}$	1	1	1	0	0	0	0	0	1	0	0	0
$\mathbf{m_2}$	1	0	1	1	0	1	0	0	0	0	0	1
$\mathbf{m_3}$	1	1	0	1	0	0	0	0	0	0	0	0
$\mathbf{m_4}$	1	1	0	0	0	0	1	0	0	1	0	0
$\mathbf{m_5}$	0	0	0	0	1	1	0	1	0	0	0	0
$\mathbf{m_6}$	0	0	1	0	1	0	1	1	0	0	1	0
$\mathbf{m_7}$	0	0	0	0	1	0	0	1	0	0	0	0
$\mathbf{m_8}$	0	0	0	0	0	0	0	0	1	1	1	1
$\mathbf{m_9}$	0	1	0	0	0	0	0	0	1	0	1	0
$\mathbf{m_{10}}$	0	0	1	0	0	0	0	0	1	0	1	1

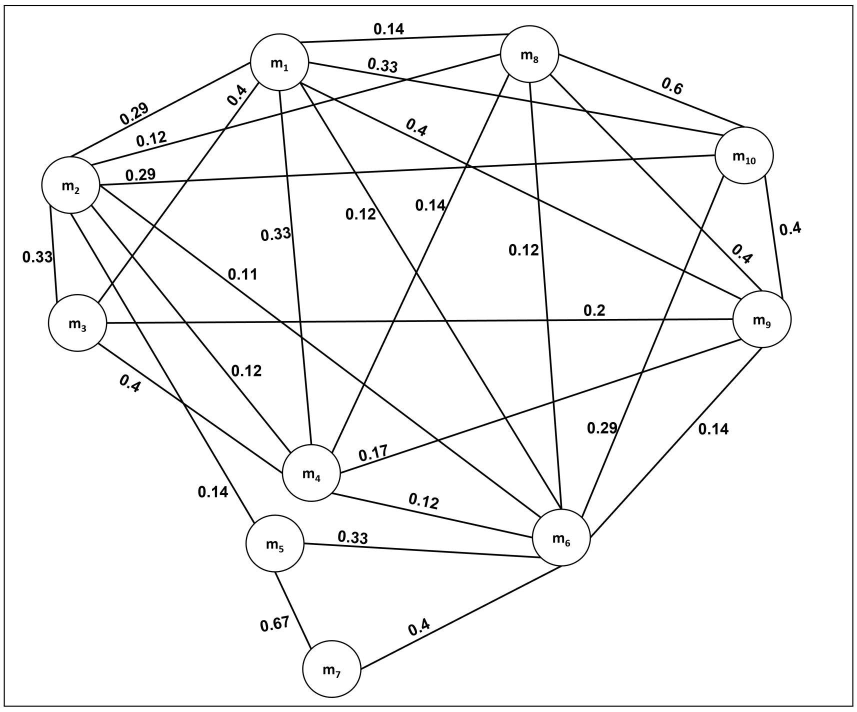

Figure 2. The graph shows the dependency between each pair of methods in the hypothetical classes used in the worked example.

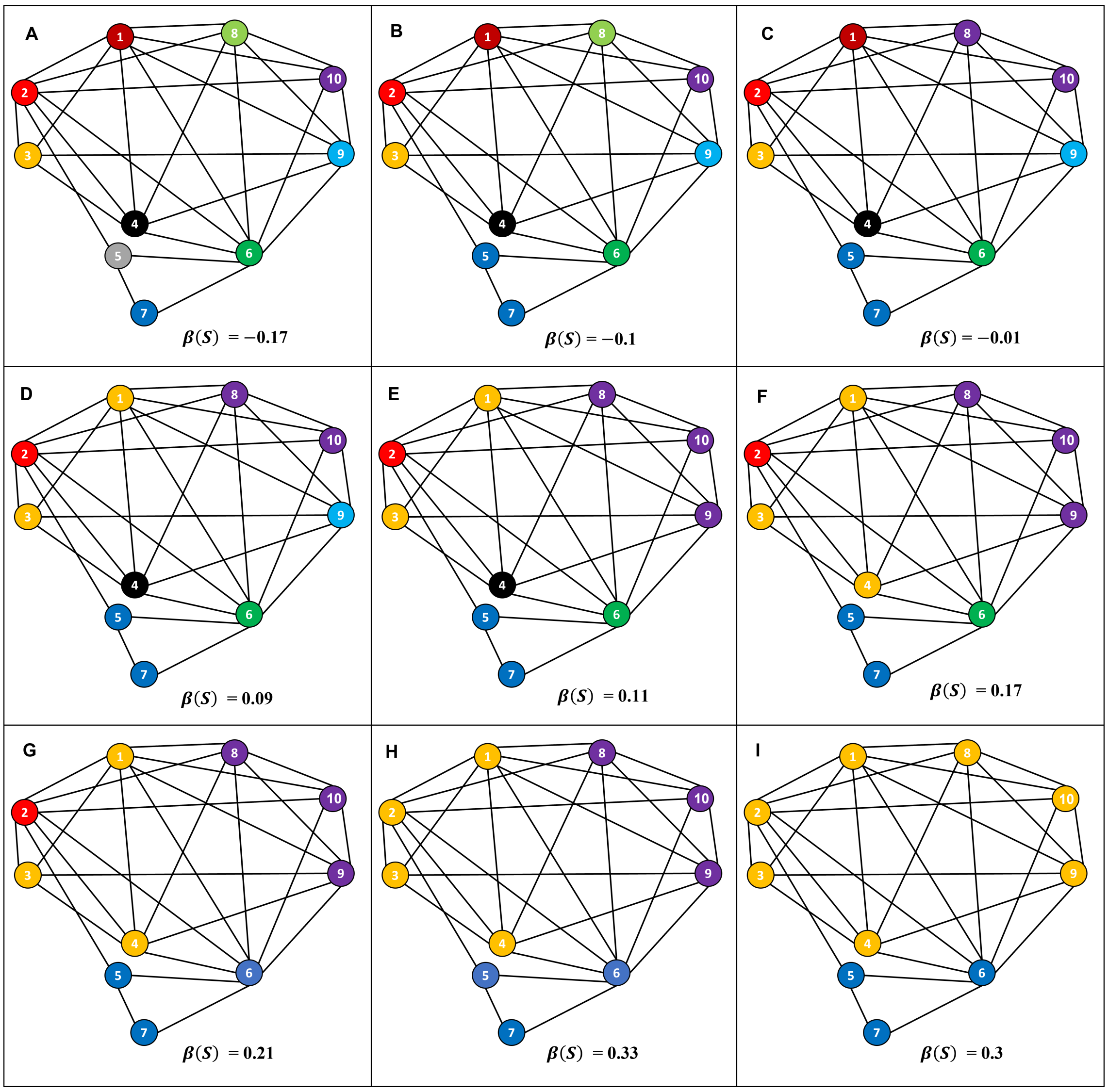

Figure 3. A colored graph that shows the classes in the set S during the implementation of the proposed ECR algorithm.

4. Empirical Evaluation

This section presents an empirical evaluation of the proposed approach based on Blobs taken from two real systems.

4.1. Research Questions and Planning

The purpose of this evaluation is to show that applying the proposed approach to real Blobs will extract from the Blobs classes with better overall quality and to show that the proposed approach (which considers both cohesion and coupling) can suggest refactoring solutions with better overall quality than the solutions suggested by considering only the cohesion. For this purpose, the following two research questions are defined:

- **RQ1.** When applied on real Blobs, does the proposed approach extract classes with better overall quality than the quality of the Blobs?
- **RQ2.** Can the refactoring solutions suggested by considering both the cohesion and coupling have better overall than the refactoring solutions suggested by considering only the cohesion?

To answer RQ1, the proposed approach is first applied on each considered Blob. Then the combined value for the cohesion and coupling for the set of classes extracted by the proposed approach (i.e., β value, see Equation (4)) is compared with the β of the Blobs where β for a Blob is equal to the cohesion of the Blob according to Equation (4) as the combined value for the cohesion and coupling for a set of classes that has only one class (which is the Blob in our case) equals to the cohesion of the class. It is expected that the proposed approach will extract classes with higher β values compared to the β values of the input Blobs.

To answer RQ2, the β value of the extracted classes suggested by the proposed approach is compared with the β value of the extracted classes suggested by a variation of the proposed approach that only considers the cohesion when choosing the set of the classes to be extracted from a Blob and when merging the small classes. Specifically, the variation of the proposed approach results from altering Algorithms 1 and 3, such that the average cohesion (see Equation (2)) of the extracted classes is used instead of the β value of the extracted classes (see Algorithms 4 and 5). Algorithm 2 is not altered in the variation approach because β is not used in the algorithm.

Algorithm 4: variationOfExtractClassRefactoring(A)//a variation of Algorithm 1. It considers the average cohesion instead of β

```
   Input: the class A to undergo ECR
   Output: a set of suggested classes to be extracted from the input class.
1  begin
2      $S$ = { };
3      SuggestedExtractedClasses = [ ]; // a list of sets. Each set contains different
         classes that can be extracted from the input class.
4      averageCohesion = [ ]; // a list of float values. The value in
         averageCohesion[i] is the value of the average cohesion for the classes in
         SuggestedExtractedClasses[i], see Equation (2).
5      Extract each method in the input class A into a separate class and add the class
         to the set $S$; // the number of classes in the set $S$ will be equal to the number
         of methods in the input class A.
6      while $|S| > 2$ do
7          S = mergeTwoClasses($S$); // a greedy choice, see Algorithm 2.
8          add the current classes in $S$ to the list SuggestedExtractedClasses;
9          add the average cohesion of the classes in $S$ to the list averageCohesion; //
             see Equation 2.
10     end
11     max = $-\infty$;
12     j = 1;
13     for $i$ = 1 to the length of the list averageCohesion do
14         if averageCohesion[i] $\geq$ max then
15             max = averageCohesion[i];
16             j = i;
17         end
18     end
19     return variationOfMergeSmallExtractedClasses(SuggestedExtractedClasses[j]);
         //see Algorithm 5.
20 end
```

Algorithm 5: variationOfMergeSmallExtractedClasses(S)//a variation of Algorithm 3. It considers the average cohesion instead of β.

```
Input: a set of classes.
Output: the input set after merging small classes.
begin
    max = $-\infty$;
    mergedClasses = {};
    for each class $c_i \in S$ such that $|M(c_i)| < 2$ do
        for each class $c_j \in S-\{c_i\}$ do
            merge the two classes $\{c_i, c_j\}$ into one temporary class t;
            if average cohesion of the classes in $(S \cup \{t\} - \{c_i, c_j\}) \geq max$ then
                mergedClasses = $\{c_i, c_j\}$;
                max = average cohesion of the classes in$(S \cup \{t\} - \{c_i, c_j\})$;
            end
        end
        merge the two classes in the set mergedClasses into one class R;
        add R to $S$;
        max = $-\infty$;
    end
    return $S-mergedClasses$; //remove the merged classes from $S$.
end
```

4.2. Considered Blobs

The considered classes (i.e., Blobs) were taken from Xerces2 (Xerces-J 2.12.0) [45] and GanttProject (ganttproject-1.10.2) [46] systems. Both systems are open-source object-oriented systems implemented in Java. Xerces2 is a library that can be used in Java systems to parse, validate, and manipulate XML elements. GanttProject is a management project tool that can be used to manage small to medium projects. The tool provides support for different project management activities such as task scheduling and resource loading. Tables 2 and 3 provide information about the used classes including the fully qualified names, number of methods, and number of lines of code (LOC). The fully qualified name of a class shows the packages to which the class belongs. The number of methods reported in Tables 2 and 3 includes the constructors, setters, and getters of the considered classes. LOC given in the two tables counts all the lines in the source code files of the classes including the comments and empty lines. The considered classes were chosen in this empirical evaluation because they have been classified as Blobs and have been used in several empirical studies in the literature (e.g., [11,12,38]).

Table 2. The considered Blobs from Xerces2.

Fully Qualified Name of the Class	Number of Methods	LOC
org.apache.xerces.dom.DeferredDocumentImpl	80	2155
org.apache.xerces.xinclude.XIncludeHandler	117	3102
org.apache.xerces.dom.CoreDocumentImpl	129	2815
org.apache.xerces.parsers.AbstractDOMParser	46	2656
org.apache.xerces.dom.DOMNormalizer	32	2101
org.apache.xml.serialize.BaseMarkupSerializer	64	1850
org.apache.xerces.jaxp.datatype.DurationImpl	48	1868
org.apache.xerces.parsers.AbstractSAXParser	52	2401

Table 3. The considered Blobs from GanttProject.

Fully Qualified Name of the Class	Number of Methods	LOC
net.sourceforge.ganttproject.gui.GanttTaskPropertiesBean	28	919
net.sourceforge.ganttproject.GanttTree	49	1848
net.sourceforge.ganttproject.GanttProject	92	2727
net.sourceforge.ganttproject.GanttGraphicArea	44	2459
net.sourceforge.ganttproject.ResourceLoadGraphicArea	30	1258

4.3. Tools

Two tools were developed to automatically apply the proposed approach and its variation on the considered Blobs. The first tool was implemented in Java based on JavaParser [47]. The Java tool takes as an input the source code file of a Blob and outputs a methods-by-attributes matrix (similar to the matrix given in Table 1) that represents the Blob. The second tool was developed using Python 3. The Python tool takes as an input the methods-by-attributes matrix resulting from the Java tool and applies the proposed approach and its variation on the matrix. The output of the Python tool is disjoint sets of methods where each set represents an extracted class from the input Blob.

4.4. Results and Discussion

The proposed approach and its variation were applied on the considered Blobs. Constructors, setters, and getters were excluded from the Blobs before the application of the proposed approach. Constructors are special methods used to instantiate objects from classes, and they were removed because they usually use all the attributes of the class which means they will have a dependency with each method that accesses an attribute in the class. Similarly, the setters and getters are special methods in the class used to change and get the values of the attributes. They were removed from the Blobs because usually each setter and getter accesses only one attribute in the class meaning that they will not have a dependency with other setters and getters in the class. In addition, methods that do not access any attribute in the class were removed because they will have no dependency with other methods in the class. These methods were removed because the proposed approach conducts the refactoring based on the cohesion and coupling which are calculated in the approach based on one kind of dependency between the methods of the class (i.e., the structural dependency resulted from accessing common attributes, see Equation (1)). The setters and getters constitute most of the removed methods. Direct and indirect access to attributes by methods were considered when identifying the set of attributes accessed by the methods of the class. A method directly accesses an attribute if the attribute appears in the body of the method (i.e., the piece of code that implements the method). On the other hand, a method indirectly accesses an attribute if the attribute appears in the body of another method that is directly or indirectly called by the method.

Tables 4 and 5 show the number considered methods (NCM) and the β value of each Blob before refactoring where the β value for one class is equal to the cohesion of the class. In addition, the two tables show the number of extracted classes (NEC), the number of methods (NM) in each extracted class, the β value, and the average cohesion (Ave. Cohesion) of the extracted classes after the application of the proposed ECR approach. For instance, results in Table 4 show that NCM in the Blob DeferredDocumentImpl is 34 and the β value of the Blob before refactoring is 0.44. The proposed approach suggests 3 classes to be extracted from the Blob DeferredDocumentImpl. The number of methods (NM) in the first, second, and third extracted classes are 27, 4, and 3, respectively. The number of extracted classes suggested by the proposed approach varies from one Blob to another. For some Blobs (e.g., XIncludeHandler), the number of suggested extracted classes is high compared to the other Blobs. The reasons behind this are that these Blobs have high NCM and low cohesion because most of their considered methods have 0 or low dependency between each other.

Answering RQ1: The results reported in Tables 4 and 5 show that the β value (which reflects the overall quality in terms of cohesion and coupling) of the extracted classes are higher than the β value of the Blobs. In addition, the average cohesion of the extracted classes is higher than the cohesion of the original Blobs. Therefore, it can be stated that the proposed approach extract classes from real Blobs with better overall quality than the quality of the Blobs.

Answering RQ2: Tables 6 and 7 show the refactoring solutions resulting from the application of the variation approach (that considers only the cohesion) on the considered Blobs. As it can be seen, the refactoring solutions in Table 6 and 7 for some Blobs are different than the refactoring solutions suggested by the proposed approach (given in Tables 4 and 5). For a case, the proposed approach suggests partitioning the Blob DeferredDocumentImpl into three classes (i.e., the extracted classes) whereas the variation approach suggests partitioning the Blob into 4 classes. Although the average cohesion of the extracted classes suggested by the variation approach is higher in the case of DeferredDocumentImpl than the average cohesion of the extracted classes suggested by the proposed approach, the β value (which indicates the overall quality in terms of cohesion and coupling) of the extracted classes suggested by the variation approach is lower than the β value of the extracted classes suggested by the proposed approach. Thus, it can be claimed that the extracted classes suggested by the proposed approach for the Blob DeferredDocumentImpl have better overall quality than the classes suggested by the variation approach. In other cases, (e.g., the case of XIncludeHandler and GanttProject) the extracted classes suggested by the proposed approach have higher average cohesion than the extracted classes suggested by the variation approach. Most importantly, when comparing all the refactoring solutions of proposed approach that are different than the refactoring solutions of the variation approach, the β values of the extracted classes suggested by the proposed approach are higher than the β values of the extracted classes suggested by the variation approach. Based on these observations, it can be stated that considering both the cohesion and coupling during ECR can extract classes with better overall quality than considering only the cohesion.

Comparing the results with literature: Tables 8 and 9 compare the results of the proposed approach with the results published in [12] based on the number NEC and the average value of LCOM2 of the extracted classes for the considered Blobs from Xerces2 and GanttProject, respectively. The reason why the results of this study are compared with the results in [12] is that the Blobs used in this study were also used in the empirical evaluation in [12]. In addition, the study in [12] is well-known in the literature and is highly cited. LCOM2 is an inverse cohesion metric that measures the lack of cohesion for a given class. It is calculated by subtracting the number of method pairs in the class that does not share any attribute from the number of method pairs that share at least one attribute. If the result of the subtraction is negative, then the value of LCOM2 is set to 0. Thus, the value of LCOM2 ranges from 0 to $+\infty$ where a smaller value of LCOM2 indicates better quality in terms of cohesion and vice versa. LCOM2 was selected as a criterion of comparison because the LCOM2 was reported for the extracted classes suggested by the approach in [12]. In addition, the metric has been used as a quality indicator in many empirical studies in the literature. The Python tool (see Section 4.3) that implements the proposed ECR approach was extended to calculate the average of LCOM2 of the extracted classes suggested by the proposed approach. The average of LCOM2 of the extracted classes suggested by the approach in [12] was calculated manually based on results reported in [12]. For example, the average of LCOM2 of the extracted classes suggested by the approach in [12] from the Blob DeferredDocumentImpl is 20.5 because the approach extracted two classes from the Blob with LCOM2 values of 0 and 41 as reported in [12]. It can be seen from the results given in Tables 8 and 9 that the proposed approach suggests to extract higher number of classes that have smaller average of LCOM2 compared to the approach in [12]. There are two main reasons behind this. First, the proposed approach considers only one type of dependency between the methods which is the structural dependency (see Equation (1)) that exists between two methods when they share common attributes. The same type of dependency is used in LCOM2 as the metric calculates the cohesion of a class based on the dependency resulting from sharing at least one common attribute between the methods of the class. On the other hand, the approach in [12] considers structural and semantic dependency between the methods of the class. Thus, the approach in [12] may suggest extracting a class with methods that are semantically dependent on each other but they do not share any attribute. Such a class would have a high value of LCOM2 and would be considered poorly cohesive when evaluated using LCOM2 as a quality indicator. The second reason is that the proposed approach excludes the setters and getters, and the methods that do not access any attributes in the class which was not excluded in the approach in [12]. LCOM2 is badly affected by these methods because they usually will not share attributes with most of the methods in the class which will lead to higher values of the metric. A further note regarding the semantic dependency, it is challenging to calculate the semantic dependency between the methods of the class automatically. It was calculated in the approach in [12] using the Latent Semantic Indexing based on the text-similarity of the methods. A major drawback of this approach is that the semantic similarity or dependency between the methods is greatly affected by the volume of the text (e.g., comments) in the methods and by the naming convention which varies from one programmer to another. This drawback may lead to poor refactoring solutions when considering the semantic dependency in ECR.

Table 4. The ECR solutions suggested by the proposed approach for the Blobs from Xerces2.

Pre-Refactoring			**Post-Refactoring**			
Class	**NCM**	**β**	**NEC**	**NM**	**β**	**Ave. Cohesion**
DeferredDocumentImpl	34	0.44	3	27, 4, 3	0.77	0.80
XIncludeHandler	77	0.1	13	15, 10, 8, 8, 5, 5, 4, 4, 4, 4, 4, 3, 3	0.5	0.55
CoreDocumentImpl	40	0.17	7	14, 8, 5, 4, 3, 3, 3	0.71	0.84
AbstractDOMParser	34	0.21	3	19, 12, 3	0.32	0.43
DOMNormalizer	16	0.16	3	5, 4, 7	0.52	0.55
BaseMarkupSerializer	50	0.24	10	8, 8, 6, 5, 4, 4, 4, 4, 4, 3	0.57	0.80
DurationImpl	22	0.33	3	13, 6, 3	0.66	0.73
AbstractSAXParser	30	0.15	6	12, 5, 4, 3, 3, 3	0.61	0.68

Table 5. The ECR solutions suggested by the proposed approach for the Blobs from GanttProject.

Pre-Refactoring			**Post-Refactoring**			
Class	**NCM**	**β**	**NEC**	**NM**	**β**	**Ave. Cohesion**
GanttTaskPropertiesBean	12	0.08	2	9, 3	0.20	0.26
GanttTree	23	0.28	4	8, 7, 5, 3	0.36	0.57
GanttProject	52	0.35	5	20, 12, 9, 8, 3	0.38	0.62
GanttGraphicArea	26	0.20	4	10, 10, 3, 3	0.36	0.49
ResourceLoadGraphicArea	17	0.17	3	10, 4, 3	0.35	0.39

Table 6. The ECR solutions suggested by the variation approach (which only considers the cohesion) for the Blobs from Xerces2.

Class	NEC	NM	β	Ave. Cohesion
DeferredDocumentImpl	4	24, 4, 3, 3	0.72	0.82
XIncludeHandler	14	15, 10, 8, 5, 5, 4, 4, 4, 4, 4, 4, 4, 3, 3	0.47	0.52
CoreDocumentImpl	8	13, 6, 4, 4, 4, 3, 3, 3	0.66	0.84
AbstractDOMParser	3	19, 12, 3	0.32	0.43
DOMNormalizer	4	5, 5, 6	0.46	0.49
BaseMarkupSerializer	10	8, 8, 6, 5, 4, 4, 4, 4, 4, 3	0.57	0.80
DurationImpl	2	19, 3	0.63	0.70
AbstractSAXParser	6	12, 5, 4, 3, 3, 3	0.61	0.68

Table 7. The ECR solutions suggested by the variation approach (which only considers the cohesion) for the Blobs from GanttProject.

Class	NEC	NM	β	Ave. Cohesion
GanttTaskPropertiesBean	2	9, 3	0.20	0.26
GanttTree	4	8, 7, 5, 3	0.36	0.57
GanttProject	6	17, 9, 9, 8, 6, 3	0.35	0.58
GanttGraphicArea	4	13, 6, 4, 3	0.36	0.53
ResourceLoadGraphicArea	3	10, 4, 3	0.35	0.39

Table 8. Comparison between the ECR solutions suggested by the proposed approach and the ECR solutions suggested by the approach given in [12] for the Blobs from Xerces2.

The Proposed Approach			The Approach Given in [12]	
Class	**NEC**	**Ave. LCOM2**	**NEC**	**Ave. LCOM2**
DeferredDocumentImpl	3	0	2	20.5
XIncludeHandler	13	2	4	223.75
CoreDocumentImpl	7	8.71	3	1218.33
AbstractDOMParser	3	0	2	0
DOMNormalizer	3	1.5	2	108
BaseMarkupSerializer	10	0	2	192.5
DurationImpl	3	1	2	283
AbstractSAXParser	6	1.67	2	24.5

Table 9. Comparison between the ECR solutions suggested by the proposed approach and the ECR solutions suggested by the approach given in [12] for the Blobs from GanttProject.

The Proposed Approach			The Approach Given in [12]	
Class	**NEC**	**Ave. LCOM2**	**NEC**	**Ave. LCOM2**
GanttTaskPropertiesBean	2	0	2	28
GanttTree	4	0.5	2	254
GanttProject	5	0	3	411
GanttGraphicArea	4	5.75	2	257.5
ResourceLoadGraphicArea	3	1.33	2	104.5

4.5. Threats to Validity

Several issues may pose threats to the validity of this empirical evaluation and limit the generality of its reported results. The first issue is that the sets of attributes accessed by methods were extracted automatically from the source code of the considered Blobs using a Java tool develop based on JavaParser [47]. These sets were not manually validated for all the methods of each Blob. However, two methods from each Blob were randomly selected, and the sets of their accessed attributes extracted by the Java tool were manually validated and they were found to be correct. In addition, the Java tool used in this study is an extension of a tool that was developed in [41] and extensively tested to automatically calculate a set of cohesion metrics from the source code of a set of Java projects consisting of a large number of classes.

The second issue that may cause a threat to the validity of this empirical evaluation is the removal of the methods that do not access any of the attributes of the Blobs. Although it might seem to be a bad design, a class can have methods that implement some features of the class without accessing the local attributes of the class such as the methods that implement user interface functionality [48]. Nevertheless, including them in this study could have affected the results of the proposed approach because the cohesion and coupling were measured in the proposed approach based on the dependency resulting from sharing common attributes by the methods of the class. Each of these methods would end up alone in a separate extracted class after the execution of Algorithm1 1. Then they would be merged with other extracted classes with which they have 0 dependencies after the execution of Algorithm 3 to avoid the extraction of small classes which would decrease the average cohesion and the overall quality of the extracted classes. This issue can be mitigated by considering conceptual (or semantic) dependency between the methods of the class (which is out of the scope of this study) besides the dependency considered in the proposed approach.

The last issue is that the refactoring solutions suggested by the proposed approach and its variation were not evaluated by software practitioners. Although the extracted classes from the considered Blobs were evaluated based on the cohesion and coupling and were shown to have better overall qualities than the original Blobs, these refactoring solutions might not be useful to some software practitioners. However, the literature has proven the importance of the cohesion and coupling and how they have an impact on other quality characteristics that are important in the software industry such as maintainability and testability. In addition, many of the refactoring approaches that were previously introduced in the literature have been evaluated using cohesion and coupling metrics.

5. Conclusions and Future Work

ECR improves the software quality, but it takes much time and effort when carried out manually. This paper introduced a novel ECR approach that automatically suggests a set of classes to be extracted from a large class. The proposed approach considers ECR as an optimization problem that should be solved by striking a balance between the cohesion and coupling of the extracted classes to avoid the extraction of classes with tight coupling or loose cohesion. Therefore, a new combined measure of the cohesion and coupling for a set of classes that can be extracted from a large class was defined in the paper for the purpose of evaluating the overall quality of the extracted classes in terms of their cohesion and coupling. The novelty of the proposed approach lies in using the combined measure of the cohesion and coupling to identify the set of the classes to be extracted from the large class in question. An empirical evaluation was conducted to assess the performance of the proposed approach based on real Blobs taken from two open-source Java systems. The findings of the empirical evaluation showed the following: (1) The proposed approach was able to extract classes from the Blobs with better overall quality than the original Blobs. (2) Considering the combined measure of the cohesion and coupling in the proposed approach is better than considering only the cohesion with respect to the overall quality of extracted classes in terms of the cohesion and coupling. This is an important finding because most of the existing ECR approaches in the literature aimed to improve only the cohesion of extracted classes without paying attention to the coupling between them. (3) The proposed approach extracted classes that have a lower average of LCOM2 compared to the results of a well-known study in the literature where lower values of LCOM2 indicate better quality.

The cohesion and coupling of the classes were measured in the proposed approach based only on the structural dependency between the methods which occurs between the methods when they share attributes. A future study can extend the proposed approach by considering the semantic (or conceptual) dependency between the methods of the classes when measuring cohesion and coupling. Methods of the class can be semantically related to each other even if they do not share attributes. The semantic dependency between the methods can be calculated using information retrieval techniques such as Latent Semantic Indexing. In addition, ECR is considered to be an optimization problem. Thus, further research should be conducted in future to try to identify optimal solutions for the problem using algorithmic techniques such as dynamic programming and linear programming.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ECR	Extract class refactoring
LCOM	Lack of cohesion of methods
LOC	umber of lines of code
NEC	number of extracted class
NCM	number considered methods
NM	number of methods

References

1. Ramasubbu, N.; Kemerer, C.F.; Hong, J. Structural complexity and programmer team strategy: An experimental test. *IEEE Trans. Softw. Eng.* **2011**, *38*, 1054–1068. [CrossRef]
2. Almugrin, S.; Albattah, W.; Melton, A. Using indirect coupling metrics to predict package maintainability and testability. *J. Syst. Softw.* **2016**, *121*, 298–310. [CrossRef]
3. Alzahrani, M.; Melton, A. Defining and validating a client-based cohesion metric for object-oriented classes. In Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 4–8 July 2017; Volume 1, pp. 91–96.
4. Alzahrani, M.; Alqithami, S.; Melton, A. Using client-based class cohesion metrics to predict class maintainability. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; Volume 1, pp. 72–80.
5. Al Dallal, J. Object-oriented class maintainability prediction using internal quality attributes. *Inf. Softw. Technol.* **2013**, *55*, 2028–2048. [CrossRef]
6. Lehman, M.M.; Ramil, J.F. Software evolution—Background, theory, practice. *Inf. Process. Lett.* **2003**, *88*, 33–44. [CrossRef]
7. Li, R.; Liang, P.; Soliman, M.; Avgeriou, P. Understanding software architecture erosion: A systematic mapping study. *J. Softw. Evol. Process.* **2022**, *34*, e2423. [CrossRef]
8. Fowler, M. *Refactoring: Improving the Design of Existing Code*; Addison-Wesley Professional: Boston, MA, USA, 2018.
9. Şanlıalp, I.; Öztürk, M.M.; Yiğit, T. Energy Efficiency Analysis of Code Refactoring Techniques for Green and Sustainable Software in Portable Devices. *Electronics* **2022**, *11*, 442. [CrossRef]
10. Fokaefs, M.; Tsantalis, N.; Chatzigeorgiou, A.; Sander, J. Decomposing object-oriented class modules using an agglomerative clustering technique. In Proceedings of the 2009 IEEE International Conference on Software Maintenance, Edmonton, AB, Canada, 20–26 September 2009; pp. 93–101.
11. Bavota, G.; De Lucia, A.; Oliveto, R. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *J. Syst. Softw.* **2011**, *84*, 397–414. [CrossRef]
12. Bavota, G.; De Lucia, A.; Marcus, A.; Oliveto, R. Automating extract class refactoring: An improved method and its evaluation. *Empir. Softw. Eng.* **2014**, *19*, 1617–1664. [CrossRef]
13. Alzahrani, M.; Alqithami, S. An External Client-Based Approach for the Extract Class Refactoring: A Theoretical Model and an Empirical Approach. *Appl. Sci.* **2020**, *10*, 6038. [CrossRef]
14. Bavota, G.; Oliveto, R.; De Lucia, A.; Antoniol, G.; Guéhéneuc, Y.G. Playing with refactoring: Identifying extract class opportunities through game theory. In Proceedings of the 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010; pp. 1–5.

15. Kaur, S.; Singh, P. How does object-oriented code refactoring influence software quality? Research landscape and challenges. *J. Syst. Softw.* **2019**, *157*, 110394. [CrossRef]
16. Mohan, M.; Greer, D. A survey of search-based refactoring for software maintenance. *J. Softw. Eng. Res. Dev.* **2018**, *6*, 1–52. [CrossRef]
17. Shatnawi, R.; Li, W. An empirical assessment of refactoring impact on software quality using a hierarchical quality model. *Int. J. Softw. Eng. Its Appl.* **2011**, *5*, 127–149.
18. Makkar, P.; Sikka, S.; Malhotra, A. A Multi-Objective Approach for Software Quality Improvement. In Proceedings of the Journal of Physics: Conference Series, Nanchang, China, 26–28 October 2021; IOP Publishing: Bristol, UK, 2021; Volume 1950, p. 012068.
19. Kaur, A.; Kaur, M. Analysis of code refactoring impact on software quality. In Proceedings of the MATEC Web of Conferences, EDP Sciences, Amsterdam, The Netherlands, 23–25 March 2016; Volume 57, p. 02012.
20. Kaur, A.; Kaur, M. Analysis of code refactoring impact on software quality: A scientific explanation. *Adv. Asp. Eng. Res.* **2021**, *7*, 43–52.
21. Tarwani, S.; Chug, A. Assessment of optimum refactoring sequence to improve the software quality of object-oriented software. *J. Inf. Optim. Sci.* **2020**, *41*, 1433–1442. [CrossRef]
22. Ivers, J.; Nord, R.L.; Ozkaya, I.; Seifried, C.; Timperley, C.S.; Kessentini, M. Industry Experiences with Large-Scale Refactoring. *arXiv* **2022**, arXiv:2202.00173.
23. Bansiya, J.; Davis, C.G. A hierarchical model for object-oriented design quality assessment. *IEEE Trans. Softw. Eng.* **2002**, *28*, 4–17. [CrossRef]
24. Al Dallal, J. Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics. *Inf. Softw. Technol.* **2012**, *54*, 1125–1141. [CrossRef]
25. Bavota, G.; De Carluccio, B.; De Lucia, A.; Di Penta, M.; Oliveto, R.; Strollo, O. When does a refactoring induce bugs? An empirical study. In Proceedings of the 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation, Riva del Garda, Italy, 23–24 September 2012; pp. 104–113.
26. Prete, K.; Rachatasumrit, N.; Sudan, N.; Kim, M. Template-based reconstruction of complex refactorings. In Proceedings of the 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010; pp. 1–10.
27. Śliwerski, J.; Zimmermann, T.; Zeller, A. When do changes induce fixes? *ACM Sigsoft Softw. Eng. Notes* **2005**, *30*, 1–5. [CrossRef]
28. Canfora, G.; Cerulo, L.; Cimitile, M.; Di Penta, M. How changes affect software entropy: An empirical study. *Empir. Softw. Eng.* **2014**, *19*, 1–38. [CrossRef]
29. Gatrell, M.; Counsell, S. The effect of refactoring on change and fault-proneness in commercial c# software. *Sci. Comput. Program.* **2015**, *102*, 44–56.
30. Kim, M.; Zimmermann, T.; Nagappan, N. An empirical study of refactoringchallenges and benefits at microsoft. *IEEE Trans. Softw. Eng.* **2014**, *40*, 633–649. [CrossRef]
31. Simon, F.; Steinbruckner, F.; Lewerentz, C. Metrics based refactoring. In Proceedings of the Proceedings Fifth European Conference on Software Maintenance and Reengineering, Lisbon, Portugal, 14–16 March 2001; pp. 30–38.
32. Marinescu, R. Detection strategies: Metrics-based rules for detecting design flaws. In Proceedings of the 20th IEEE International Conference on Software Maintenance, 2004 Proceedings, Chicago, IL, USA, 11–14 September 2004; pp. 350–359.
33. Chidamber, S.R.; Kemerer, C.F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.* **1994**, *20*, 476–493. [CrossRef]
34. Bafandeh Mayvan, B.; Rasoolzadegan, A.; Javan Jafari, A. Bad smell detection using quality metrics and refactoring opportunities. *J. Softw. Evol. Process.* **2020**, *32*, e2255. [CrossRef]
35. Fokaefs, M.; Tsantalis, N.; Stroulia, E.; Chatzigeorgiou, A. Jdeodorant: Identification and application of extract class refactorings. In Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE), Honolulu, HI, USA, 21–28 May 2011; pp. 1037–1039.
36. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2022.
37. Wang, Y.; Yu, H.; Zhu, Z.; Zhang, W.; Zhao, Y. Automatic software refactoring via weighted clustering in method-level networks. *IEEE Trans. Softw. Eng.* **2017**, *44*, 202–236. [CrossRef]
38. Akash, P.S.; Sadiq, A.Z.; Kabir, A. An Approach of Extracting God Class Exploiting Both Structural and Semantic Similarity. In Proceedings of the ENASE, Heraklion, Greece, 4–5 May 2019; pp. 427–433.
39. Blei, D.M.; Ng, A.Y.; Jordan, M.I. Latent dirichlet allocation. *J. Mach. Learn. Res.* **2003**, *3*, 993–1022.
40. Alzahrani, M. Using clients to support extract class refactoring. In *Advances in Software Engineering, Education, and e-Learning*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 695–704.
41. Alzahrani, M. Measuring class cohesion based on client similarities between method pairs: An improved approach that supports refactoring. *IEEE Access* **2020**, *8*, 227901–227914. [CrossRef]
42. Gui, G.; Scott, P.D. Coupling and cohesion measures for evaluation of component reusability. In Proceedings of the 2006 International Workshop on Mining Software Repositories, Shanghai, China, 22–23 May 2006; pp. 18–21.
43. Bonja, C.; Kidanmariam, E. Metrics for class cohesion and similarity between methods. In Proceedings of the 44th Annual Southeast Regional Conference, Melbourne, FL, USA, 10–12 March 2006; pp. 91–95.
44. Al Dallal, J.; Briand, L.C. A precise method-method interaction-based cohesion metric for object-oriented classes. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2012**, *21*, 1–34. [CrossRef]
45. Xerces2. Available online: https://xerces.apache.org/xerces2-j/ (accessed on 22 June 2022).

46. Ganttproject. Available online: https://sourceforge.net/projects/ganttproject/ (accessed on 22 June 2022).
47. JavaParser. Available online: https://javaparser.org/ (accessed on 22 June 2022).
48. Tsantalis, N.; Chatzigeorgiou, A. Identification of move method refactoring opportunities. *IEEE Trans. Softw. Eng.* **2009**, *35*, 347–367. [CrossRef]