Information and Software Technology 53 (2011) 601–614

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems

Jian Feng Cui [a], Heung Seok Chae [b,*]

[a] Department of Computer Science and Technology, Xiamen University of Technology, 600 LiGong Rd., Xiamen 361024, China
[b] Department of Science and Engineering, Pusan National University, 30 Changjeon-dong, Keumjeong-gu, Busan 609-735, South Korea

ARTICLE INFO

Article history:
Received 23 January 2010
Received in revised form 9 January 2011
Accepted 15 January 2011
Available online 23 January 2011

Keywords:
Component identification
Agglomerative hierarchical clustering algorithm
Weighting scheme
Similarity measure
Legacy systems
Software reengineering

ABSTRACT

Context: Component identification, the process of evolving legacy system into finely organized component-based software systems, is a critical part of software reengineering. Currently, many component identification approaches have been developed based on agglomerative hierarchical clustering algorithms. However, there is a lack of thorough investigation on which algorithm is appropriate for component identification.
Objective: This paper focuses on analyzing agglomerative hierarchical clustering algorithms in software reengineering, and then identifying their respective strengths and weaknesses in order to apply them effectively for future practical applications.
Method: A series of experiments were conducted for 18 clustering strategies combined according to various similarity measures, weighting schemes and linkage methods. Eleven subject systems with different application domains and source code sizes were used in the experiments. The component identification results are evaluated by the proposed size, coupling and cohesion criteria.
Results: The experimental results suggested that the employed similarity measures, weighting schemes and linkage methods can have various effects on component identification results with respect to the proposed size, coupling and cohesion criteria, so the hierarchical clustering algorithms produced quite different clustering results.
Conclusions: According to the experimental results, it can be concluded that it is difficult to produce perfectly satisfactory results for a given clustering algorithm. Nevertheless, these algorithms demonstrated varied capabilities to identify components with respect to the proposed size, coupling and cohesion criteria.

1. Introduction

Legacy systems are software systems that cannot be easily dealt with but are vital to our organizations [1]. Legacy systems continue to be used because of the prohibitive cost of replacing or redesigning them, despite their poor competitiveness and compatibility with modern equivalents [2]. The evolution of a legacy system is a complex task, which is influenced by several concerns (e.g. its decomposability, budget, technical and time constraints, etc.) [3]. A key activity in software reengineering consists of gathering the software entities comprising the system into meaningful (highly cohesive) and independent (loosely coupled) components; this is called component identification. The common process of component identification starts by parsing the source code of a legacy system, and then the source code is organized into cohesive subsystems that are loosely interconnected by a particular clustering algorithm [11]. According to [5], a component specifies a formal contract of services that it provides to its clients and those that it requires from other components or services in the system in the terms of its interfaces. In terms of object-oriented software systems, a component consists of a set of member classes and interfaces which specify their services.

Agglomerative hierarchical clustering algorithms have been applied in many existing component identification approaches, because a multi-level architectural view produced by agglomerative hierarchical clustering algorithms facilitates architectural understanding [4,7,8,10]. Understanding the behaviors of employed clustering algorithms is the first important step for meaningful utilization of clustering techniques for component identification [6]. The similarity measure and linkage method are the two most important factors in the agglomerative hierarchical clustering algorithms studied by many researchers. In object-oriented systems, the calculation of connection strength between software entities, which is also known as weighting scheme, are affected by the connection manners between software entities. Weighting

* Corresponding author. Tel.: +82 51 510 3517; fax: +82 51 515 2208.
E-mail address: hschae@pusan.ac.kr (H.S. Chae).

0950-5849/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2011.01.006

schemes can have significant effects on clustering behaviors, but most of them just focused on binary feature values [7,8,10,35]. In this paper, we classified weighting schemes as binary weighting scheme, absolute weighting scheme and relative weighting scheme. We performed a series of experiments to examine their performances.

Agglomerative hierarchical clustering algorithms have been widely employed in software clustering techniques, however, many researches merely focused on proposing an approach to solving a particular problem [16]. In this paper, we performed a study on evolving object-oriented legacy systems into component-based systems by using a variety of agglomerative hierarchical clustering methods which employed various similarity measures, linkage methods and weighting schemes. The goal of this paper is to examine whether there is a superior method in the clustering process. A series of experiments were conducted with various clustering algorithms for several object-oriented legacy systems. Based on the clustering results, we provided an evaluation of the relative strengths and weaknesses of various agglomerative hierarchical clustering algorithms by using a set of criteria with respect to the size, coupling and cohesion criteria. Our focus is on analyzing agglomerative hierarchical clustering algorithms in software reengineering, and then identifying their respective strengths and weaknesses in order to apply them effectively for future practical applications.

The rest of this paper is organized as follows. Section 2 gives an overview of agglomerative hierarchical clustering algorithms. Section 3 illustrates the similarity measures and weighting schemes used for agglomerative hierarchal clustering. Section 4 presents the experimental settings: the tool CIETool developed for automating the process of component identification and evaluation, the clustering strategies, the criteria used for evaluating the clustering results and the employed subject systems. In Section 5, we present the clustering results and evaluate them with the proposed size, coupling and cohesion criteria. Section 6 discusses related work. Finally, Section 7 summarizes our study and provides suggestions for future work.

2. An overview of agglomerative hierarchical clustering algorithms

In this section, we provide an overview of agglomerative hierarchical clustering algorithms and illustrate how they are utilized for software component identification. The agglomerative hierarchical clustering algorithms (AHCA) start from a set of individual entities that are first grouped into small clusters; these are in turn grouped into larger clusters until reaching a final all inclusive clustering [8]. One advantage of these algorithms is that they are non-supervised. They do not need any extra information such as the number of expected clusters and candidate regions of search space for locating each cluster. AHCA provides a view for software clustering; the earlier iterations present a detailed view of the software architecture and the later ones present a high-level view.

When AHCA is utilized for software component identification, the first problem that needs to be addressed is to determine the types of entities to be clustered. In this study, our aim is to reengineer an object-oriented legacy system into a component-based system. Classes are treated as the entities to be clustered, because they are the fundamental parts comprising an object-oriented software system. Interface classes and abstract classes are ignored, because they generally contribute to system structures construction rather than concrete functions realization. The inner class is also ignored, because it is nested in a regular outer class and it is naturally viewed as a member of latter. In this paper, we call the types of classes under study *normal classes*. For simplicity, all usage of the term *class* refers to the normal classes hereinafter unless otherwise indicated in this paper.

Generally a software system is composed of a set of classes and various relations. Classes can be thought of as the nodes in a graph and their relations as the edges in this graph. Fig. 1 shows an example system represented by an undirected graph.

Classes can be connected by various relations within a software system. In this study, we define two classes cls_i and cls_j to be connected if and only if at least one of the following connections exists between them:

- *attr* is an attribute of cls_i, and cls_j is the type of *attr*;
- *op* is an operation of cls_i, and cls_j is the type of a parameter of *op*, or the return type of *op*;
- *op* is an operation of cls_i, and cls_j is the type of a local variable of *op*;
- *op* is an operation of cls_i, and cls_j is the type of a parameter of an operation invoked by *op*;
- *op* is an operation of cls_i, *attr* is an attribute of cls_j, and *op* references *attr*;
- *op* is an operation of cls_i, op' is an operation of cls_j, and *op* invokes op'.

In addition, the classes can be distinguished by their invocation directions. For example, in the above statements, cls_i is the accessing class, and cls_j is the accessed class. Considering inheritance, in this study, we just consider implemented attributes and operations for classes. That is, a parent class connecting some class does not necessarily mean that its child class also connects that class.

Throughout this paper the following notations are used to describe a software system:

- Assuming that *UC* denotes the set of classes in a software system *S*, then $|UC|$ is the number of classes of *S*.
- Classes can be grouped into different components. Let *CSET(cmp)* denote the set of classes in the component *cmp*. *We suppose that* $CSET(cmp_m) \cap CSET(cmp_n) = \emptyset$, *if* $m \neq n$.
- A software system can be partitioned into a set of components. Let *UCMP* denote the set of components that compose a software system. Suppose that the system *S* is composed of *t* components, then $|UCMP| = t$.
- Let $conn(cls_i, cls_j)$ denote the set of connections between classes cls_i and cls_j, where cls_i is the accessing class and cls_j is the accessed class.
- Let $conn(cmp_i, cmp_j)$ denote the set of connections between components cmp_i and cmp_j, where cmp_i is the accessing component and cmp_j is the accessed component, which are determined by the invocation direction of the classes in the corresponding components.

Fig. 2 illustrates clustering of AHCA. AHCA accepts the set of individual classes *UC* of the software system *S* as the input. A given similarity measure is necessary to quantitatively calculate the similarity between each pair of classes. *UCMP* is the set of components. Initially, each element of *UCMP* contains an individual class of the

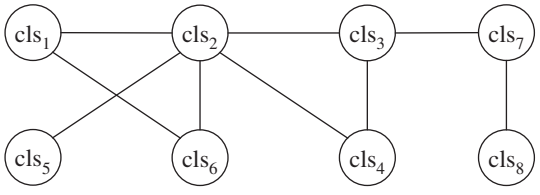

Fig. 1. An example system.

Input: *UC*

Algorithm:

1. Calculate the similarity between each pair of classes within *UC*
2. A set of components each of which contains only one class is created. Create the set of components *UCMP* each element of which contains an individual class. That is, $|UCMP|=|UC|$
3. Repeat
   - 3.1 Group the most similar components cmp_i and cmp_j into the new component cmp_m, where $CSET(cmp_m) = CSET(cmp_i) \cup CSET(cmp_j)$.
   - 3.2 Update *UCMP* by replacing the newly formed component cmp_m for cmp_i and cmp_j.
   - 3.3 Recalculate the similarity between cmp_m and all the other components.

   Until $|UCMP|=1$
4. Return *UCMP*

Fig. 2. The process of AHCA.

UC. In each iteration, the two components that are the most similar are combined into a new one. Then, *UCMP* is updated with the newly formed component. The similarity values between the newly formed and existing components are recalculated afterwards. Clustering terminates when all classes are combined into a single component.

During clustering, the similarity between the newly formed and existing components should be iteratively recalculated (Step 3.3 in Fig. 2). There are various linkage methods for this. They are described below, where cmp_i, cmp_j and cmp_n denote existing components, cmp_m is the newly formed component which combines cmp_i and cmp_j and $sim(cmp_m, cmp_n)$ denotes the similarity value between cmp_m and cmp_n. [10].

- Single Linkage:
  $sim_{Single}(cmp_n, cmp_m) = max(sim(cmp_n, cmp_i), sim(cmp_n, cmp_j))$.
- Complete Linkage:
  $sim_{Complete}(cmp_n, cmp_m) = min(sim(cmp_n, cmp_i), sim(cmp_n, cmp_j))$.
- Weighted Average Linkage:
  $sim_{WA}(cmp_n, cmp_m) = (sim(cmp_n, cmp_i) + sim(cmp_n, cmp_j))/2$.
- Unweighted Average Linkage:
  $sim_{UWA}(cmp_n, cmp_m) = (sim(cmp_n, cmp_i) * |cmp_i| + sim(cmp_n, cmp_j) * |cmp_j|)/(|cmp_i| + |cmp_j|)$.

As was mentioned earlier, AHCA can produce a hierarchical decomposition for a software system without defining the number of components in advance. The earlier iterations of the hierarchical clustering approach present a detailed view of the architecture and the later ones present a high-level view. Fig. 3 gives a hierarchical representation of the software system by utilizing AHCA. As shown in Fig. 3, more similar associated classes are combined at lower cuts and less similar associated ones are combined at higher cuts.

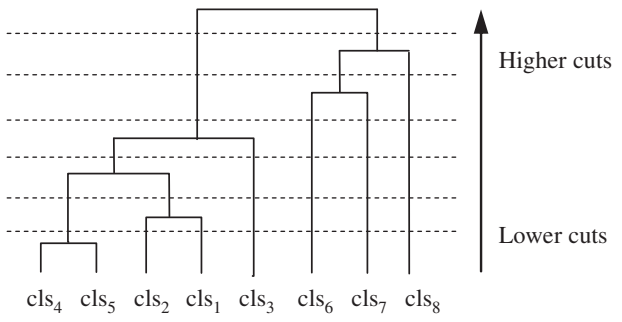

Fig. 3. The hierarchical representation of a software system.

3. Similarity measures for AHCA

Similarity measures determine how similar a pair of classes is. The similarities between classes can be quantitatively calculated by a variety of measures. According to [9], the choice of a proper similarity measure has even more influence on the result than the clustering algorithm. In addition, class connections can be weighted for finer tuning of similarity measures. In this section, we present three types of class weighting schemes, and then illustrate the similarity measures.

3.1. Weighting schemes

Weighting schemes are used to calculate the connection strength between a pair of classes. We considered three different weighting schemes in this study, the binary weighting scheme (W_B), absolute weighting scheme (W_A) and relative weighting scheme (W_R). Detailed descriptions are as follows:

- Binary weighting scheme (W_B)

The binary weighting scheme has been widely used in software clustering algorithms [7,8,10–12]. It determines the weight between a pair of classes by considering whether they have any connections: 1 denotes the existence of connections between cls_i and cls_j, and 0 denotes that there are none. A class is supposed to be connected to itself, so there is a 1 value when i and j are equal. The binary weighting scheme is defined as follows:

$$W_B(cls_i, cls_j) = \begin{cases} 1 & \text{if } |\, conn(cls_i, cls_j)\,| > 0 \\ & \text{or } |\, conn(cls_j, cls_i)\,| > 0 \text{ or } i = j \\ 0 & \text{otherwise} \end{cases}$$

- Absolute weighting (W_A)

In [12], the researchers proposed the weighted combined algorithm for software clustering, which considers the sum of features that are present in both the entities. The absolute weighting scheme is similar to the weighted combined algorithm. It considers the number of connections between a pair of classes. Take Fig. 1 as an example. Suppose that cls_1 invokes 5 out of 20 operations of cls_2, and it invokes 1 out of 2 operations of cls_6. It is plausible that cls_1 has a tighter relation with cls_6 than cls_2. We suppose that the weight is 1 when the two classes are equal. The absolute weighting scheme is then defined as follows:

$$W_A(cls_i, cls_j) = \begin{cases} 1 & \text{if } i = j \\ \frac{|conn(cls_i, cls_j)|}{|v(cls_j)|} + \frac{|conn(cls_j, cls_i)|}{|v(cls_i)|} & \text{otherwise} \end{cases}$$

where $v(cls)$ denotes the set of referenced attributes and invoked operations in *cls*. Only the invoked or referenced members contribute to the connections between classes.

- Relative weighting scheme (W_R)

The relative weighting scheme additionally considers how frequently a class is connected with others by comparison with the absolute weighting scheme. Take Fig. 1 as an example. It is plausible that cls_7 has a tighter relation with cls_8 than cls_2 because cls_8 has connections with only cls_7 while cls_2 connects with 5 out of 7 classes. The weighting scheme TF-IDF has been used in document clustering [13]. This is a statistical measure to evaluate how important a word is to a document in a collection or corpus and there is a detailed definition in [13]. The TF-IDF method was adapted to software component identification and is called the relative weighting scheme. The relative weighting scheme is defined as follows:

$$W_R(cls_i, cls_j) = \begin{cases} \log\frac{|UC|}{|f(cls_i)|} & \text{if } i = j \\ \frac{|conn(cls_i, cls_j)|}{|v(cls_j)|} \cdot \log\frac{|UC|}{|f(cls_j)|} + \frac{|conn(cls_j, cls_i)|}{|v(cls_i)|} \cdot \log\frac{|UC|}{|f(cls_i)|} & \text{otherwise} \end{cases}$$

where $f(cls)$ denotes the set of classes that connect with cls; the definition of $v(cls)$ is the same as that defined in the absolute weighting scheme.

An example of these three weighting schemes is given below. Take Fig. 1 as an example. The weights among cls_2, cls_3 and cls_5 are calculated by these three respective weighting schemes. The following values are assigned to the three classes: $|conn(cls_2, cls_5)| = 5$, $|conn(cls_5, cls_2)| = 2$, $|conn(cls_2, cls_3)| = 4$, $|conn(cls_3, cls_2)| = 7$; $|v(cls_2)| = 20$, $|v(cls_3)| = 5$, $|v(cls_5)| = 5$; $|f(cls_2)| = 5$, $|f(cls_3)| = 3$ and $|f(cls_5)| = 1$. The weights between each pair of classes are shown in Table 1.

As shown in Table 1, the weighting of the three pairs of classes differs according to the weighting scheme. The weight between cls_3 and cls_5 is 0 for all three weighting schemes, because there are no connections among them. The binary weighting scheme ensures that (cls_2, cls_3) and $(cls_2, clas_5)$ have the same weighting. When the absolute weighting scheme is employed, (cls_2, cls_3) has a greater weight than (cls_2, cls_5). However, when the relative weighting scheme is employed, $class_2$ and $class_5$ have the largest weight. The weighting schemes are deemed to play an important role in component identification approaches; this will be demonstrated in the following sections.

3.2. Similarity measures

The similarity measure is used to evaluate the similarity between a pair of classes. The quantitative computation of the similarities between classes can differ according to the measure. Therefore, various algorithms can generate more than one clustering result for the same subject software system. Generally, the similarity measures are categorized into two types: the direct link method and the feature vector-based method [10].

The relations between the classes can be weighted by the summed weights of their connections. It is plausible that the greater the number of connections between a pair of classes, the greater their similarity. This type of similarity measure is called the direct link method. Another type of measure considers the scores of the classes in terms of a number of characteristics or features. A class can be quantitatively represented by a feature vector. In this case, the classes are weighted by the features, and the scores can be determined based on a given weighting scheme. This type of similarity measure is called the feature vector-based method.

The direct link based method has an appealing simplicity. If one class calls another, they are similar to some degree. But this only considers the local dependency information between each pair of classes. The feature vector-based method takes into account more complicated dependency information between the classes. That is, the similarity between a pair of classes depends not only on how closely related they are but also on how closely related each class is with the others. A comparison of these two types of similarity measures was done in [8,14], where the authors claimed that the direct link method was inferior to the feature vector-based method in software clustering. We focus on feature vector-based methods in this study.

Table 1
Weights between classes for various weighting schemes.

Classes	W_B	W_A	W_R
(cls_2, cls_3)	1	1.15	0.42
(cls_2, cls_5)	1	1.10	0.92
(cls_3, cls_5)	0	0	0

In this subsection, first we give an introduction of feature vector in Section 3.2.1. We use an example to illustrate how an independency matrix is used to represent feature vectors. Then, in Section 3.2.2, we give the definitions of distance coefficient-based similarity measures, and provide two major distance coefficient-based similarity measures. Finally, in Section 3.2.3, we discuss the definitions and classification of association coefficient-based similarity measures.

3.2.1. Feature vector

An important issue in the feature vector-based method is to decide which features should be used for representing a class. The features can be broadly divided into non-formal features and formal features [8]. There are a fairly wide range of non-formal features, such as the file naming convention, class creation date, and developer's class name. Non-formal features do not directly affect system behaviors. The formal features have a direct impact on the system behaviors. For example, if a class changes its invocation from one class to another, we should expect changes in the system's behavior.

In [15], the authors proposed an interdependency matrix, which can be applied to formal feature vector-based method for calculating similarities between software entities; this is illustrated in Table 2. With regard to object-oriented software systems, Table 2 can be interpreted as the collection of feature values of the set of classes within a software system, where each row denotes the values of the feature vector for each class listed in the first column. The values of the feature vectors can be calculated by either binary weighting scheme, absolute weighting scheme, or relative weighting scheme. In Table 2, the feature values are calculated by the binary weighting scheme. The feature values can also be calculated by the absolute weighting scheme or the relative weighting scheme depending on the practical situation, where the exact values can be obtained by the formulations given in Section 3.1. The 1 entry denotes that a class has calling relationships with a feature, and the 0 entry denotes that it does not. For example, cls_1 can be represented by the vector $\langle 1, 1, 1, 0, 0, 0, 0, 0\rangle$, which denotes that cls_1 has calling relationships with the classes cls_1, cls_2, and cls_3.

The relationships can be considered as either indirected or directed, so the matrix can be either symmetric or asymmetric. Note that the 1 entries are used in the main diagonal; this indicates that a class is dependent on itself. As shown in Table 1, various weighting schemes will have various affects on the values of the feature vectors; this will be illustrated in more detail in Section 5.

3.2.2. Distance coefficient-based similarity measures

Distance coefficient measures treat class similarity as a geometry distance problem. The dimensional coordinates in a geometry space can be interpreted as the numerical values of a class in the feature vector. The distance coefficients are inverse similarity measures. A higher distance coefficient indicates less similarity between a pair of classes. The most popular distance measures are the Euclidean distance measure and the Camberra distance measure [10]. They are described as follows:

Table 2
Example of an interdependency matrix.

	cls_1	cls_2	cls_3	cls_4	cls_5	cls_6	cls_7	cls_8
cls_1	1	1	1	0	0	0	0	0
cls_2	1	1	0	1	0	0	0	0
cls_3	1	0	1	1	1	0	0	0
cls_4	0	1	1	1	0	1	1	0
cls_5	0	0	1	0	1	0	0	0
cls_6	0	0	0	1	0	1	0	1
cls_7	0	0	0	1	0	0	1	1
cls_8	0	0	0	0	0	1	1	1

- $dist_{Euclidean}(cls_i, cls_j) = \sqrt{\sum_{k=1}^{n} |x_k - y_k|^2}$
- $dist_{Camberra}(cls_i, cls_j) = \sum_{k=1}^{n} \frac{|x_k - y_k|}{|x_k + y_k|}$

where n represents the total number of features of the feature vector; and x_k and y_k represents the values of cls_i and cls_j on the kth feature, respectively.

3.2.3. Association coefficient-based similarity measures

Unlike the distance coefficients, the association coefficients measure not only how similar a pair of classes is, but also how dissimilar they are with the other classes in terms of the features. A higher association coefficient measure value indicates more similarity between a pair of classes. Fig. 4 gives an example of an association coefficient denoted by binary weights. Let a denote the number of features present in both cls_i and cls_j (1–1 match), b denote the number of features present in cls_i but not cls_j (1–0 match), c denote the number of features present in cls_j but not cls_i (0–1 match), and d denote the number of features not present in both cls_i and cls_j (0–0 match).

The association coefficients are defined as follows [10]:

- Simple coefficient: $sim_{Simple}(cls_i, cls_j) = (a + d)/(a + b + c + d)$

This coefficient treats 1–1 matches and 0–0 matches equally, both contribute to the similarity. Matches and mismatches are weighted equally.

- Jaccard coefficient: $sim_{Jaccard}(cls_i, cls_j) = a/(a + b + c)$

This coefficient does not take 0–0 matches into account at all. Therefore this measure is very well suited for asymmetric binary features, where 1 means that the feature is present and 0 means that it is absent.

- Sorensen–Dice coefficient: $sim_{Sorensen\text{-}Dice}(cls_i, cls_j) = 2a/(2a + b + c)$

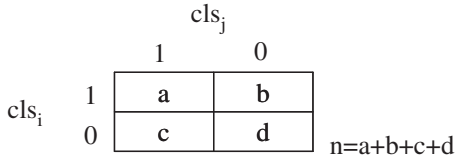

Fig. 4. Representation of the association coefficient.

This coefficient applies double weighting of 1–1 matches to the Jaccard coefficient.

In addition, association coefficients can be also applied to nonbinary features [12]. Let Ma represent the sum of feature values present in both classes. Let Mb and Mc represent the sum of feature values present in one class but not the other. Two nonbinary association coefficients are given below:

- Ellenberg measure:

$$sim_E(cls_i, cls_j) = \frac{1}{2} * Ma \Big/ \left(\frac{1}{2} * Ma + Mb + Mc\right)$$

- Unbiased Ellenberg measure:

$$sim_{UE}(cls_i, cls_j) = \frac{1}{2} * Ma \Big/ \left(\frac{1}{2} * Ma + b + c\right)$$

4. Experimental settings

The application of different agglomerative hierarchical clustering algorithms to identify components from the legacy systems can generate different results. We have performed a series of experiments on the agglomerative hierarchical clustering algorithms by employing various weighting schemes, similarity measures and linkage methods. In this section, we present the experimental settings employed in this study.

4.1. CIETool: the tool for component identification and evaluation

In order to facilitate component identification and evaluation, we have developed the tool CIETool. CIETool integrates a set of clustering algorithms and evaluation criteria to help software engineers to automatically extract class dependency information from legacy systems, perform component identification approaches, and then evaluate the results based on the provided evaluation criteria. To ensure the correctness of the experimental results, we used the open APIs provided by Borland Together [17]. Fig. 5 shows the logical architecture of CIETool. As can be seen in Fig. 5, CIETool has three major modules: the fact extraction module, component identification module and component evaluation module:

- Fact extraction

The fact extraction module parses the Java source code and extracts the class dependency information from a legacy system. It

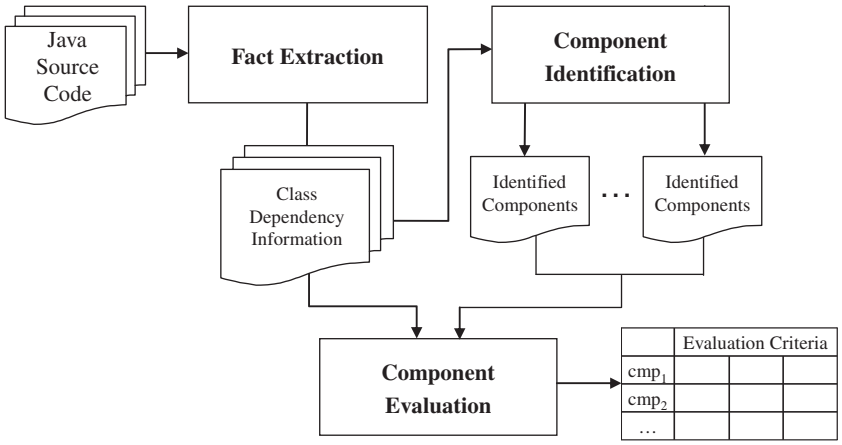

Fig. 5. Logical architecture of CIETool.

was based on the Borland Together platform [17]. Borland Together provides open Java APIs for developers in order to facilitate accesses to Java source code. The fact extraction module extracts class dependency information based on the previously defined six types of connections in Section 2.

- Component identification

The component identification module treats class dependency information as the input, executes agglomerative hierarchical clustering algorithms and produces clustering results for the subject systems. We have implemented different clustering algorithms in this module; this will be illustrated in detail in Section 4.2.

- Component evaluation

The component evaluation module accepts the class dependency information and the clustering results as the input and evaluates the quality of the identified components. The criteria used for evaluating components include size, cohesion and coupling; these are described in detail in Section 4.3.

4.2. Clustering strategies

As was illustrated in Sections 2 and 3, when AHCA is applied to component identification for legacy systems, the clustering results can be influenced by the employed similarity measures, weighting schemes and linkage methods. A thorough study on agglomerative clustering algorithms is almost impossible. In order to conduct a comparative study, we devised 18 clustering strategies, which were composed of two typical similarity measures, three typical weighting schemes and three typical linkage methods. These 18 clustering strategies have covered the major types of agglomerative hierarchical clustering algorithms. Table 3 gives the detailed configuration of the clustering strategies.

The previously presented three weighting schemes, binary weighting scheme, absolute weighting scheme and relative weighting scheme, are employed in the experiments.

Considering two major categories of similarity measures, the employed distance coefficient is the Euclidean measure, and the employed association coefficients are the Jaccard and unbiased Ellenberg measures. Limited by their characteristics, the Jaccard measure can only be employed with the binary weighting scheme, and the unbiased Ellenberg measure is employed with the absolute and relative weighting schemes. We have presented four commonly used linkage methods in Section 2.

As was stated in [8], the weighted average linkage method and the unweighted average linkage method give similar clustering results. For simplicity, we employ 3 out of 4 linkage methods: the complete, single and unweighted average linkage methods.

4.3. Evaluation criteria

In this study, several evaluation criteria are proposed to qualify the clustering results produced by various clustering strategies. The coupling and cohesion aspects of a software system are the basic quality attributes that can seriously impact the maintenance, evolution and reuse. In addition, a well organized component should also have appropriate numbers of implementation classes. So we evaluate the clustering results in terms of the size, coupling and cohesion.

Fig. 6 gives an example illustrating the evaluation criteria. As shown in Fig. 6, a system which contains eight classes is partitioned into three components: cmp_1, cmp_2 and cmp_3. Each component contains one or more classes. Only the interface is exposed to component users. Therefore, the interface operations owned by a component are also a key consideration. In our study, the interface operation is defined as the class members in the component that are invoked or referenced by any other component outside it. The interface operation is represented by a directed line between a pair of classes. For example, class cls_2 invokes the operation op_{61} of class cls_6, so op_{61} is an interface operation of component cmp_2. Similarly, op_{21} is an interface operation of cmp_1.

4.3.1. Size

In [18], the authors claimed that the ideal case consisted of several interconnected components comprising a system and possibly

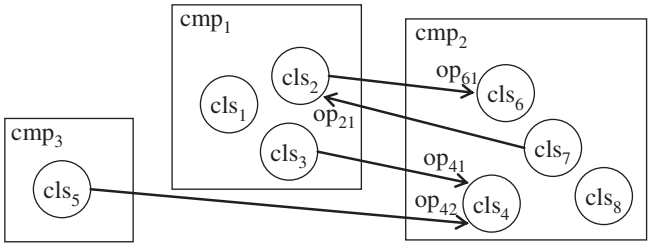

Fig. 6. An example system.

Table 3
Clustering strategies.

Category	ID	Strategy	Similarity measure	Weighting scheme	Linkage method
Distance coefficient	01	S1W1L1	Euclidean	Binary	Single
	02	S1W1L2	Euclidean	Binary	Unweighted average
	03	S1W1L3	Euclidean	Binary	Complete
	04	S1W2L1	Euclidean	Absolute	Single
	05	S1W2L2	Euclidean	Absolute	Unweighted average
	06	S1W2L3	Euclidean	Absolute	Complete
	07	S1W3L1	Euclidean	Relative	Single
	08	S1W3L2	Euclidean	Relative	Unweighted average
	09	S1W3L3	Euclidean	Relative	Complete
Association coefficient	10	S2W1L1	Jaccard	Binary	Single
	11	S2W1L2	Jaccard	Binary	Unweighted average
	12	S2W1L3	Jaccard	Binary	Complete
	13	S2W2L1	Unbiased Ellenberg	Absolute	Single
	14	S2W2L2	Unbiased Ellenberg	Absolute	Unweighted average
	15	S2W2L3	Unbiased Ellenberg	Absolute	Complete
	16	S2W3L1	Unbiased Ellenberg	Relative	Single
	17	S2W3L2	Unbiased Ellenberg	Relative	Unweighted average
	18	S2W3L3	Unbiased Ellenberg	Relative	Complete

a larger component acting as the system core. In [8], the authors claimed that an optimum clustering algorithm should avoid the case of many single class components and a single very large one. We propose a set of size criteria to accommodate their statements:

- The ratio of single class components, that is, the number of single class components divided by the total number of classes.
- The ratio of classes in the largest component, that is, the number of classes in the largest component divided by the total number of classes.
- The ratio of other classes (in the intermediate components), that is, the number of classes in the intermediate components divided by the total number of classes.

For example, in Fig. 6, the software system is composed of eight classes. The largest component is cmp_2, which has four classes, so the ratio of classes in the largest component is 50%. There is a single class component, cmp_3, so the ratio of single class components is 12.5% and cmp_1 is the only single intermediate component that has three classes, so the ratio of classes in the intermediate components is 37.5%. The sum of these three ratios is 100%, which indicates that all the classes in the subject system have been considered by these three ratios.

4.3.2. Coupling

In component-based systems, coupling represents how tightly one component interacts with the others. There have been several studies on component coupling metrics [19,20]. They considered different types of dependencies among components and measured different quality features of the components such as reusability, maintainability and understandability. However, they have not been widely accepted by the software engineering community. On the other hand, the class coupling metric *CBO* has been used by many researchers to predict the quality of classes [21,22,26]. The computation of *CBO* is supported by many public and commercial tools, such as OOMeter [23] and Borland Together [17]. Therefore, we propose a metric based on *CBO* for evaluating component coupling, viz., *CCR* (Coupled Components Ratio).

Originally, the *CBO* of a class was defined as the number of classes interacting with it. Accordingly, two components are said to be coupled if there is a connection between them. The coupling of a component cmp_i is defined as the ratio of the number of components coupled with cmp_i to the total number of components, ignoring cmp_i itself.

Let $CP(cmp_i)$ denote the set of components which are coupled to cmp_i in *UCMP*, $cmp_i \in UCMP$.

$CP(cmp_i) = \{cmp_j | cmp_j \in UCMP \wedge (|conn(cmp_i, cmp_j)| > 0 \vee conn(cmp_j, cmp_i)| > 0) \wedge i \neq j\}$

Then, the coupling of component cmp_i is defined as follows:

$$CCR(cmp_i) = \frac{|CP(cmp_i)|}{|UCMP| - 1}.$$

The *CCR* value of a component lies in [0, 1]. A component with a smaller *CCR* is better than that with a larger one. A $CCR(cmp_i)$ of 1 indicates that cmp_i is coupled with all the other components. In contrast, a $CCR(cmp_i)$ of 0 indicates that cmp_i is entirely independent; that is, it has no connections with any other components.

In Fig. 6, cmp_2 is connected with the other two components, so $CCR(cmp_2) = 1.00$. Likewise, $CCR(cmp_3) = 0.50$.

4.3.3. Cohesion

In component-based systems, cohesion represents how tightly classes in the same component are associated. Cohesion metrics usually cooperate with coupling metrics to measure quality features of components, such as reusability and maintainability. Existing studies on component cohesion metrics, such as [19,25], have not yet been widely applied in software engineering. However, Tight Class Cohesion has long been applied to predict class qualities such as fault-proneness and maintainability [21,22] and it is supported by many commercial tools, such as OOMeter [23] and Borland Together [17].

Tight Class Cohesion was defined as the percentage of public method pairs of a class that use common attributes. Accordingly, we proposed the component cohesion metric, viz., *TCC* (Tight Component Cohesion). *TCC* is designed in accordance with the original concept of Tight Class Cohesion. It is defined as the ratio of the number of interface operation pairs sharing the same class to the total number of interface operation pairs within a component.

Let $RC_{cmp}(op)$ be the set of classes in component *cmp* which are referenced by the interface operation *op* of *cmp*. Let *IOP*(*cmp*) be the set of interface operations that belong to component *cmp*. Let *Q*(*cmp*) be the set of interface operation pairs which share the same implementation class in component *cmp*: $Q(cmp) = \{(op_i, op_j) | RC_{cmp}(cmp) = \{(op_i, op_j) | RC_{cmp}(op_i) \cap RC_{cmp}(op_j) \neq \varphi \wedge op_i, op_j \in IOP(cmp) \wedge i \neq j\}$. Let *R*(*cmp*) be the set of total interface operation pairs: $R(cmp) = \{(op_i, op_j) | op_i, op_j \in IOP(cmp) \wedge i \neq j\}$. Then *TCC* can be defined as follows:

$$TCC(cmp) = \begin{cases} 0 & \text{if } |R(cmp)| = 0 \\ |Q(cmp)| / |R(cmp)| & \text{otherwise} \end{cases}.$$

The *TCC* value of a component lies in [0, 1]. A higher *TCC* value indicates that more similar behaviors are grouped together and thus more tightly related implementation classes belong together. This is preferable for component reusability and maintainability. A *TCC*(*cmp*) of 1 indicates that all the interface operations in *cmp* are connected by the implementation classes. If this is not the case, $TCC(cmp) = 0$.

In Fig. 6, suppose that $RC_{cmp2}(op_{41}) = \{cls_6\}$, $RC_{cmp2}(op_{42}) = \{c_6, c_7\}$, $RC_{cmp2}(op_{61}) = \{c_7\}$, then $Q(cmp_2) = \{(op_{41}, op_{42}), (op_{42}, op_{61})\}$, $R(cmp_2) = \{(op_{41}, op_{42}), (op_{41}, op_{61}), (op_{42}, op_{61})\}$, so $TCC(cmp2) = |Q(cmp2)|/|R(cmp_2)| = 2/3 = 0.67$. Since cmp_3 does not have any interface operation, $|R(cmp_3)| = 0$, and $TCC(cmp_3) = 0$.

4.4. Subject systems

We have used 11 subject systems in this study, which were developed using the Java programming language. Most were obtained directly from the Internet community and they vary in terms of the application domains and source code sizes. Table 4 gives details of the subject systems.

Fig. 7 shows the distribution of the number of class connections within the subject systems. For example, the ratio of classes which are connected to 5–10% of the classes in a subject system is 24.23%. Commonly, the classes in a software system are sparsely connected, that is, most have connections with only a few other ones. As illustrated in Fig. 7, more than 50% of classes have connections with less than 5% of other ones and around 80% of classes have connections with less than 10% of other ones. However, there are still a certain number of classes which have connections with many other ones. As shown in Fig. 7, around 1% of classes connect with more than 50% of other ones.

5. Experiments and evaluation results

A series of experiments were conducted for the 18 clustering strategies and the 11 subject systems in order to identify their respective strengths and weaknesses. The experiments aimed to evaluate the various clustering strategies for producing components of good quality. In this section, we present the experimental

Table 4
Details of the subject systems.

ID	System name	Version	# Of classes	LOC	Domain
01	DefectPredictor [27]	1.0	44	2804	Testing software
02	LBFSrv [28]	1.0	44	11,375	Networking software
03	BattlefiledSim	1.9	57	11,149	Computer game
04	Jaga	13.04.04	66	5138	Algorithm framework
05	JavaCC	4.0	84	13,933	Code generator
06	MTGForge		93	19,542	Computer game
07	Rhino	1.6R7	98	38,109	Chat server
08	Openreports	3.0	147	14,469	Web application
09	SableCC	3.1	161	28,318	Code generator
10	RSSOwl	1.2.3	187	46,074	RSS feed reader
11	Jigsaw	2.2.6	249	94,015	Application server

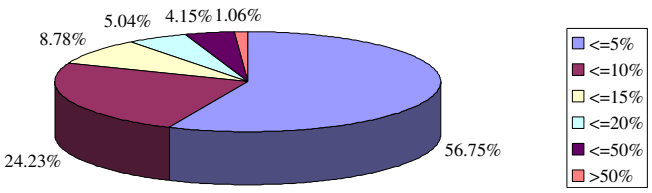

Fig. 7. The distribution of class connections within the subject systems.

results, which are evaluated by the proposed size, coupling and cohesion criteria. Note that we did not separately consider the 11 subject systems in this study. Instead, we considered them together, and the following experimental results give the average values of the 11 cases.

5.1. Size

Fig. 8 shows the experimental results of the 18 clustering strategies, which are evaluated by the size criteria for the 11 subject systems. In Fig. 8, the *x*-axis represents the clustering cuts (in steps of 5%), and the *y*-axis represents the distribution of the classes in terms of the three types of components (single, intermediate and largest). The results are averaged for the 11 subject systems. As can be seen in Fig. 8, at the beginning of clustering, single class components account for 100% of classes, while they are all merged into a single component at the end, as illustrated by the fact that the largest component accounts for 100% of classes at the 100th percentile of clustering. Fig. 8 shows that the distributions of classes in different types of components vary during clustering according to the clustering strategy.

As shown in Fig. 8, the *L*1-based strategy (referred to as the single linkage method-based strategy hereinafter) commonly identifies the smallest ratio, and the *L*3-based strategy identifies the largest ratio of classes in intermediate components for each similarity measure and weighting scheme. The *L*2-based strategy falls between the *L*1 and *L*3-based strategies, a result consistent with the conclusion presented in [7,8]. Generally the *L*1 linkage method leads to a very large component by continuously combining other single ones, while the *L*3 linkage method tends to produce many small components that are combined during the later stages.

For the *W*1 and *W*3 weighting schemes, *S*2-based strategies generally identify more intermediate components than *S*1-based ones, for each linkage method. For example, *S2W1L*1 clearly identifies more intermediate components than *S1W1L*1. But the case for *W*2-based strategies is different. We can see from Fig. 8 that *S*1-based strategies identify more intermediate components when the *L*1 and *L*2 linkage methods are employed, while the *S*2-based strategy identifies more intermediate components when *L*3 is employed.

In addition, it is interesting that the performance of the weighting schemes differs under various similarity measures. As shown in Fig. 8, although both schemes consider the class connection strengths, when *S*1 is employed, *W*2-based strategies clearly identify the largest ratio of intermediate components, while *W*3-based strategies identify the smallest ratio of intermediate components throughout clustering. Considering *S*2-based strategies, for a given linkage method, no great differences are found.

5.2. Coupling

Fig. 9 presents the *CCR* results of all the clustering strategies averaged by the 11 systems at various cuts during clustering, ranging from 0% to 100% of iterations, in steps of 5%. For convenience of expression, the curves with squares represent *S*1-based strategies, and those with triangles represent *S*2-based strategies. The various colors represent various weighting schemes and linkage methods.

As shown in Fig. 9, all the clustering strategies have the same *CCR* values at the 0th percentile of iterations, because clustering starts from the same set of single class components. However, clustering is affected by the various similarity measures and linkage methods employed in the clustering strategies, so the overall coupling of the partitions in the subject systems diverges as clustering proceeds. After all the classes are combined into a single component, we can no longer calculate the *CCR* values, so it is null at the 100th percentile of iterations in the figure.

As can be seen in Fig. 9, the *CCR* values increase as the clustering proceeds to higher iterations. This is because as ever more classes are combined into components, the number of coupled components of each component does not decrease as fast as the total number of components; this results in convergence of the numerator and denominator of the *CCR* criterion and causes a deterioration in coupling. As shown in Fig. 9, as clustering proceeds, the differences in the *CCR* values by the various strategies are increasing.

As we previously mentioned, the *L*1 linkage method naturally overrates the similarities between components. The combination of a component with a less similar one results in extra coupling with other ones, so *L*1 produces higher overall coupling compared with the other linkage methods. This can be validated by our experimental results on *CCR*, as shown in Fig. 9. For a given similarity measure and weighting scheme, *L*1 results in lower overall coupling while *L*3 results in higher overall coupling for the subject systems. The *CCR* values from the *L*2-based strategies fall between those of the *L*1 and *L*3-based ones.

Regarding the similarity measures, the weighting schemes have distinct influences on the *CCR* values of the identified components. Fig. 9 shows that *S*1-based strategies perform better with the *W*3 weighting scheme than the *W*1 one for a given linkage method. But *S*2-based strategies differ in that they perform better with the *W*1 weighting scheme than the *W*3 one for a given linkage method.

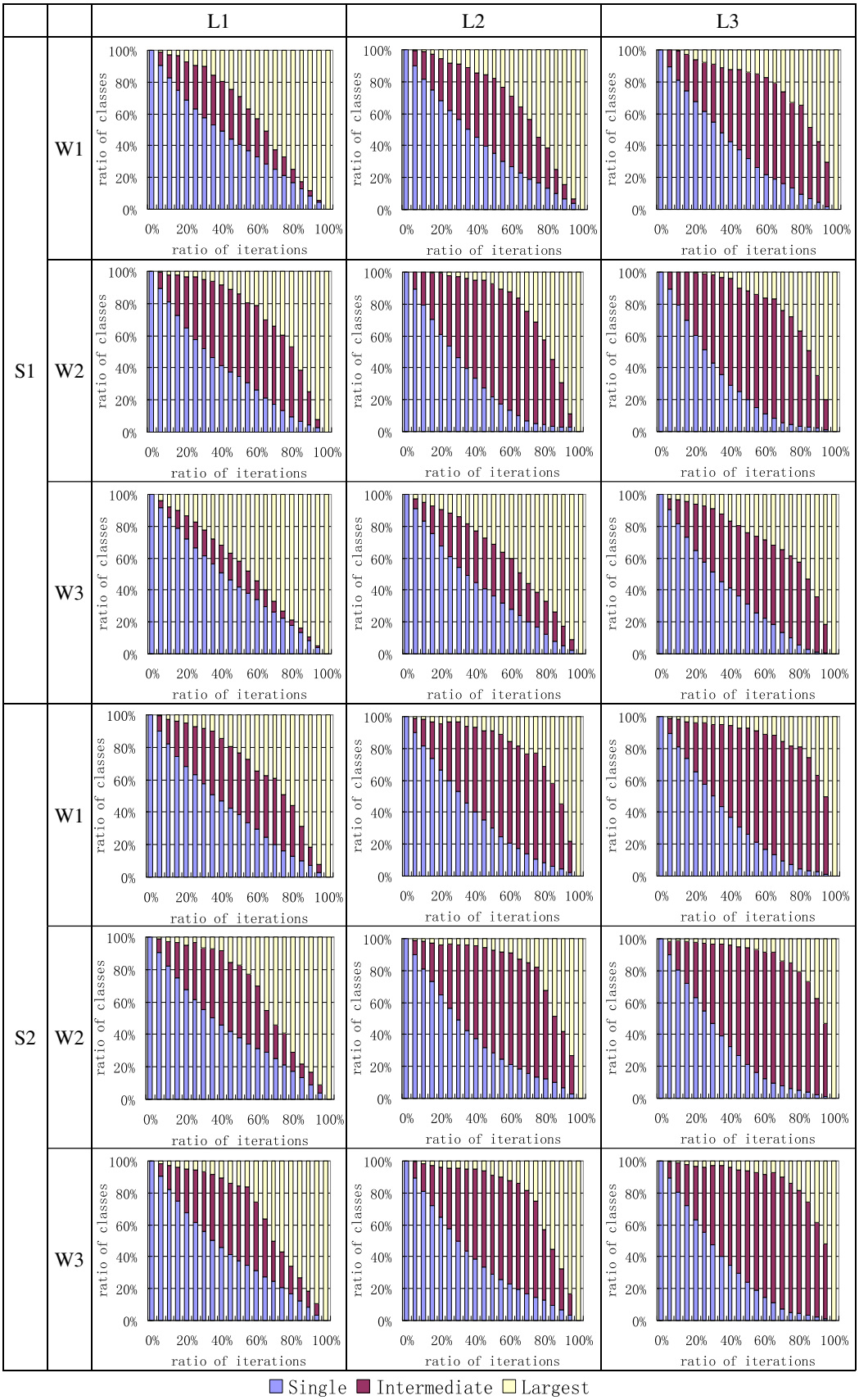

Fig. 8. The distribution of classes in various types of components.

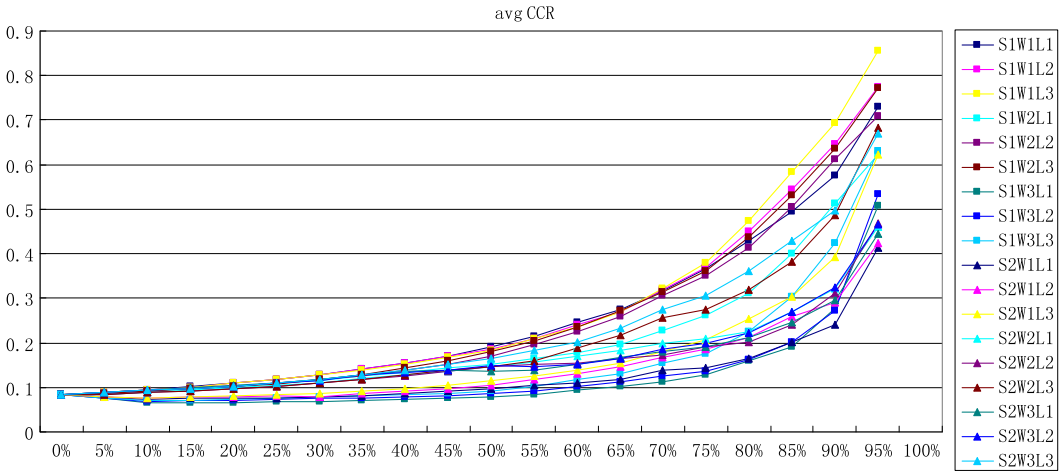

Fig. 9. Coupling evaluation results by various clustering strategies.

5.3. Cohesion

The cohesion criterion *TCC* evaluates how tightly the interface operations of a component are connected by the implementation classes. We evaluated the overall cohesion of the subject systems at various stages of clustering by using the *TCC* criterion. Fig. 10 presents the *TCC* values for all the clustering strategies averaged by the 11 subject systems at various cuts during clustering. Similar to that in the *CCR* evaluation, the curves with various colors and symbols represent various similarity measures, weighting schemes and linkage methods. All the clustering strategies have the same *TCC* values at the 0th percentile of iterations, as shown in Fig. 10. However, the cohesion values diverge as clustering proceeds, because clustering is affected by various similarity measures and linkage methods employed in the clustering strategies.

Fig. 10 shows that various clustering strategies lead to very different cohesion cases as clustering proceeds to higher iterations. Regarding the similarity measures, *S*1-based strategies produce more cohesive components than *S*2-based clustering algorithms when the *W*1 and *W*2 weighting schemes are employed, while *S*2-based clustering algorithms produce more cohesive components when the *W*3 one is employed. Considering the weighting schemes, *W*1-based clustering strategies produce the most cohesive components, and *W*3-based ones produce the most non-cohesive components. Considering the linkage methods, when employed with various weighting schemes, they show various clustering abilities. As can be seen from Fig. 10, for both *S*1 and *S*2-based clustering strategies, during most of the clustering process, *L*1 produces the most cohesive components under the *W*1 weighting scheme, while L3 produces the most cohesive components under the *W*2 and *W*3 ones.

In particular, *S*1-based strategies produce both the best and worst results. As shown in Fig. 10, when *W*1 is employed, the cohesion values remain at a relatively higher level during most of the iterations. However, when *W*3 is employed, *S*1 results in the lowest cohesion values. The other strategies fall between these two groups.

According to the definition of *TCC*, the number of interface operations is a key factor that influences the cohesion of a component. Fig. 11 shows the average number of interface operations owned by each component during clustering. As can be seen from Fig. 11, there are no evident differences among various clustering strategies during the first half of the clustering process. However, some strategies cause the average number of interface operations to increase, while others cause the number to decrease during the last half of the process. On the whole, all the *S*1-based strategies tend to produce components with a large number of interface operations. While, *S*2-based strategies exhibit a different pattern, that is, *L*3-based strategies retain a large average number of interface operations, while the other two linkage methods have a much smaller number.

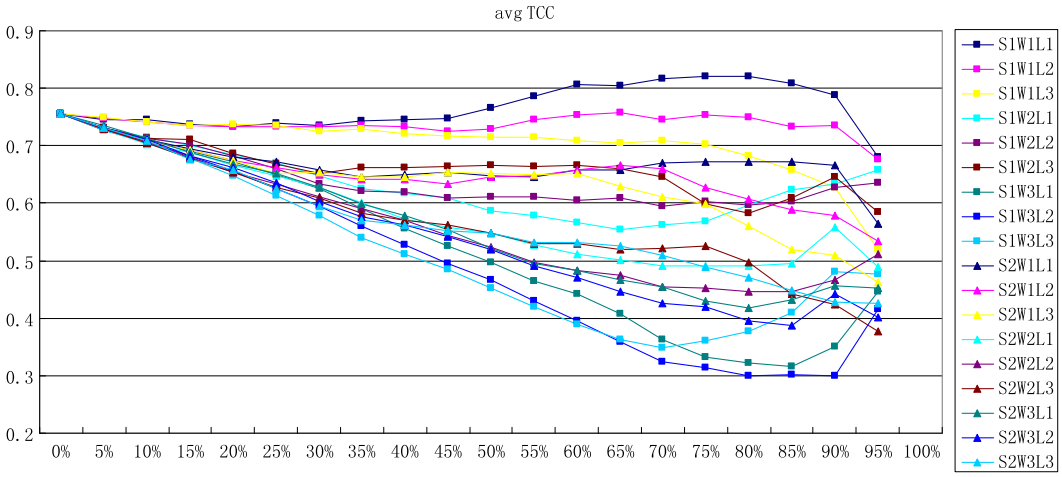

Fig. 10. Cohesion evaluation results by various clustering strategies.

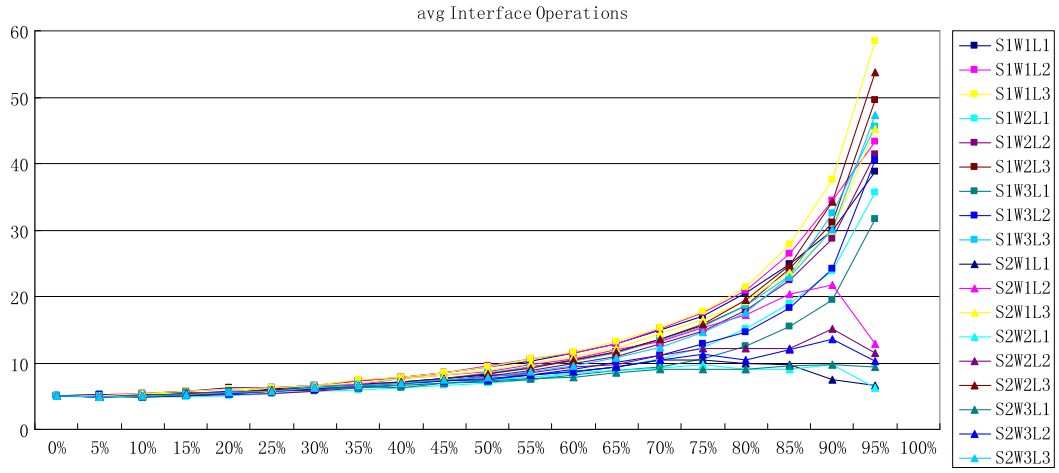

Fig. 11. Average number of interface operations.

An interesting observation is that although *W*1 and *W*3 produce more components with a large number of interface operations under the *S*1 similarity measure, these are definitely different cases of cohesion. Recall that strategies based on *S*1 and *W*1 result in the most cohesive components and those based on *S*1 and *W*3 result in the most non-cohesive components during most of the iterations, as shown in Fig. 10. We can infer that under *S*1 the *W*3 weighting scheme tends to merge classes that are independent but are widely used by the software system.

5.4. Overall evaluation

Considering the performance of the linkage methods with respect to the size, coupling and cohesion criteria, *L*3 performs best in terms of size, but worst in terms of coupling under both the *S*1 and *S*2 similarity measures. *L*1 has the opposite effect. It performs best in terms of coupling, but worst in terms of size. Considering cohesion, the performance of the linkage method differs according to the weighting scheme. For example, under the *W*1 weighting scheme, *L*1 is best and *L*3 is worst. Under the *W*3 weighting scheme, *L*3 performs best and *L*2 performs worst. Compared with the linkage methods, the weighting schemes have more significant effects on the similarity measures. For example, when *W*1 is employed with *S*1, the clustering results tend to be tightly cohesive but loosely coupled. When *W*3 is employed with *S*1, the clustering results tend to be loosely coupled, but the numbers of classes in the intermediate components are small. In addition, when *W*2 is employed with *S*2, the clustering results have many classes in intermediate components, but when *W*3 is employed with *S*2, the clustering results tend to be weakly cohesive.

The evaluation results reveal that the clustering algorithms have various abilities to identify components with respect to the proposed size, coupling and cohesion criteria, and it is difficult to produce perfectly satisfactory clustering results for a given clustering algorithm. Nevertheless, these observations can assist various component identification tasks according to practical demands.

Fig. 12 shows the coupling and cohesion of the clustering results by various clustering strategies in the same graph. In Fig. 12, the *x*-axis denotes the *TCC* values and the *y*-axis denotes the *CCR* values. The curves proceed from the bottom of the graph (lower cuts) towards the top (higher cuts). We divide this graph into four regions. The clustering results belonging to Region IV are considered the best, because the overall coupling is low and the overall cohesion is high, while the clustering results belonging

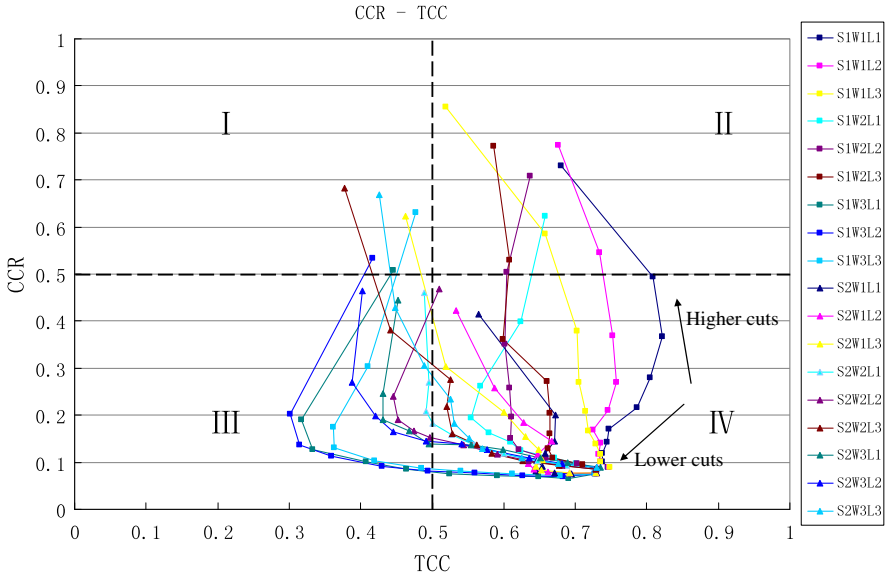

Fig. 12. The overall coupling and cohesion by various clustering strategies.

to Region I are the worst, because they have high coupling and low cohesion.

We can make some interesting observations from Fig. 12. First, during most of the clustering process, the clustering strategies avoid producing poor clustering results which are tightly coupled and weakly cohesive. The fact that only a few of the clustering results belong to Region I and appear at very late clustering stages suggests that the clustering strategies perform well enough to avoid producing components which are both tightly coupled and weakly cohesive. And, many clustering strategies tend to perform well on either coupling or cohesion, but not both. As shown in Fig. 12, *S*2-based strategies produce less coupled but less cohesive components than *S*1-based ones. The weighting schemes have notable effects on the similarity measures. The strategies based on *S*1 and *W*1 tend to produce both highly cohesive and tightly coupled components (Region II), while those based on *S*1 and *W*3 tend to produce both loosely coupled and weakly cohesive components (Region III).

Although clustering strategies can have good performance either in terms of coupling or cohesion, they do not always have good performance in terms of the size criteria. Fig. 13 shows the sizes of the clustering results at each stage within Region IV. The numbers marked at each node represent the clustering cuts, the average size of the components and the ratio of the single class components at a certain clustering cut. Because the later stages of clustering are more significant, we focus on the clustering results from the second half of the clustering process.

As can be seen from Fig. 13, various clustering strategies have the same average component sizes at the same cuts during clustering; however, the ratios of the single class components are significantly different. In addition, during the second half of the clustering process, all or parts of the clustering results of the *W*1 and *W*2-based strategies belong to Region I, while for the *W*3 weighting scheme, only *L*3-based strategies belong to this region. Fig. 13 is further divided into three layers based on the coupling and cohesion values. The clustering results belonging to an inner layer such as Layer (1) are more favorable than those belong to an outer layer such as Layer (3). As can be seen, only the strategies based on *S*1 and *W*1 produce clustering results belonging to Layer (1); however, the ratios of the single class components are relatively higher than the other clustering results. Among the clustering results belonging to Region I, those produced by the strategies based on *S*1 and *W*2 have the lowest ratios of single class components and most belong to Layers (2) and (3).

5.5. Threats to validity

The primary threat to validity for our study is related to the number and type of subject systems. We have used 11 subject systems for our experiments. Experiments on other subject systems may be performed to further support our arguments according to the proposed criteria. Moreover, to generalize our results, it is necessary to perform experiments with subject systems in various size and application domains. Our subject systems were chosen from different application domains. However, according to Table 4, the sizes of the subject systems ranged from 2KLOC to 9KLOC, which means that our study only focused on systems of small and medium size.

In order to conduct fair experiments, we have developed a tool (CIETool) to facilitate component identification and evaluation. Since there was a lack of widely accepted coupling and cohesion metrics, we proposed a set of criteria to evaluate the coupling and cohesion of the identified components. The size, coupling and cohesion criteria were derived from existing metrics that are widely used by researchers to reduce subjectivity during clustering results evaluation.

6. Related work

In [10], Wiggerts presented an overview of software clustering techniques. He performed a detailed analysis on similarity measures and existing clustering algorithms. Those studies involved only a concept level comparison. Instead, we performed an empirical study by using a set of practical subject systems. In [29], Lee et al. proposed a hierarchical approach for object-oriented legacy systems evolution. In their approach, software entities were partitioned into small groups by various static object-oriented relationships, such as inheritance, composition and aggregation, and then they were hierarchically combined into larger components by considering various class dependency relationships. They proposed a set of component metrics, including cohesion, complexity and connectivity metrics to facilitate clustering. Mitchell et al. [11] have developed a tool called Bunch to partition legacy software systems at various granularity levels. They implemented heuristic search

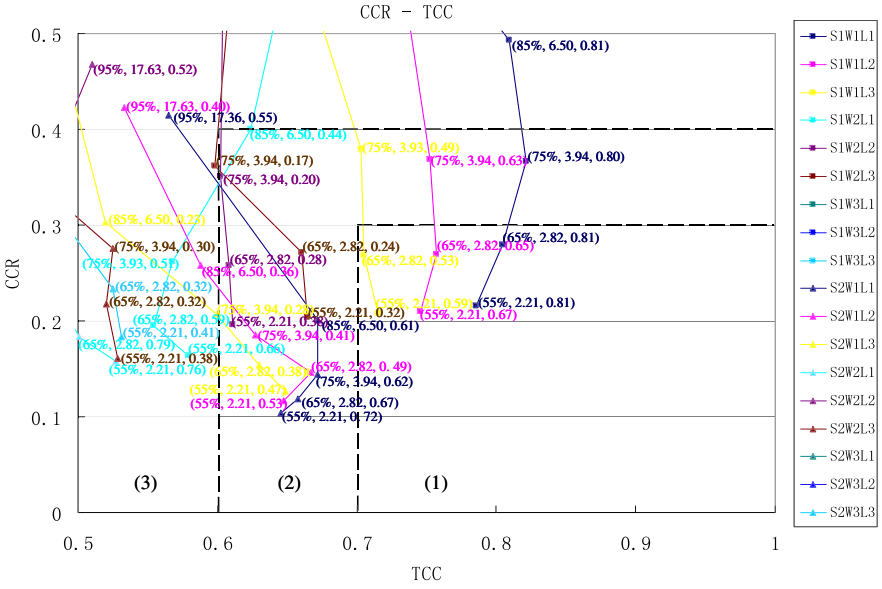

Fig. 13. The comparison of coupling, cohesion and size criteria.

algorithms, such as the genetic and hill-climbing algorithms, in the Bunch tool. They proposed an optimization function called Modularization Quality (MQ), which aims to maximize the intra-connectivity of the identified components and minimize their inter-connectivity. Because existing approaches were proposed for various purposes, the clustering results produced generally differed greatly. In contrast to them, we performed a comparative study on existing approaches, but not proposed a new approach.

Several component metrics have been proposed to qualify various characteristics of software components. Washizaki et al. [19] proposed a metrics suite for measuring the reusability of components based on the limited static information that can be obtained from outside components without any source code. Different from us, they did not use cohesion and coupling criteria, because these usually needed metrics that required analysis of the source code. Choi et al. [20] proposed a component coupling metric based on both static and dynamic relationships between classes. They considered different call types between methods, such as create, delete, write and read, and assigned different weights to them. In contrast to them, we just considered static relationships because the dynamic aspects of interaction between objects at run-time lacks of an accurate specification in practice yet.

Experiments for evaluating component identification approaches have also been conducted by some researchers. Luo et al. [31] performed an experimental study of two graph analysis-based component identification approaches: the agglomerative approach and the partitioning approach. Their study showed that the former was better than the latter in terms of the component capture capability, and the former was more promising for recovering system architectures, because it can provide hierarchical structures and a better partition. Different from them, we just considered the agglomerative approach, and investigated the various performances when various weighting schemes, similarity measures and linkage methods are applied. In [32], Davey et al. presented their studies on data clustering techniques for software remodularization. They argued that the behaviors of different clustering techniques differed according to the types of software systems they were applied to. In contrast, we observed that different clustering algorithms had different performances when they were applied to the same subject software systems.

Some research efforts have aimed at componentization for legacy systems. Quinlan et al. [33] proposed an approach to automating the generation of components from legacy scientific applications. They extract language-neutral specifications of interfaces from source code and reduce interfaces and code to only those methods that are really needed by users. Kim et al. [34] presented an adaptation method through adaptation pattern for component reuse. Different from us, they aim to solve the problem that how to connect components in case that component interfaces are different from each other when components are assembled into an application component. Their approach focuses on components in the binary level, but not source code level. In the field of software reengineering, Palladio Component Model [30] is an approach that allows the prediction of performance attributes, such as throughput and response time, for component-based software architectures. In contrast, our focus is on the evaluation of internal component structure, such as cohesion and coupling.

7. Conclusion and future work

Component identification is a critical part of software reengineering. In this paper, we proposed three types of weighting schemes according to various judgments on the connection manners between the classes, considering the characteristics of object-oriented systems. We performed a series of experiments for 18 clustering strategies employing various similarity measures, weighting schemes and linkage methods. Eleven subject systems having different application domains and source code sizes were used in the experiments. In order to identify the respective weaknesses and strengths of various clustering strategies, a set of criteria were proposed to evaluate the clustering results. The evaluation results revealed that the hierarchical clustering algorithms produced quite different clustering results and it was difficult to produce perfectly satisfactory results for a given clustering algorithm. Nevertheless, these algorithms demonstrated varied capabilities to identify components with respect to the proposed size, coupling and cohesion criteria.

Future work will focus on the following aspects. Our experiments show that it is difficult to define a given number of components or a cutoff position to obtain satisfactory clustering results. Stopping criteria are used to determine when to terminate clustering in order to achieve an overall high quality of clustering results. Several stopping criteria have been proposed for hierarchical clustering algorithms, such as the required number of components and the cutoff at a given clustering stage. We expect that a more comprehensive stopping criterion will be proposed to balance the various factors influencing the overall quality of a software system. Considering component metrics, we evaluate components sizes by calculating their member classes. Additionally, in consideration of the fact that a component should be a reusable part, interface could also be a measure of component size. There are many different mechanisms capable of constructing class connections, such as import invocation, export invocation and inheritance [22]. Because of the limitations of application environments and time costs, a thorough exploration of all the connection mechanisms is generally unpractical. We expect more research efforts on applying an appropriate class connection mechanism to component identification technology.

In this study, the evaluation criteria were used to examine if various clustering algorithms produced the same results when they were applied to the same subject software systems. However, when the subject systems are selected from various application areas, a specific clustering algorithm may have various performances. So it is necessary to investigate the various performances of a specific clustering algorithm when it is applied to various subject systems. In addition, many popular component metrics focused on evaluating a flat decomposition of the software system [24]. Therefore, we adopted a set of component metrics for flat decomposition in this study. However, a flat structure is not enough for deciding whether the architecture of a system is good or not, component metrics for layered architecture are expected for evaluating the quality of a software system. Finally, the hierarchical clustering method is a highly time-consuming process, especially when it is employed in a large-scale software system. Improving the efficiency of agglomerative hierarchical clustering algorithms should also be considered in future research.

Acknowledgment

This work was supported by the Grant of the Korean Ministry of Education, Science and Technology (The Regional Core Research Program/Institute of Logistics Information Technology).

References

[1] K. Bennet, Legacy systems: coping with success, IEEE Software 12 (1) (1995) 19–23.
[2] N. Gold, The Meaning of Legacy System, SABA Project Report: PR-SABA-01 Version 1.1, The Centre for Software Maintenance, Computer Science Department, University of Durham, 1998.
[3] M. Colosimo, A.D. Lucia, G. Scanniello, G. Tortora, Evaluating legacy system migration technologies through empirical studies, Information and Software Technology 51 (2) (2009) 433–447.
[4] B. Mirkin, Mathematical Classification and Clustering, Springer, 1996.

[5] P. Dissaux, Using the AADL for mission critical software development, in: Proceedings of ERTS Conference, January 2007.
[6] V. Tzerpos, R.C. Holt, Software Botryology: automatic clustering of software systems, in: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, 1998, pp. 811–819.
[7] O. Maqbool, H.A. Babri, Hierarchical clustering for software architecture recovery, IEEE Transactions on Software Engineering 33 (11) (2007) 759–780.
[8] F.C. Meng, D.C. Zhan, X. F. Xu, Business component identification of enterprise information system: a hierarchical clustering method, in: Proceedings of the IEEE International Conference on e-Business Engineering, October 2005, pp. 473–480.
[9] D.A. Jackson, K.M. Somers, H.H. Harvey, Similarity coefficients: measures of co-occurrence and association or simply measures of occurrence?, The American Naturalist 133 (3) (1989) 436–453
[10] T.A. Wiggerts, Using clustering algorithms in legacy systems remodularization, in: Proceedings of the 4th Working Conference on Reverse Engineering, October 1997, pp. 33–43.
[11] B.S. Mitchell, S. Mancoridis, On the automatic modularization of software systems using the bunch tool, IEEE Transactions on Software Engineering 32 (3) (2006) 193–208.
[12] O. Maqbool, H.A. Babri, The weighted combined algorithm: a linkage algorithm for software clustering, in: Proceedings of Software Maintenance and Reengineering, March 2004, pp. 15–24.
[13] G. Salton, Development in automatic text retrieval, Science 253 (1991) 974–979.
[14] T. Kunz, Developing a Measure for Process Cluster Evaluation, Technical Report TI-2/93, Technical University Darmstadt, 1993.
[15] H. Lung, M. Zaman, A. Nandi, Applications of clustering techniques to software partitioning, recovery and restructuring, Journal of Systems and Software 73 (2) (2004) 227–244.
[16] G. Canfora, M.D. Penta, New frontiers of reverse engineering, in: Proceedings of Future of Software Engineering, 2007, pp. 326–341.
[17] Borland Together, <http://www.borland.com/us/products/together>.
[18] D.H. Hutchens, V.R. Basili, System structure analysis: clustering with data bindings, IEEE Transactions on Software Engineering 11 (8) (1985) 749–757.
[19] H. Washizaki, H. Yamamoto, Y. Fukazawa, A metrics suite for measuring reusability of software components, in: Proceedings of the 9th International Software Metrics Symposium, September 2003, pp. 211–223.
[20] M. Choi, S. Lee, A coupling metric applying the characteristics of components, in: Proceedings of Workshop on Component Based Software Engineering and Software Process Model, vol. 3983/2006, May 2006, pp. 966–975.
[21] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[22] L. C. Briand, J. Wüst, S. V. Ikonomovski, Hakim Lounis, Investigating quality factors in object-oriented designs: an industrial case study, in: Proceedings of the 21st International Conference on Software Engineering, May 1999, pp. 345–354.
[23] J.S. Alghamdi, R.A. Rufai, S.M. Khan, OOMeter: a software quality assurance tool, in: Proceedings of the 9th European Conference on Software Maintenance and Reengineering, March 2005, pp. 190–191.
[24] M. Shtern, V. Tzerpos, Lossless comparison of nested software decompositions, in: Proceedings of the 14th Working Conference on Reverse Engineering, October 2007, pp. 249–258.
[25] L.C. Briand, J.W. Daly, J.K. Wüst, A unified framework for cohesion measurement in object-oriented systems, Empirical Software Engineering 3 (1) (2004) 65–117.
[26] L.C. Briand, J.W. Daly, J.K. Wüst, A unified framework for coupling measurement in object-oriented systems, Transactions on Software Engineering 25 (1) (1999) 91–121.
[27] T.Y. Kim, Y.K. Kim, H.S. Chae, An experimental study of generality of software defects prediction models based on object oriented metrics, Journal of Information Processing Systems 16-D (3) (2009) 407–416.
[28] J.G. Park, H.S. Chae, E.S. So, A dynamic load balancing approach based on the standard RFID middleware architecture, in: Proceedings of IEEE International Conference on E-Business Engineering, October 2007, pp. 337–340.
[29] E.J. Lee, W.C. Shin, B.J. Lee, C.S. Wu, Extracting components from object-oriented system: a transformational approach, IEICE Transactions on Information and Systems E88-D (6) (2005) 1178–1190.
[30] S. Becker, H. Koziolek, R. Reussner, Model-based performance prediction with the Palladio component model, in: Proceedings of the 6th International Workshop on Software and Performance, February 2007.
[31] J. Luo, R. Jiang, L. Zhang, H. Mei, J. Sun, An experimental study of two graph analysis based component capture methods for object-oriented systems, in: Proceedings of the 20th International Conference on Software Maintenance, September 2004, pp. 390–398.
[32] J. Davery, E. Burd, Evaluating the suitability of data clustering for software remodularization, in: Proceedings of the 7th Working Conference on Reverse Engineering, November 2000, pp. 268–276.
[33] D. Quinlan, Q. Yi, G. Kumfert, T. Epperly, T. Dahlgren, M. Schordan, B. White, Toward the automated generation of components from existing source code, in: Proceedings of the 2nd Workshop on Productivity and Performance in High-end Computing, February 2005.
[34] J. Kim, O. Kwon, J. Lee, G. Shin, Component adaptation using adaptation pattern components, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, October 2001, pp. 1025–1029.
[35] P. Andritsos, V. Tzerpos, Information-theoretic software clustering, IEEE Transactions on Software Engineering 31 (2) (2005) 150–165.