Component Identification and Evaluation for Legacy Systems
—— An Empirical Study ——

JianFeng CUI†, Member and HeungSeok CHAE††, Nonmember

SUMMARY In the field of software reengineering, many component identification approaches have been proposed for evolving legacy systems into component-based systems. Understanding the behaviors of various component identification approaches is the first important step to meaningfully employ them for legacy systems evolution, therefore we performed an empirical study on component identification technology with considerations of their similarity measures, clustering approaches and stopping criteria. We proposed a set of evaluation criteria and developed the tool CI-ETool to automate the process of component identification and evaluation. The experimental results revealed that many components of poor quality were produced by the employed component identification approaches; that is, many of the identified components were tightly coupled, weakly cohesive, or had inappropriate numbers of implementation classes and interface operations. Finally, we presented an analysis on the component identification approaches according to the proposed evaluation criteria, which suggested that the weaknesses of these clustering approaches were the major reasons that caused components of poor-quality.

key words: component identification, evaluation criteria, legacy system, software reengineering, empirical study

1. Introduction

Software reengineering is the process of analyzing a legacy system to identify the system’s components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction [1]. Component identification technology has been used to reengineer legacy systems in order to reconfigure them into finely organized component-based systems. With a suitable component identification approach, the situation of a legacy system becoming completely obsolete can be avoided and the redevelopment cost for users is reduced.

The basic assumption motivating this study is that well-designed software systems are organized into cohesive subsystems that are loosely interconnected [2]. The process of component identification starts by parsing the source code of a legacy system. The parsed code is then analyzed to produce views of the software structure at varying levels of detail. These views encapsulate source code level entities and their relationships into software components. So far a number of component identification approaches have been presented in the literature and they commonly focus on forming groups of entities that are similar to one another and different from those in other groups [11].

Understanding the behaviors of various component identification approaches is the first important step to meaningfully employ them for legacy systems evolution, so we provided an empirical study on existing component identification approaches with considerations of their similarity measures, clustering approaches and stopping criteria. In this paper, our interest is object-oriented legacy systems. We selected three typical component identification approaches in the literature, applied them to various subject systems, and then justified the performances of those approaches by evaluating the identified components with a set of proposed metrics.

The major contributions of this study are: First, we provided an overview of existing component identification approaches by considering their similarity measurements, clustering approaches and stopping criterion. Second, we performed a repeatable and comparable empirical study on component identification technology. In order for that, we proposed a set of component metrics and developed the tool CI-ETool to automate the process of component identification and evaluation. Third, through the evaluation results we found that existing component identification approaches produced many components of poor quality when evaluated with criteria such as size, cohesion, and coupling; that is, most of the identified components were tightly coupled, weakly cohesive, or had inappropriate sizes of components and their interfaces. Finally, we provided an analysis on these poor-quality components and discussed the reasons why they were produced by those employed component identification approaches.

The rest of this paper is organized as follows. In Sect. 2, an overview of component identification technology is provided, where we also introduce three typical component identification approaches. Section 3 presents experimental settings: the tool CI-ETool developed to automate the process of component identification and evaluation, the evaluation criteria for identified components, and the employed subject systems. In Sect. 4, we present experimental results and observations from this study. Section 5 is related work. Finally, Sect. 6 summarizes our study and provides suggestions for future work.
2. Overview of Component Identification Approaches

Generally speaking, component identification is a software clustering technique applied in the area of software reengineering [8]. When a component identification approach is applied to legacy systems, we should firstly decide which entities to cluster. The candidate entities may be routines and modules for structured systems, or classes for object-oriented systems. In this paper, we limit our study in object-oriented software systems, so classes are considered the entities to be clustered. Generally an object-oriented system is composed of a set of classes and various usage dependencies among them. Classes can be thought of as nodes in a graph and usage dependencies between them as edges in that graph. The weights of edges indicate how strongly classes connect with each other. As an example, Fig. 1 shows a weighted graph which represents a software system composed of 5 classes (nodes) and each of the usage dependencies (edges) is given a value (weight).

The majority of work on component identification approaches for object-oriented systems focuses on identifying components by usage dependencies between classes, which can be derived from a static analysis of design documents or source code. The dynamic aspects of interaction between objects at run-time have barely been investigated and are not yet considered in practice, so we restrict ourselves to the analysis of static usage dependencies.

Many component identification approaches have been presented in the literature. Each of them looks at component identification as a software clustering problem from different aspects. In this section, we will firstly discuss several key features in component identification technology, and then, introduce three typical component identification approaches which are used in this empirical study.

2.1 Key Features of Component Identification Technology

Component identification technology tries to help software engineers reengineer legacy software systems into finely organized component-based systems. Generally, the component identification results can be significantly influenced by three key factors: similarity measurement, clustering approach and stopping criterion. They are illustrated as below:

- Similarity measurement
  In order to identify tightly cohesive and loosely coupled components, we should consider how to determine whether two entities are closely related or in other words, when two entities are found similar. In order to decide which two entities are more similar than the other ones, some kind of measure of similarity is required. According to Jackson et al. [9], the choice of a proper similarity measurement has a significant influence on the component identification results.

  Under various aspects of considerations on usage dependencies, a variety of similarity measurements can be applied to component identification approaches. Commonly they are divided into two types [4]: direct link and sibling link. The direct link method puts together entities that depend on one another, and the sibling link method puts together entities that have the same behavior. Take Fig. 1 as an example. By using the direct link method, \(cls_1\) and \(cls_3\) are the most similar because the weight of the edge between \(cls_1\) and \(cls_3\) is the largest; on the other hand, by using the sibling link method, \(cls_4\) and \(cls_5\) are the most similar because they have the same usage dependencies (they both have connections with \(cls_1\) and \(cls_2\)).

- Clustering approaches
  Clustering techniques have been widely utilized in the area of software engineering. Software clustering approaches can be broadly divided into two categories, namely, partitioning and hierarchical [11]. The partitioning approach usually starts with an initial partition consisting of a certain number of components. The partition is then modified iteratively until some criterion is optimized while keeping the number of components constant. For example, a partitioning clustering approach is applied to the system shown in Fig. 1 for producing two components. The system can be partitioned in different ways, and the most optimal one is determined by a specific similarity measurement. Figure 2 (a) shows an example of two different partitions of the system.

  By contrast, the hierarchical approach can produce a hierarchical decomposition for a software system without defining the number of components in advance. Earlier iterations of the hierarchical clustering approach provide a detailed view of the architecture and later iterations present a high-level view. For example, Fig. 2 (b) gives a hierarchical view of the system given in Fig. 1 by the hierarchical approach. As shown in Fig. 2 (b), more similar classes are joined at earlier iterations and less similar classes are joined at later iterations.

\[\text{Fig. 1 An example system.}\]

\[\text{Fig. 2 Examples of clustering approaches.}\]
• Stopping criterion
Generally component identification approaches continuously optimize the partitioning of a software system until a stopping criterion is fulfilled. Typically, this is taken to be some time or budget constraint on computation effort or it may be formulated as a criterion that must be met by the proposed solution [30]. More complex stopping criteria which combine multiple software metrics may be utilized. Commonly used stopping criteria include number of components, similarity threshold or time limitation.

2.2 Typical Component Identification Approaches
For conducting a thorough and comparative empirical study on component identification technology, three typical component identification approaches which employed different similarity measurements, clustering approaches and stopping criteria were considered. Let us call them Lung’s approach [14], Lee’s approach [15] and the Bunch approach [16], respectively. A brief description of these approaches is provided below by considering the three issues discussed in the above subsection.

• Lung’s approach
Lung et al. [14] performed a study on applying a numerical taxonomy clustering technique to software applications and proposed an approach based on this technique, called Lung’s approach. Lung’s approach uses sibling link method as the similarity measurement. The usage dependencies between classes are denoted by a binary value 0 or 1. For quantitatively calculating similarity, an adjacency matrix is created to represent usage dependencies between classes. Lung’s approach employs a typical hierarchical clustering algorithm which iteratively combines classes into components until the similarity value arrives at a predefined threshold.

• Lee’s approach
Lee et al. [15] proposed a component identification approach for object-oriented system evolution, which considered various structure relationships between classes. Lee’s approach consists of two phases. In the first phase classes with strong structure relationships such as composition and generalization relationships are grouped into components. These components constitute a set of basic components. The second phase aims to refine the system based on the basic components along with a hierarchical clustering algorithm and a set of quality metrics. Lee’s approach uses a direct link similarity measurement in the refinement phase, which is based on the proposed coupling and cohesion metrics. An overall quality metric is proposed to balance coupling and cohesion among the total components, which is used as the stopping criterion. Once there is no improvement on system overall quality, the clustering process is terminated.

• The Bunch approach
Mitchell et al. [16] proposed a partitioning based clustering approach and implemented it in a tool named Bunch. The Bunch approach uses heuristics algorithms to navigate through the search space of all possible system partitions. During the clustering process, it evaluates the quality of a system partition with a direct link similarity measurement \( MQ \). \( MQ \) determines the quality of a system partition quantitatively as the trade-off between interconnectivity (i.e., dependencies between the modules of two distinct subsystems) and intraconnectivity (i.e., dependencies between the modules of the same subsystem). The process of clustering terminates when \( MQ \) no longer improves.

As illustrated above, these three component identification approaches employ different types of similarity measurements, clustering approaches and stopping criteria to control their clustering processes. Table 1 summarizes the characteristics of the three component identification approaches.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Similarity measurement</th>
<th>Clustering approach</th>
<th>Stopping Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lung’s</td>
<td>Sibling link</td>
<td>Hierarchical</td>
<td>Similarity threshold</td>
</tr>
<tr>
<td>Lee’s</td>
<td>Direct link</td>
<td>Hierarchical</td>
<td>Overall quality</td>
</tr>
<tr>
<td>Bunch</td>
<td>Direct link</td>
<td>Partitioning</td>
<td>MQ</td>
</tr>
</tbody>
</table>

3. Experimental Settings
In this section, the experimental settings utilized in our study are presented. In order to perform a comparative study on component identification technology, the earlier described three approaches are applied to eight subject systems, and the identified components are evaluated with a set of component metrics. The process of component identification and evaluation is automated by the developed tool CIETool. The detailed experimental settings are described as following.

3.1 CIETool
In order to automate the process of component identification and evaluation, we have developed a tool named CIETool which integrates three component identification approaches and a set of evaluation criteria. CIETool helps software engineers to extract class dependency information from legacy systems, perform component identification approaches, and then evaluate the results based on a set of evaluation criteria. To ensure the correctness of the experimental results, we used the open APIs provided by Borland Together [12] to extract dependency relations from source codes of the subject systems.

3.2 Evaluating Criteria
Size, coupling and cohesion aspects of a software system
are the quality attributes that can seriously impact the maintenance, evolution, and reuse, so our evaluation criteria consider these three aspects for the components produced by various component identification approaches.

Figure 3 gives an example of a software system which contains 7 classes and is partitioned into three components: cmp1, cmp2, and cmp3. Each component contains one or more classes. Interface operations contain all of the functions of a component and they are the only things that are exposed to component users. An interface operation between two components is represented by a directed line. For example, class cls2 invokes the operation op61 of class cls6, and cls2 and cls6 are owned by components cmp1 and cmp2, respectively, so op61 is an interface operation of component cmp2. Similarly, op21 is an interface operation owned by cmp1.

The following notations are used to describe a software system:

- Let UC denote the set of classes in a software system S, then |UC| is the number of classes of S.
- Classes can be grouped into various components. Let CSET(cmp) denote the set of classes in the component cmp. We suppose that CSET(cmpm) ∩ CSET(cmpn) = ∅, if m ≠ n.
- Let IOP(cmp) be a set of interface operations that belong to the component cmp.
- For classes clsi and clsj, the predicate uses(clsi, clsj) returns true if the attributes of clsi is referenced by clsj or the operations of clsj is invoked by clsi, otherwise false.
- For two components cmpi and cmpj, the predicate uses(cmpi, cmpj) returns true if a class cls is in component cmpi uses a class cls in component cmpj, otherwise false.

The evaluation criteria are illustrated as below.

### 3.2.1 Size Criteria

Size is a critical feature for software components. A large number of size metrics have been proposed in the literature. According to [17], the size of a component and the number of its interface operations provide an estimate of the complexity for the component and they could be helpful in improving system’s quality because complex component complicate testing, debugging and maintenance. So, in this study, we chose the number of implementation classes and the number of interface operations as size criteria. These two metrics are defined as below:

- **NOIC** (Number Of Implementation Classes)
The number of implementation classes owned by a component cmp is defined as:
  \[ \text{NOIC}(\text{cmp}) = |\text{CS ET}(\text{cmp})| \]
  For example, in Fig. 3, component cmp3 consists of only one class, so NOIC(cmp3) = 1. Component cmp2 has three classes, so NOIC(cmp2) = 3.
  Components with either a very large number or a very small number of classes are not desirable. A component which owns a single class indicates it is never involved in the process of component identification, and a component which owns too many classes conceals its design details. Both of these two types of components are of poor quality.

- **NOIO** (Number Of Interface Operations)
  In this study, interface operation is defined as the class members in a component that are invoked or referenced by any other component. Interface operations can be obtained from the subject systems by a static analysis of the source code.
  The number of interface operations owned by a component cmp is defined as:
  \[ \text{NOIO}(\text{cmp}) = |\text{IOP}(\text{cmp})| \]
  For example, in Fig. 3, the operations in component cmp3 are never invoked by other components, so NOIO(cmp3) = 0; for component cmp2, NOIO(cmp2) = 3 because three operations, op41, op42 and op61, are invoked by the other two components.
  Components with either a very large number or a very small number of interface operations are not desirable. Many interface operations bring about high complexity to a component. On the contrary, a component without interface operations does not provide any usability to users.

### 3.2.2 Coupling Criterion

In component-based systems, coupling represents how tightly a component interacts with other components. There were some studies on component coupling metrics, such as [6], [7]. They considered different types of usage dependencies among components and evaluated different quality features of components such as reusability, maintainability and understandability. Currently, there are no widely accepted component coupling metrics. However, CBO (Coupling Between Objects) [13] is one of the most widely used class coupling metric in the software engineering community. Many researches have applied CBO as means to predict quality such as fault-proneness and the maintainability of classes [27], [28]. The computation of CBO is supported by many public and commercial tools such as OOMeter [29] and Borland Together [12].

In this study, we proposed a coupling metric based on CBO, called CCR (Component Coupling Rate). Originally, CBO was defined as a count of the number of other classes...
which it interacts with. Conforming to this concept, two components are coupled if there is a uses relationship between them. CCR for a component cmp is defined as the ratio of the number of components coupled to component cmp to the total number of components within a system except cmp itself.

Let CP(cmp) denote a set of components coupled to component cmp, within the system S:

\[ CP(cmp) = \{cmp_j|(uses(cmp, cmp_j))\} \cup uses(cmp, cmp) \cap \{cmp, cmp_j \in S \land cmp \neq cmp_j\} \]

Then, CCR for a component cmp can be defined as follows:

\[ CCR(cmp) = \frac{|CP(cmp)|}{|S| - 1}, 0 \leq CCR(cmp) \leq 1 \]

As an example, in Fig. 3, cmp2 is connected with the other two components, so CCR(cmp2) = 2/(3 - 1) = 1.00. Similarly, CCR(cmp3) = 1/(3 - 1) = 0.50.

CCR is designed to indicate that the larger CCR value implies a tighter coupling between components. In other words, a component with a smaller CCR value is more independent than that with a larger one. If CCR(cmp) = 1, then component cmp is coupled to all the other components. In contrast, CCR(cmp) = 0 means that component cmp is entirely independent; that is, it has no usage dependency with other components.

### 3.2.3 Cohesion Criterion

In component-based systems, cohesion represents how tightly classes in a same component depend on each other [3]. A number of cohesion metrics have been proposed for evaluating component cohesion which only consider intra-dependencies in a component, such as [6], [12]. However, many components still can be considered cohesive although they have few intra-dependencies if their interface operation performs similar functions.

As for cohesion, the class cohesion metric Tight Class Cohesion [3] is one of the most popular class cohesion metric. It is defined as the percentage of pairs of public operations of the class with common attributes usage. Tight Class Cohesion has been applied to predict quality features such as fault-proneness and maintainability [27], [28] of classes and the calculation of Tight Class Cohesion is supported by many tools such as OOMeter [29] and Borland Together [12].

Following the original definition of Tight Class Cohesion, we defined a new metric for evaluating component cohesion and name it TCC (Tight Component Cohesion). TCC is defined as the ratio of the number of interface operation pairs which share a same implementation class to the total number of interface operation pairs within a component. The detailed definition of TCC is as following:

Let RCcmp(op) be a set of classes in a component cmp which are referenced by an interface operation op of cmp. Let Q(cmp) be a set of interface operation pairs which share the same implementation class in component cmp:

\[ Q(cmp) = \{(op_i, op_j)|RC_{cmp}(op_i) \land RC_{cmp}(op_j) \neq \emptyset \land op_i, op_j \in IOP(cmp) \land i \neq j\} \]

Let R(cmp) be a set of total interface operation pairs:

\[ R(cmp) = \{(op_i, op_j)|op_i, op_j \in IOP(cmp) \land i \neq j\} \]

Then TCC can be defined as follows:

\[ TCC(cmp) = \frac{|Q(cmp)|}{|R(cmp)|} \]

As an example, in Fig. 3, considering cmp2, suppose RCcmp2(op41) = \{cls6\}, RCcmp2(op42) = \{cls6, cls7\}, RCcmp2(op61) = \{cls7\}, then 

\[ Q(cmp2) = \{(op_{41}, op_{42}), (op_{42}, op_{61})\}, R(cmp2) = \{(op_{41}, op_{42}), (op_{41}, op_{61}), (op_{42}, op_{61})\}, \text{ so } TCC(cmp2) = \frac{|Q(cmp2)|}{|R(cmp2)|} = 2/3 = 0.67 \]

Concerning cmp3, because cmp3 does not have any interface operation, |R(cmp3)| = 0, therefore, TCC(cmp3) = 0.

If TCC(cmp) = 1, then all of the interface operations are connected with each other through implementation class. In contrast, if all the interface operations do not connect with each other through implementation classes, then TCC(cmp) = 0. A larger TCC value indicates that more similar functions are grouped together and thus more strongly related implementation classes belong together; therefore, it is more preferred for component reusability and maintainability.

### 3.3 Subject Systems

A large number of open-source software systems in the software community are good candidates for conducting the experiments. In order to reduce the chance of subjectivity during this empirical study, we have considered eight
subject systems which are developed by experienced programmers and most of which have experienced many times of enhancement: RSSOwl [18], SableCC [19], DesmoJ [20], Rhino [21], JavaCC [22], Jaga [23], LBFSrv [24], and DOT-Parser [25]. These subject systems vary in application domains and their sizes of source code. For simplicity, each system is given a unique ID. Table 2 gives the details of the subject systems.

4. Experimental Results

In this section, we present the experimental results of a series of experiments which are carried out at overall and detailed levels. The purpose of the experiments is to examine if existing component identification approaches produce components of good quality. The experimental results for three component identification approaches are evaluated by using the proposed coupling, cohesion and size criteria. Section 4.1 describes the overall experimental results. Section 4.2 presents the overall evaluation results where three approaches are executed by using recommended threshold values given by the authors. Section 4.3 presents detailed evaluation results where three approaches are executed by gradually changing their threshold values. Section 4.4 summarizes the experimental results and presents our observations. Section 4.5 discusses threats to validity that affect this empirical study.

4.1 Overall Experimental Results

We have utilized the tool CIETool to extract necessary information and applied three component identification approaches to identify components from eight subject systems. Figure 4 shows the distributions of the numbers of implemented operations of classes of eight subject systems. All classes in the subject systems are considered together. In Fig. 4, the x-axis represents size ranges of implemented operations owned by classes, and the y-axis represents ratios of classes according to each range. As can be seen, 42.8% of the classes in all the subject systems have 1 to 5 implemented operations. Only 3.4% of the classes have over 40 implemented operations.

Figure 5 shows the numbers of components produced by all three component identification approaches for each subject system. In Fig. 5, eight subject systems are placed in the x-axis and the y-axis represents the number of identified components. Note that the number beside the system label indicates the number of classes in the system. For example, S1(196) represents that the subject system S1 consists of 196 classes.

The experimental results show that three approaches produce quite different numbers of components for each individual subject system. For example, Fig. 5 shows that even though S1 is larger than S2 by 3 classes, Lung’s approach produces 46 more components than Lee’s approach for S1. In contrast, five more components are identified by Lee’s approach than Lung’s approach for S2. In could be observed that the Bunch approach generally produces the smallest number of components for all subject systems. For example, in Fig. 5, system S1, which has 196 classes in total, is grouped into 106 components by Lung’s approach and 60 components by Lee’s approach, but only 9 by the Bunch approach.

4.2 Overall Evaluation Results

In this subsection, we present overall evaluation results for the three approaches. The identified components from all subject systems are collectively evaluated using the proposed size, coupling and cohesion criteria.

4.2.1 Coupling and Cohesion

In this study, the coupling of identified components is measured by the CCR metric and cohesion is measured by the TCC metric. These three component identification approaches produce components with different coupling and cohesion values. The results are discussed below.

Figure 6 shows the distributions of CCR values for identified components by each approach. The x-axis represents ranges of CCR values and the y-axis represents ratios of identified components according to each range. As can be seen from Fig. 6, only 44.5% of components are found to be loosely coupled (CCR ≤ 0.5), while the other 55.5% are relatively

![Fig. 4](image-url) The distribution of the numbers of implemented operations of classes.

![Fig. 5](image-url) The number of identified components.
tightly coupled ($CCR > 0.5$). In contrast, Lung’s and Lee’s approaches produce many loosely coupled components. As can be seen, the largest ratios of identified components for Lung’s and Lee’s approaches appear in the range of loose coupling $[0, 0.1]$; that is, they account for 54.9% and 45.7% respectively.

Figure 7 shows the distributions of $TCC$ values for the identified components by each approach. The x-axis represents ranges of $TCC$ values, and the y-axis represents ratios of identified components according to each range.

The overall experiments show that all three approaches produce poor components from the perspective of cohesion; most identified components are very weakly cohesive. As can be seen from Fig. 7, the largest ratios of identified components by all three approaches appear in the range $[0, 0.1]$ and they are 53.1%, 51.3%, and 41.7%, respectively.

In addition, both of coupling and cohesion are investigated by considering the $CCR$ and $TCC$ values of the identified components. Figure 8 shows the distributions of identified components according to their $CCR$ and $TCC$ values for three approaches. The x-axis represents $CCR$ values and the y-axis represents $TCC$ values of identified components.

As seen in Fig. 8 (a) and Fig. 8 (b), by Lung’s and Lee’s approaches, many components have small $CCR$ (coupling) values and small $TCC$ (cohesion) values. This means that Lung’s and Lee’s approaches produce many loosely coupled but weakly cohesive components. Figure 8 (c) shows that the Bunch approach produces many components of large $CCR$ (coupling) values and small $TCC$ (cohesion) values. In other words, the Bunch approach produces many tightly coupled and weakly cohesive components.

### 4.2.2 Component Size and Interface Size

Component has two basic size properties: component size and interface size. In this experiment, the numbers of implementation classes ($NOIC$) and interface operations ($NOIO$) are used for evaluating the identified components.

The $NOIC$ and $NOIO$ results show that all of these three approaches have different effects on sizes of components and their interfaces of identified components, and they all produce a number of components with inappropriate component size or interface size.

Figure 9 shows the ratios of the identified components according to their sizes in each range by each approach. As can be seen, Lung’s and Lee’s approaches tend to produce components of small sizes; it is evident that about 95% of components by Lung’s and Lee’s approaches consist of less than 10 implementation classes. In contrast, the Bunch approach tends to produce components of large sizes; it is evident that about 50% of components consist of more than 10 implementation classes.

In addition, as shown in Fig. 9, 57.3% and 71.2% of
components have a single class by Lung’s and Lee’s approaches, respectively. By the Bunch approach, 11.1% of components have a single class. Although it is difficult to determine appropriate sizes for components, a component with a single class is not desirable because it suggests that this class is never involved in the clustering process.

Figure 10 shows the ratios of the identified components which are grouped into different interface size ranges by each approach. As can be seen, by Lung’s and Lee’s approaches, most components have 1 to 5 interface operations. In contrast, about 40% of the components produced by the Bunch approach have more than 20 interface operations.

It also can be observed from Fig. 10 that these three approaches even produce components with no interface operations; 22.6% for Lung’s approach, 12.0% for Lee’s approach, and 13.9% for the Bunch approach. In other words, more than 10% of the produced components have no operations which are invoked from other components. Because components should expose their functions as interface operations, having no interface operations indicates that those components are not actually good ones.

4.3 Detailed Evaluation Results

The overall experiments show that three approaches produce components of poor quality which are loosely cohesive, tightly coupled, or have inappropriate sizes of components and their interfaces. Three approaches have proposed different thresholds to control their clustering processes. For further investigation, we carried out more detailed experiments for three approaches at different threshold levels. The detailed experiments aim to examine if adjusting the threshold value is an effective method for comprehensively improving the quality of identified components. In the detailed experiments, the identified components from eight subject systems are collectively evaluated from the aspects of coupling, cohesion and size at different threshold levels.

4.3.1 Lung’s Approach

In Lung’s approach, the similarity threshold $h$ is used to control the clustering process. If a very small threshold is defined, the clustering process will not terminate until all classes are almost grouped into one component; otherwise, if a very large threshold is used, the clustering process will terminate at a very early stage. We carried out the detailed experiment for Lung’s approach with $h$ gradually ranging from 1 to 0.1.

Figure 11 shows average $CCR$ (coupling) and $TCC$ (cohesion) values of the identified components with $h$ ranging through different threshold levels for Lung’s approach. As can be seen, $TCC$ is proportional to $h$, while $CCR$ is inversely proportional to $h$. That is, the components with a higher $h$ value are more cohesive and less coupled than those of a lower $h$ value.

The $CCR$ and $TCC$ evaluation results show that the cohesion of identified components is low at all threshold levels, even though coupling and cohesion of the identified components can be varied by adjusting the threshold value. As shown in Fig. 11, at all threshold levels the average $TCC$ values are smaller than 0.4.

Figure 12 shows the ratios of identified components in different ranges of component size with the $h$ value ranging from 1 to 0.1. As can be seen, the sizes of components have different ranges at each threshold level. For example, the ratio of components with $NOIO > 10$ is close to 0 when $h = 1$, but it is around 20% when $h = 0.1$.

The $NOIC$ evaluation results show that adjusting $h$ cannot avoid components with inappropriate numbers of implementation classes, even though the ratios of components in different component size ranges are varied at different threshold levels. As shown in Fig. 12, the largest ratios of the identified components appear in the range $NOIC = 1$ at all threshold levels. Although at $h = 0.4$ Lung’s approach produces the smallest ratio of components with single implementation class, it is still over 50%.

Figure 13 shows the ratios of identified components
with interface sizes in different ranges with $h$ ranging from 1 to 0.1. As can be seen, the ratios of interface size in each range are different at different threshold levels. For example, the largest ratio of components appears in the range $[1, 5]$ when $h = 1$, but appears at $NOIO = 0$ when $h = 0.1$.

The $NOIO$ evaluation results show that components with inappropriate numbers of interface operations cannot be avoided by adjusting $h$, although the ratios of components in different interface size ranges are varied at different threshold levels. As can be seen from Fig. 13, components with no interface operations ($NOIO = 0$) can be found at all threshold levels and the smallest ratio is even over 15%, appearing at $h = 1$.

In Lung’s approach, the detailed experiment shows that even though $h$ may be an effective method for improving some aspects of the quality of identified components, it cannot meet all component criteria at the same time. As can be seen from the above results, when $h$ is high, Lung’s approach tends to produce many single class components. When $h$ is low, the ratio of components with no interface operations becomes large. Through all threshold levels, identified components are weakly cohesive.

4.3.2 Lee’s Approach

In Lee’s approach, $MAXCOX$ is a complexity threshold used to control the clustering process; that is, all identified components should have a lower complexity than $MAXCOX$. We use a same set of basic components to carry out the detailed experiment for Lee’s approach. The value of $MAXCOX$ is set as a different percentage of the maximum complexity value of identified components. The detailed experiments are carried out with $MAXCOX$ ranging from 100% to 0%.

Figure 14 shows the average $CCR$ and $TCC$ values with $MAXCOX$ gradually ranging through different threshold levels for Lee’s approach. As can be seen, there are very few changes for both $CCR$ and $TCC$ values through the whole threshold levels.

The average $CCR$ and $TCC$ values show that adjusting $MAXCOX$ cannot avoid components of weak cohesion. As can be seen from Fig. 14, the average $TCC$ values are below 0.32 through all threshold levels; that is, many of the identified components are weakly cohesive, which are recognized to be of poor quality for component-based systems.

Figure 15 shows the ratios of identified components with their sizes in different ranges with $MAXCOX$ ranging from 100% to 0% for Lee’s approach. We can see that component sizes of identified components range variously at each threshold level, but there are very small changes through all threshold levels.

The $NOIC$ evaluation results show that one cannot avoid components with inappropriate component sizes by adjusting $MAXCOX$. We can see from Fig. 15 that through all threshold levels the largest ratios of components simultaneously appear in the range $NOIC = 1$; that is, they account for around 70% of the identified components.

Figure 16 shows the ratios of the identified components with interface size in different ranges with $MAXCOX$ ranging from 100% to 0% for Lee’s approach. As can be seen, interface sizes of identified components also show different ranges at each threshold level, but there are very small changes among different threshold levels.

The $NOIO$ evaluation results also show that adjusting $MAXCOX$ is not effective to avoid components with inappropriate interface sizes. As can be seen from Fig. 16, Lee’s approach produces components with no interface operations and the ratios remain around 12% through all threshold lev-
els.

For Lee’s approach, we can see that adjusting threshold value is not applicable to avoid components of poor quality. Because the clustering process is based on a set of basic components and some of which have reached a certain level of complexity, many classes are difficult to be involved in the clustering process. The components produced by Lee’s approach have either inappropriate component size or interface size at all threshold levels, and their average cohesion values are low.

4.3.3 The Bunch Approach

The Bunch approach provides a parameter popSz to let users decide the number of components to be produced. In the Bunch tool, a clustering option is defined to assist users to cluster classes at varying levels of detail instead of specifying popSz by users; that is, Bunch can be executed at three threshold levels: detailed-level (L0), median-level (L1) and top-level (L2). At detailed-level the Bunch approach tends to produce components of small sizes, and at top-level it tends to produce components of large sizes.

Figure 17 shows the average CCR and TCC values of identified components at different threshold levels for the Bunch approach. As can be seen, components produced at different threshold levels have different coupling and cohesion values. CCR is the smallest at detailed-level (L0) and the largest at top-level (L2). By contrast, TCC is the largest at detailed-level (L0) and the smallest at top-level (L2).

The average CCR and TCC values show that the Bunch approach cannot avoid components of poor quality which are either tightly coupled or weakly cohesive by adjusting threshold levels. As shown in Fig. 17, at all threshold levels, the average TCC values are below 0.35; at median-level (L1) and top-level (L2), the average CCR values are nearly 0.6.

Figure 18 shows the ratios of identified components according to their component sizes in different ranges at different threshold levels. As can be seen, the ratios of identified components range variously at all three threshold levels. For example, the largest ratio of components appears in the range [2, 5] at detailed-level (L0), but in the range NOIC > 10 at median-level (L1) and top-level (L2).

The NOIC results show that one cannot avoid components with inappropriate component sizes at any threshold level, even though the ratios of NOIC are varied at different threshold levels. We can see from Fig. 18 that components with single a class can be found at all three levels and they account for 3%, 11% and 22%, respectively.

Figure 19 shows the ratios of identified components with their interface sizes in different ranges at different threshold levels. As can be seen, the ratios of components with different interface sizes have different ranges at three threshold levels. The largest ratios of interface sizes at three threshold levels appear in the ranges [1, 5], [11, 20] and > 20, respectively; that is, they account for 33.3%, 40.3% and 48.6% of identified components.

The NOIO results show that components with no interface operations are produced only at the detailed-level (L0). However, at the median-level (L0) and the top-level (L2), interface sizes become considerably larger; it is evident that around 50% and 70% of components have more than 10 interface operations at the median-level (L1) and the top-level (L2).

The detailed experiment shows that the Bunch approach also cannot avoid producing components of poor quality by adjusting threshold values. At the detailed-level (L0), the Bunch approach produces components with no interface operations. At the median-level (L1) and the top-level (L2), even though the Bunch approach does not produce components with no interface operations, many of them are tightly coupled. In addition, a large ratio of components are weakly cohesive and components with one single class can be found at all threshold levels.

4.4 Discussions

In this empirical study, we found that existing approaches commonly produced components of poor quality which were weakly cohesive, tightly coupled, or had inappropriate
Table 3  Summary of overall experimental results of three approaches.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Lung’s Approach</th>
<th>Lee’s Approach</th>
<th>Bunch Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCR (Coupling)</td>
<td>About 4% of components are over 0.5, and less than 1% are equal to 1.</td>
<td>About 3% of components are over 0.5, and less than 1% are equal to 1.</td>
<td>About 56% of components are over 0.5, and 10% are equal to 1.</td>
</tr>
<tr>
<td>TCC (Cohesion)</td>
<td>About 78% of components are below 0.5, and even 44.6% are equal to 0.</td>
<td>About 73% of components are below 0.5, and even 45% are equal to 0.</td>
<td>About 93% of components are below 0.5, and even 19% are equal to 0.</td>
</tr>
<tr>
<td>NOIC (Component Size)</td>
<td>Average component size is 3, and around 57% have single class.</td>
<td>Average component size is 3, and around 72% have single class.</td>
<td>Average component size is 12, and around 11% have single class.</td>
</tr>
<tr>
<td>NOIO (Interface Size)</td>
<td>Average interface size is less than 1, and around 22% have no interface operations.</td>
<td>Average interface size is less than 1, and around 12% have no interface operations.</td>
<td>Average interface size is 41, and around 14% have no interface operations.</td>
</tr>
</tbody>
</table>

Fig. 20  The average CCR and TCC values of each subject system.

sizes of components and their interfaces with respect to the proposed criteria. In particular, adjusting threshold values also cannot avoid those components of poor quality. The experiments also revealed that although poor-quality components can be identified in most subject systems, the numbers and types of them are varied by the component identification approaches, which indicates that if an appropriate component identification approach is employed for a subject system, the components of poor quality can be reduced or eliminated. Table 3 summarizes the overall experimental results of three component identification approaches by coupling, cohesion, component size, and interface size.

4.4.1 Coupling and Cohesion

As can be seen from Table 3, a large number of components produced by these three approaches are weakly cohesive or tightly coupled. Figure 20 presents a more detailed view of the results. Figure 20 (a) shows that for all of the subject systems, the CCR values by the Bunch approach are obviously higher than those by the other two approaches. In addition, Fig. 20 (b) shows that for most of the subject systems, the TCC values by these three approaches are lower than 0.5.

The evaluation results show that the Bunch approach tends to produce many highly coupling components, while Lung’s approach and Lee’s approach produce much fewer. This can be explained because they use different types of clustering approaches. The hierarchical clustering approaches used in Lung’s approach and Lee’s approach perform a bottom-up clustering process. They iteratively merge small components into larger ones using similarity measurements as criteria. However, the partitioning approach used in the Bunch approach starts by generating a random partition of the subject system and then optimizes it using an objective function. This increases the possibility that tightly coupled classes are placed in different components, and thus causes the overall coupling increased. In addition, as shown in Fig. 20 (a), though the average CCR values by the Bunch approach are higher than those by the other two approaches for the subject system $S_8$, the differences are not so large. This is perhaps due to the relatively small size of $S_8$.

TCC criterion considers that classes having more similar functions are more cohesive. Those classes are generally dependent on the same classes. The major reason that causes low cohesion is that the employed similarity measurements in all of these three approaches do not consider the direction of invocation between classes, so the possibility that classes have similar functions are placed in the same components is reduced. In addition, current similarity measurements just consider direct accesses between classes. Indirect interaction also indicates some kind of cohesion between classes, which is rarely considered in existing approaches. In addition, Fig. 20 (b) shows that the subject system $S_7$ produced more cohesive components than most of the other systems by three approaches. This can be explained by the fact that a large number of classes in $S_7$ invoked the same class and they were all identified in the same component, which caused the overall cohesion value remarkably increased.

4.4.2 SCCs and NIocs

Considering size criteria, Table 3 suggests that two types of poor-quality components widely exist in the identified com-
components: components with a single class, and components with no interface operations. For simplicity, we call them \textit{SC C} (Single Class Component) and \textit{NIO C} (No Interface Operation Component), respectively. \textit{SC C} indicates that a class is never involved in the process of clustering by the employed component identification approaches. \textit{NIO C} is considered of poor quality because it does not exhibit any usability to users. Both of them are indicators of inappropriate determinations on the employed component identification approaches. Figure 21 gives a more detailed view of the results of \textit{SC Cs} and \textit{NIO Cs}. As shown in Fig. 21 (a), \textit{SC Cs} exist in most of the subject systems. In particular, Lung’s approach and Lee’s approach identify a large ratio of \textit{SC Cs}. In addition, in Fig. 21 (b), we can see that \textit{NIO Cs} are also identified by the component identification approaches for most of the subject systems.

\textit{SC Cs} may be caused by two reasons: isolated classes and underestimation of usage dependencies. Generally isolated classes are identified as \textit{SC Cs} because they do not have usage dependencies with other classes. However, isolated classes are not necessarily dead code. For example, a software framework which aims to provide generic functionalities generally consists of a lot of isolated classes. Libraries and APIs also consist of many loosely coupled classes. Four out of the eight subject systems (S1, S3, S5 and S6) have this type of \textit{SC Cs} and they are identified by all of these three approaches. This is because the employed similarity measurements only focus on formal features such as interaction relationships extracted from source code, which are not sufficient for software systems. A more comprehensive similarity measurement which considers non-formal features such as file names and source code authors is expected for isolated classes. For example, dis-similarity measurement, which is the opposite of similarity measurement, may be a good candidate to deal with isolated classes.

In another case, dependencies among classes are recognized by component identification approaches, but \textit{SC Cs} may also be identified. That is because of underestimating usage dependencies, or in other words, types of usage dependencies are not differentiated. Both Lung’s approach and Lee’s approach identified many components of this type. As shown in Fig. 22, components \textit{cmp}$_1$ and \textit{cmp}$_3$ consist of two classes, respectively. The component \textit{cmp}$_2$ is \textit{SC C}. \textit{cmp}$_2$ connects with \textit{cmp}$_1$ through a composition relationship and \textit{cmp}$_3$ connects with \textit{cmp}$_1$ through an association relationship. From the perspective of object-oriented system, the composition relationship is a stronger relationship than association. But without considering this factor, \textit{cmp}$_1$ may be combined with \textit{cmp}$_3$ instead of \textit{cmp}$_2$. Underestimating usage dependencies may cause that single class components can not be combined into larger components at earlier clustering stage. Although Lee’s approach has taken it into account by giving weights to various usage dependencies, they are usually affected by subjective factors, such as various perspectives of software engineers. The Bunch approach performs better than the other two approaches. It does not identify any components of this type. This is caused by the inherent characteristics of the employed clustering approach in the Bunch approach.

A detailed investigation suggests that the identified \textit{NIO Cs} can be categorized into the following three types: activation components, non-activation components and autonomous components. But not all of them are necessarily of poor quality, such as activation components. Some components are inherently not invoked by other components, but their functionalities are implemented depending on other components. They are invoked from the environment such as users, devices, or external systems over a network. We call them activation components. Activation components are generally inevitable, and actually necessary in a component-based system. For example, a component containing the main function is an activation component. Activation components have been identified by all three approaches.

Some components also do not provide interface operations, but depend on other components. However, they are not inherently invoked from the environment. A significant difference between non-activation components and activation components is that non-activation components do not
have a main function. One feasible method is to merge non-activation components into a tightly coupled normal component which owns interface operations. In addition, some components do not have any usage dependency with other components. They are called autonomous components. These components execute some functions of the system independently of other components. All of the three approaches produced this type of component. More non-formal features should be considered for dealing with autonomous components, thus, they can be merged into other components.

As shown in Fig. 21 (b), there is no an obvious trend for three component identification approaches to produce NIOCs, whereas there is a strong dependency for NIOCs on SCCs. Figure 23 shows that a large ratio of NIOCs is SCCs in the meantime. Thus it is expected that NIOCs can be reduced with the reduction of SCCs.

4.5 Threats to Validity

This subsection discusses threats to validity that affect this empirical study. The external validity is the degree to which results from the empirical study can be generalized in the population. Our empirical study aims to examine if existing component identification approaches produce components of good quality for object-oriented legacy systems. Three selected component identification approaches have covered major similarity measurements and clustering approaches, and they have been tested on real legacy systems, which reduce this threat to external validity. In addition, the subject systems are also representative of the population where the result should be generalized. They are derived from the open-source software community and developed by experienced programmers. 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 that the size and structure of the subject systems to be representative of typical legacy systems. Keeping this in mind, our subject systems were chosen from different application domains. However, they are small to moderate in size.

The threats to construct validity could be present in the proposed evaluation criteria. Our investigation suggests that existing component metrics such as coupling and cohesion are not widely applied in the industry, but there exist several class coupling and cohesion metrics which have been justified and widely applied, such as CBO (Coupling) and TCC (Cohesion). In order to mitigate the threats to construct validity, we proposed a new component coupling metric CCR and cohesion metric TCC conforming to the original conceptions of coupling and cohesion metrics for classes. In addition, component size and interface size are the most basic features of components. They can be used to effectively depict the characteristics of components. Although it is difficult to determine the exact numbers of component size and interface size, a component with a single class or with no interface operations is evidently undesirable.

The internal validity threats are relevant for the experiments presented in this section as we aim at concluding that existing component identification approaches produce many components of poor quality. The internal validity threats are mitigated by the experiment design. We have carried out a series of experiments at overall and detailed levels by using recommended threshold values and gradually adjusting those threshold values. In order to uniformly and consistently conduct these experiments we have developed a CI-ETool to facilitate the process of component identification and evaluation. Even though different component identification approaches produce different sets of components at different threshold levels, their evaluation results are very similar; that is, components of poor quality are universally produced by all three approaches in overall and detailed experiments.

5. Related Work

So far a number of component identification approaches have been presented in the literature. Each of them employs different similarity measurements and clustering approaches. Wiggerts et al. [9] presented an overview of software clustering techniques. They performed a detailed analysis on similarity measurements and existing clustering algorithms, and those studies were compared on the concept level. They stated that determining the input of the clustering process and specifying the criteria for good clustering would depend heavily on the software environment used. Lakhotia [26] proposed a united framework for expressing component identification approaches, which was comprised of consistent terminology, notation, and symbols. The investigation of subsystem classification techniques stemmed from the interest in developing automated support for identifying modules as a precursor to reengineering legacy source code. Given the differences in the problem domains, the subsystem classification techniques had been presented using different terminology and symbols. They implemented several existing approaches based on their framework, but did not provide experimental comparison or evaluation.

There were several component metrics proposed to qualify various characteristics of components. Washizaki et al. [6] proposed a metrics suite for measuring the reusability
of components based on the limited static information that can be obtained from the outside of components without any source code. They do not use cohesion and coupling criteria because these criteria usually need metrics that require the analysis of the source code of components. Choi et al. [7] proposed a component coupling metric by considering both static and dynamic relationships between classes. They considered different call types between methods - such as create, delete, write and read - and gave different weights to them. Tzerpos et al. [10] proposed a metric, called Mojo, which can be used to compare two different decompositions of a software system. When Mojo is used to evaluate the quality of identified components, an authorized decomposition of the subject system is necessary, which is difficult to acquire in practice.

Experiments for evaluating component identification approaches have been conducted by researchers. Luo et al. [5] performed an experimental study of two graph analysis based component identification approaches: the bottom-up approach and the top-down approach. They focused on the comparison of three aspects of the two methods using different criteria: the reusability of identified components, the capability of architecture recovery, and the execution time. The experimental study shows that the bottom-up approach is better than the top-down approach in terms of component capture capability, and the bottom-up approach is more promising for recovering system architectures as it can provide hierarchical structures and a better partition. In [4], Davey et al. presented their studies on data clustering techniques for software re-modularization. They argued that different clustering techniques behave differently when applied to different types of software systems. A technique might impose an artificial structure on the existing system instead of bringing out the natural one.

6. Conclusions

Component identification technology is a critical part of software reengineering. It has been used to facilitate legacy systems to be reconfigured into fine organized component-based systems. Understanding the behaviors of various component identification approaches is the first important step to meaningfully employ them for legacy systems evolution, so we conducted an empirical study on component identification technology. We performed experiments and evaluations with three different component identification approaches for eight subject systems with a range of different application domains and sizes of source code.

From the experimental results, we found that these three approaches produced many components of poor quality; that is, many of the identified components were tightly coupled, weakly cohesive, or had inappropriate numbers of implementation classes and interface operations. These poor-quality components cannot be avoided by adjusting the threshold values of component identification approaches. The experiments also revealed that although components of poor quality can be identified in most subject systems, the numbers and types of them were varied by the component identification approaches. Our investigation suggested that few poor-quality components were caused by the software systems themselves, whereas weaknesses in the component identification approaches are the major reason. The experimental results could be beneficial for future researchers in this area.

In this empirical study, we have revealed deficiencies in existing component identification approaches. In order to improve existing component identification approaches and enable them to give better results, we should do more research into the coupling mechanisms of object-oriented systems. There are many different mechanisms that can establish usage dependencies between classes such as import coupling, export coupling, inheritance coupling. Limited by the application environment and time cost, a thorough consideration of all coupling mechanisms is generally impractical for a component identification approach. More research efforts are expected on the subject of applying appropriate component coupling mechanisms to component identification technology.

Acknowledgments

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract.

References

[22] JavaCC, https://javacc.dev.java.net/

HeungSeok Chae received the BS degree in nuclear engineering from Seoul National University in 1994, the MS and PhD degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1996 and 2000, respectively. Since 2004, he has been on the faculty of the Department of Computer Science and Engineering, Pusan National University, Busan, Korea. His current research interests include object-oriented analysis and design, object-oriented frameworks, component-based software development, software metrics, middleware architecture for availability and scalability.

JianFeng Cui received the MS degree in software engineering from Nankai University, China, in 2004, and the PhD degree from Pusan National University, Korea, in 2010. Since 2010, he has been on the faculty of the Department of Computer Science and Technology, Xiamen University of Technology, China. His current research interests include mobile agent technology for distributed system, component-based software development and software reengineering.