A Unified Framework for Cohesion Measurement in Object-Oriented Systems[1]

Lionel C. Briand
Fraunhofer IESE
Sauerwiesen 6
D-67661
Kaiserslautern
Germany
briand@iese.fhg.de

John W. Daly
Fraunhofer IESE
Sauerwiesen 6
D-67661
Kaiserslautern
Germany
daly@iese.fhg.de

Jürgen Wüst
Fraunhofer IESE
Sauerwiesen 6
D-67661
Kaiserslautern
Germany
wuest@iese.fhg.de

Abstract

The increasing importance being placed on software measurement has led to an increased amount of research developing new software measures. Given the importance of object-oriented development techniques, one specific area where this has occurred is cohesion measurement in object-oriented systems. However, despite an interesting body of work, there is little understanding of the motivations and empirical hypotheses behind many of these new measures. It is often difficult to determine how such measures relate to one another and for which application they can be used. As a consequence, it is very difficult for practitioners and researchers to obtain a clear picture of the state-of-the-art in order to select or define cohesion measures for object-oriented systems.

To help remedy this situation a unified framework, based on the issues discovered in a review of object-oriented cohesion measures, is presented. The unified framework contributes to an increased understanding of the state-of-the-art as it is a mechanism for (i) comparing measures and their potential use, (ii) integrating existing measures which examine the same concepts in different ways, and (iii) facilitating more rigorous decision making regarding the definition of new measures and the selection of existing measures for a specific goal of measurement.

Keywords: cohesion, object-oriented, measurement.

1. Introduction

The market forces of today's software development industry have begun to place much more emphasis on software quality. This has led to an increasingly large body of work being performed in the area of software measurement, particularly for evaluating and predicting the quality of software. In turn, this has led to a large number of new measures being proposed for quality design principles such as cohesion. Modules of a high quality software design, among many other principles, should obey the principle of high cohesion. Stevens *et al.*, who first introduced cohesion in the context of structured development techniques, define cohesion as a measure of the degree to which the elements of a module belong together. In a highly cohesive module, all elements are related to the performance of a single function. Such modules are hypothesised to be easier to develop, maintain, and reuse, and to be less fault-prone. Some empirical evidence exists to support this hypothesis for systems developed with structured and object-based techniques; see, e.g., [8], [9], and [6].

In object-oriented software, classes replace modules, with methods and attributes as their elements. In this context, cohesion is the degree to which the methods and attributes of a class belong together. Again, recent research has led to a large number of new cohesion measures for object-oriented systems. However, because cohesion is a complex software attribute in object-oriented systems (e.g., there are several different mechanisms which are considered to contribute to the cohesion of a class), and there has been no attempt to provide a structured synthesis, our understanding of the state-of-the-art is poor. For example, because there is no standard terminology and formalism for expressing measures, many measures are not fully operationally defined, i.e., there is some ambiguity in their definitions. As a result, it is difficult to understand how different cohesion measures relate to one another. Moreover, it is also unclear what the potential uses of many existing measures are and how these different measures might be used in a complementary manner. In addition, the fact that there exists little empirical validation of existing object-oriented cohesion measures means their usefulness is not supported by any empirical evidence[2].

To address and clarify our understanding of the state-of-the-art of cohesion measurement in object-oriented systems

1. A more detailed version of this paper is available as an International Software Engineering Research Network technical report (ISERN-97-05), which is available on the ISERN website: http://www.iese.fhg.de/ISERN/pub/isern_biblio_tech.html.

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

requires a comprehensive framework based on a standard terminology and formalism. This framework can then be used to facilitate comparison of existing cohesion measures, and to support the definition of new cohesion measures and the selection of existing ones based on a particular measurement goal. Analogous research for coupling measurement is described in [1]. The coupling framework presented in that paper is considered to be complementary to the cohesion framework presented here.

2. Motivation

Object-oriented measurement has become an increasingly popular research area. This is substantiated by the fact that recently proposed in the literature are (i) several different frameworks for coupling and cohesion and (ii) a large number of different measures for object-oriented attributes such as coupling, cohesion, and inheritance. While this is to be welcomed, there are several negative aspects to the mainly ad hoc manner in which object-oriented measures are being developed. As neither a standard terminology or formalism exists, many measures are expressed in an ambiguous manner which limits their use. This also makes it difficult to understand how different measures relate to one another. For example, there are many different decisions that have to be made when defining a cohesion measure - these decisions have to be made considering the measurement goal and by defining an empirical model based on clearly stated hypotheses. Unfortunately, many of the measures proposed in the literature are not the result of clearly documented decisions and hypotheses. It is therefore often unclear what the potential uses of existing measures are and how different cohesion measures could be used in a complementary manner to obtain a more detailed picture of the cohesion of classes in an object-oriented system.

Several authors have introduced different approaches and proposed measures to characterise cohesion in object-oriented systems, e.g., [10], [11], [15], [3], [14], [16], [5], [6]. Eder *et al.* define a framework aimed at providing qualitative criteria for cohesion; they also assign relative strengths to different levels of cohesion they identify within this framework [12]. However, neither this framework nor the different approaches used have characterised existing measures to the different dimensions of cohesion that have been identified. Therefore, the negative aspects highlighted above are still very prevalent ones. In our review of the literature, for example, we found 15 different measures of object-oriented cohesion. Consequently, it is not difficult to imagine how confusing the overall picture actually is.

To make a serious attempt to improve our understanding of object-oriented cohesion measurement we have to integrate all existing approaches into a unique theoretical framework, based on a homogenous and comprehensive formalism. A review of existing measures has to be performed and these measures have to be categorised according to the unified framework. This framework will then be a mechanism with which to compare measures and their potential use, and allow more rigorous (and ease of) decision making regarding the definition of new measures and the selection of existing measures in the context of a measurement goal. It should also facilitate the evaluation and empirical validation of cohesion measures by ensuring that specific hypotheses are provided which link cohesion measures to external quality attributes. Finally, it should also help identify the dimensions of cohesion which thus far have been overlooked, i.e., for which there are no measures defined.

3. Survey of cohesion measurement approaches and measures

In this section we perform a comprehensive survey and critical review of existing approaches and measures for cohesion in object-oriented systems. In Section 3.1, we present the existing approaches and measures. These are then compared in Section 3.2.

3.1. Existing approaches to measure cohesion

Eder *et al.* [12] propose a framework aimed at providing qualitative criteria for cohesion. Chidamber and Kemerer [10], [11], Hitz and Montazeri [15], Bieman and Kang [3], Henderson-Sellers [14], Lee *et al.* [16], and Briand *et al.* [5], [6] each propose different approaches to measure cohesion in object-oriented or object-based systems and define various cohesion measures accordingly.

3.1.1. Framework by Eder et al. [12]

Eder *et al.* [12] propose a framework aimed at providing comprehensive, qualitative criteria for cohesion in object-oriented systems by adapting existing frameworks for cohesion in the procedural and object-based paradigm to the specifics of the object-oriented paradigm. They distinguish between three kinds of cohesion: method, class and inheritance cohesion. For each type, various degrees of cohesion are proposed.

1. *Method cohesion.* Eder *et al.* apply Myers' classical definition of cohesion [17] to methods. Elements of a method are statements, local variables and attributes of the method's class. They define seven degrees of cohesion, based on the definition by Myers [17]. From weakest to strongest, the degrees of method cohesion are:

- *Coincidental*: The elements of a method have nothing in common besides being within the same method.

2. We call a measure of an internal attribute such as cohesion *empirically validated*, if its causal impact on some external quality attribute of interest has been demonstrated [4]. Then, the measure is *useful* in the sense that it can be used as an indicator of that external quality attribute. A measure is *theoretically validated*, if it has been demonstrated that the measure is indeed measuring the attribute it is purported to measure [4]. Both empirical and theoretical validation of the measures presented here are discussed in [2].

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

- *Logical*: Elements with similar functionality such as input/output handling are collected in one method.
- *Temporal*: The elements of a method have logical cohesion and are performed at the same time.
- *Procedural*: The elements of a method are connected by some control flow.
- *Communicational*: The elements of a method are connected by some control flow and operate on the same set of data.
- *Sequential*: The elements of a method have communicational cohesion and are connected by a sequential control flow.
- *Functional*: The elements of a method have sequential cohesion, and all elements contribute to a single task in the problem domain. Functional cohesion fully supports the principle of locality and thus minimizes maintenance efforts.

2. *Class cohesion.* Class cohesion addresses the relationships between the elements of a class. The elements of a class are its non-inherited methods and non-inherited attributes. Eder *et al.* use a categorisation of cohesion for abstract data types by Embley and Woodfield [13] and adapt it to object-oriented systems. There are five degrees of class cohesion. From weakest to strongest, these are:

- *Separable*: The objects of a class represent multiple unrelated data abstractions. For instance, the cohesion of a class is separable, if the methods and attributes can be grouped into two sets such that any method of one set invokes no methods and references no attributes of the other set.
- *Multifaceted*: The objects of a class represent multiple related data abstractions. The relation is caused by at least one method of the class which uses all these data abstractions. If we interpret the attributes of a class as a relation schema (as in a relational database), the relation schema would not be in second normal form.
- *Non-delegated*: There exist attributes which do not describe the whole data abstraction represented by a class, but only a component of it. The attributes of the class interpreted as relation schema violate third normal form. Attributes describing only a component of the data abstraction should be moved in a class of their own.
- *Concealed*: There exist some useful data abstraction concealed in the data abstraction represented by the class. Consequently, the class includes some attributes and methods which might make another class.
- *Model*: The class represents a single, semantically meaningful concept.

3. *Inheritance cohesion.* Like class cohesion, inheritance cohesion addresses the relationships between elements of a class. However, inheritance cohesion takes all the methods and attributes of a class into account, i.e., inherited and non-inherited. Inheritance cohesion is strong if inheritance has been used for the purpose of defining specialized children classes. Inheritance cohesion is weak, if it has been used for the purpose of reusing code. The degrees of inheritance cohesion are the same as those for class cohesion.

The definitions of the degrees of cohesion in this framework are not amenable to operational, automated data collection, because determining the degree of cohesion of a given class or method is subjective and must be based on a semantic analysis of the class or method. The definitions should be used as guidelines to derive syntactically-based measures which are measuring approximations of these degrees of cohesion in a particular context.

3.1.2. Approach by Chidamber and Kemerer [10], [11]

Chidamber and Kemerer base their approach to define the cohesion of a class on the notion of the *degree of similarity* of the class' methods. The degree of similarity of a set M of methods is the number of attributes used in common by all methods in M, formally denoted by $\sigma(M)$. Chidamber and Kemerer argue that $\sigma(M)$ itself is not a suitable measure for cohesion of a class c: if all but one method in c use the same set A of attributes, and the remaining method only uses attributes not in A, we have $\sigma(M) = 0$, even though most methods of c are similar. Instead, Chidamber and Kemerer propose a cohesion measure LCOM defined as follows [10]:

> Consider a Class C_1 with methods M_1, M_2, ..., M_n. Let $\{I_i\}$ = set of attributes used by method M_i. There are n such sets $\{I_1\}$, ..., $\{I_n\}$. LCOM = The number of disjoint sets formed by the intersection of the n sets.

LCOM is an *inverse* cohesion measure. A high value of LCOM indicates low cohesion and vice versa. The above definition of LCOM has been interpreted in different ways by different authors. The interpretation by Hitz and Montazeri [15] will be discussed in Section 3.1.3. Henderson-Sellers offers the following interpretation [14]: $LCOM1 = \left|\{I_i \cap I_j = \varnothing \mid \forall i, j, i \neq j\}\right|$, i.e., the number of pairs of methods in class c having no common attribute references.

In the definitions of measures LCOM and LCOM1, it is not clear whether the methods of class C_1 include inherited methods or not. Also, even though it is not said explicitly, we can assume that the set I_i of attributes used by method M_i only include attributes of class C_1, or, at most, attributes that C_1 has inherited, but not of any other classes.

In [11], Chidamber and Kemerer give a new definition of LCOM:

> Consider a Class C_1 with methods M_1, M_2, ..., M_n. Let $\{I_i\}$ = set of instance variables used by method M_i. There are n such sets $\{I_1\}$, ..., $\{I_n\}$. Let
> $P = \{(I_i, I_j) \mid I_i \cap I_j = \varnothing\}$, and

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

$Q = \{(I_i, I_j) | I_i \cap I_j \neq \emptyset\}$

If all n sets $\{I_1\}, ..., \{I_n\}$ are $\emptyset$ then let $P = \emptyset$.

$$LCOM = \begin{cases} |P| - |Q|, & if \ |P| > |Q| \\ 0, & \text{otherwise} \end{cases}$$

We will refer to this version of LCOM as LCOM2. LCOM2 is the number of pairs of methods in a class having no common attribute references, $|P|$, minus the number of pairs of similar methods, $|Q|$. However, if $|P| < |Q|$, LCOM2 is set to zero. The definition of LCOM2 is almost operational. Again, it is not stated whether inherited methods and attributes are included or not, and we have to assume that sets I_i only include attributes of class C_i.

3.1.3. Approach by Hitz and Montazeri [15]

Hitz and Montazeri base their approach on the work of Chidamber and Kemerer. They interpret the definition of LCOM in [10] as follows [15]:

> Let X denote a class where I_X is the set of its instance variables and M_X is the set of its methods. Now consider a simple, undirected graph $G_X(V,E)$ with $V=M_X$ and $E = \{(m, n) \in V \times V | \exists i \in I_X$:(m accesses i) $\wedge$ (n accesses i)$\}$
>
> LCOM is then defined as the number of connected components of G_X.

We will refer to this version of LCOM as LCOM3. The above definition is almost operational. It is not stated whether inherited methods and attributes are included or excluded in the sets I_X and M_X.

Hitz and Montazeri identified a problem with *access methods* for LCOM3. An access method provides read or write access to an attribute of the class. Access methods typically reference only one attribute, namely the one they provide access to. If other methods of the class use the access methods, they may no longer need to directly reference any attributes at all. These methods are then isolated vertices in graph G_X. Thus, the presence of access methods artificially decreases the class cohesion as measured by LCOM3. There is no empirical justification for this artificial loss of cohesion. To remedy this problem, Hitz and Montazeri propose a second version of their LCOM3 measure, where graph G_X also has an edge between vertices representing methods m and n, if m invokes n or vice versa:

$E = \{(m, n) \in V \times V | \exists i \in I_X$:((m accesses i) $\wedge$ (n accesses i)) $\vee$ (m calls n) $\vee$ (n calls m)$\}$

We refer to this measure as LCOM4.

In the case where G_X consists of only one connected component (*LCOM4=1*), the number of edges $|E|$ ranges between $|V| - 1$ (minimum cohesion) and $|V| \cdot (|V| - 1)/2$ (maximum cohesion). Hitz and Montazeri define a measure C ("connectivity") which further discriminates classes having *LCOM4=1* by taking into account the number of edges of the connected component:

$$C = 2 \cdot \frac{|E| - (|V| - 1)}{(|V| - 1) \cdot (|V| - 2)}$$

We always have $C(c) \in [0,1]$. Values 0 and 1 are taken for $|E| = |V| - 1$ and $|E| = |V|(|V| - 1)/2$, respectively.

3.1.4. Approach by Bieman and Kang [3]

The approach by Bieman and Kang to measure cohesion is similar to that of Chidamber and Kemerer. They also consider pairs of methods which use common attributes. However, the manner in which an attribute may be used is different. Besides attributes used directly by a method m, indirectly used attributes are also considered. Method m uses attribute a indirectly, if m directly or indirectly invokes a method m' which uses attribute a. Two methods are called "connected", if they directly or indirectly use common attributes.

The measure TCC (tight class cohesion) is then defined as the percentage of pairs of public methods of the class which are connected, i.e., pairs of methods which directly or indirectly use common attributes.

Measure LCC (loose class cohesion) also considers pairs of "indirectly connected" methods. If there are methods $m_1,..., m_n$, such that m_i and m_{i+1} are connected for $i=1,...,n-1$, then m_1 and m_n are indirectly connected. Measure LCC is defined as the percentage of pairs of public methods of the class which are directly or indirectly connected. See [3] or [2] for formal definitions of TCC and LCC.

With respect to inheritance, Bieman and Kang state three options for the analysis of cohesion of a class.

1. exclude inherited methods and inherited attributes from the analysis, or
2. include inherited methods and inherited attributes in the analysis, or
3. exclude inherited methods but include inherited attributes.

Bieman and Kang identified a problem with constructor methods for TCC and LCC. Constructor methods provide the class attributes with initial values and therefore access most or all of the class' attributes. If *mc* is a constructor method which references all attributes of the class, then *mc* is connected to any method m which references at least one attribute of class c. That is, the presence of *mc* creates many pairs of directly connected methods. Furthermore, if m_1 and m_2 are two methods which reference at least one, but not necessarily the same, attribute of class c, then m_1 and m_2 are indirectly connected via *mc*. That is, *mc* indirectly connects any two methods which use at least one attribute. We see that the presence of a constructor method artificially

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

increases cohesion as measured by TCC and LCC, which is not empirically justified. Bieman and Kang therefore exclude constructors (and also destructors) from the analysis of cohesion [3].

3.1.5. Approach by Henderson-Sellers [14]

Henderson-Sellers sets out to define a cohesion measure having the following properties:

- The measure yields 0, if each method of the class references every attribute of the class (this situation is called "perfect cohesion" by Henderson-Sellers).
- The measure yields 1, if each method of the class references only a single attribute.
- Values between 0 and 1 are to be interpreted as percentages of the perfect value.

Henderson-Sellers proposes the following measure, referred to as LCOM5, which satisfies the above properties:

Consider a set of methods $\{M_i\}$ $(i=1,...,m)$ accessing a set of attributes $\{A_j\}$ $(j=1,...,a)$. Let the number of methods which access each attribute be $\mu(A_j)$.

$$LCOM5 = \frac{\frac{1}{a}\sum_{j=1}^{a}\mu(A_j) - m}{1-m}$$

Again, it is unclear whether inherited methods and attributes are accounted for or not.

3.1.6. Approach by Lee et al. [16]

Lee *et al.* propose a set of cohesion measures based on information flow through method invocations within a class. For a method m implemented in class c, the cohesion of m is the number of invocations to other non-inherited methods of class c, weighted by the number of parameters of the invoked methods. The more parameters an invoked method has, the more information is passed, the stronger the link between the invoking and invoked method. The cohesion of a class is the sum of the cohesion of its non-inherited methods. The cohesion of a set of classes is simply the sum of the cohesion of the classes in the set. The definitions of these measure (referred to as ICH for information-flow based cohesion) use a formalism that would be rather lengthy to reproduce here. See [16] or [2] for formal definitions of these measures.

3.1.7. Approach by Briand et al. [5], [6]

Briand *et al.* define a set of cohesion measures for object-based systems (such as Ada implementations). In the following, we adapt these measures to object-oriented systems. We make one simplification: the original measures were defined for so-called "software parts", i.e., a module or a hierarchy of nested modules. We define the adapted object-oriented measures at the class level, but do not consider nested classes. Although some programming languages allow the definition of nested classes, nesting of classes is not a major issue in object-oriented design; it can be avoided entirely through aggregation (defining attributes as an instance of another class).

For the adaption of the cohesion measures to object-oriented systems, we see a class as a collection of *data declarations* and methods. Data declarations are (i) local, public type declarations, (ii) the class itself (as an implicit, public type), and (iii) public attributes. A data declaration a interacts with another data declaration b, if a change in a's declaration or use may cause the need for a change in b's declaration or use. We say there is a *DD-interaction* between data declarations a and b, or, shorter, *a DD-interacts* with b.

Examples:

- If the definition of a public type t uses another public type t', there is a DD-interaction between t' and t.
- If the definition of a public attribute a uses a public type t, there is a DD-interaction between t and a.
- If a public attribute a is an array and its definition uses public constant a', there is a DD-interaction between a' and a.

DD-interactions need not be confined to one class. There can be DD-interactions between attributes and types of different classes. The DD-interaction relationship is transitive. If a DD-interacts with b and b DD-interacts with c, then a DD-interacts with c.

Data declarations also can interact with methods. There is a *DM-interaction* between data declaration a and method m, if a DD-interacts with at least one data declaration of m. Data declarations of methods include their parameters, return type and local variables. For instance, if a method m of class c takes a parameter of type class c, there is a DM-interaction between m and the implicit type declaration of class c.

All DD-interactions between data declarations, and DM-interactions involving parameters and return types can be determined from the class interface, and thus are available early in the development process. We define *CI(c)* (*CI* for cohesive interactions) to be the set of all such DD- and DM-interactions. *Max(c)* is the set of all possible DD- and DM-interactions in the class interface. Measure RCI (ratio of cohesive interactions) is then defined as

$$RCI(c) = \frac{|CI(c)|}{|Max(c)|}.$$

RCI ranges between 0 and 1, where values 0 and 1 indicate minimum and maximum cohesion, respectively.

At the end of the high level design phase, designers will usually have a rough idea of which interactions there exist besides those that can be determined from the class interface. Three cases are possible:

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

- Some interaction will be known to exist. We will denote the set of all known interactions by $K(c)$. Notice that $CI(c) \subseteq K(c)$.
- Some interactions may or may not exist, the available information is not sufficient at the current development stage. We denote the set of these unknown interactions by $U(c)$.
- The remaining interactions are known not to exist.

Using this additional information, we can define three more measures:

- The neutral ratio of cohesive interactions: $NRCI(c) = |K(c)|/(|Max(c)| - |U(c)|)$, (unknown interactions are not taken into account).
- The pessimistic ratio of cohesive interactions: $PRCI(c) = |K(c)|/|Max(c)|$, (unknown interactions are considered as if they were known not to be actual interactions).
- The optimistic ratio of cohesive interactions: $ORCI(c) = (|K(c)| + |U(c)|)/|Max(c)|$, (unknown interactions are considered as if they were known to be actual interactions).

3.2. Comparison of approaches

A precise comparison of the approaches shows there are differences in the manner in which cohesion is addressed. Another reason for the differences in the approaches may be the different objectives pursued by the approaches. For example, Briand *et al.* examined only early design information to investigate potential early quality indicators while other authors investigated information mainly available at low level design and implementation; hence differences are found in the mechanisms that make a class cohesive. A second reason is that some of the issues dealt with by some authors are considered to be subjective and too difficult to measure automatically. For example, the degrees of method or class cohesion (addressed by Eder *et al.*) is not something which can be easily determined automatically or even manually. The following sections discuss in detail the significant differences between the various approaches and what can be learned from these differences.

3.2.1. Types of connection

By "type of connection" we refer to the mechanisms that link elements within a class and thus make a class cohesive. In the review of cohesion measures, we can distinguish two categories:

- In the first category, we find measures focused on counting pairs of methods that use or do not use common attributes. Chidamber and Kemerer's idea of "similar" methods falls into this category; Hitz and Montazeri have reused this idea in their approach. The approach by Bieman and Kang also is based on counting pairs of methods that access common attributes.
- In the second category, measures capture the extent to which individual methods use attributes or locally defined types (LCOM5, RCI), or invoke other methods (ICH).

It is possible to have one measure count different types of connections. For instance, measures LCOM4, TCC and LCC are focused on counting pairs of methods using common attributes, and method invocations.

The ICH suite of measures are based on method invocations solely. The attributes of a class are not considered at all. This is in sharp contrast to the definitions of all other measures.

3.2.2. Domain of the measures

Most of the reviewed measures are defined at the class level. However, finer and coarser domains are also conceivable.

- For an individual attribute or method, we could count the number of other class elements to which it is connected, thus analysing how closely related the attribute or method is to other elements of its class. This could also be interpreted as the degree to which the attribute or method contributes to the cohesion of its class. From such an analysis, we could draw conclusions as to how well the attribute or method "fits" into the class, or whether it should perhaps be moved to another class.
- We can quantify the cohesion of a set of classes or the whole system based on the cohesion of each of the participating classes.

The ICH suite of measures is an example how a measure defined at the method level is scaled up to the class level and sets of classes. However, this done in a manner such that the measures are additive, which may not be a desirable property of a cohesion measure. For instance, if two *unrelated*, but *highly cohesive* classes c and d are merged into a single class e, the cohesion of the class e would be the sum of the cohesion of the separate classes c and d. That is, class e has an even higher cohesion than any of the separate classes. This is counter-intuitive, as an object of class e represents two separate, semantic concepts and therefore should be less cohesive.

3.2.3. Direct and indirect connections

Some of the approaches to measure cohesion include the analysis of indirectly connected elements. Indirect connections are of potential interest when defining criteria for when to break up a class. To illustrate this, we apply measures LCOM1 and LCOM3 to the example classes depicted in Figure 1. In the figure, a class c is represented by a graph G_c as used in the definition of measure LCOM3. The vertices are the methods of c, and there are edges between similar methods, i.e., methods which use a common attribute. This is the type of connection both LCOM1 and LCOM3 focus on. LCOM1 counts the number of pairs of methods in a class with no common attribute references. Because each class in Figure 1 has six methods and five pairs of similar methods, we have

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

LCOM1(c)=LCOM1(d), i.e., the classes are equally cohesive according to measure LCOM1. LCOM3(c) is defined as the number of connected components of graph G_c. In Figure 1, it is LCOM3(c)=1 and LCOM3(d)=2, i.e., class c is more cohesive than class d according to measure LCOM3. This reflects an important difference between classes c and d: in class c, each method is directly or indirectly connected with every other method. In class d, on the other hand, there are pairs of methods which are not even indirectly connected. This may indicate that the methods should not be encapsulated in the same class. Note that there could be reasons why the methods should be encapsulated together in one class anyway, e.g., because of method invocations from one connected component to the other.

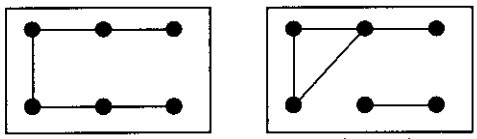

Figure 1. Example classes

Ideally, the graph G_c consists of only one connected component (LCOM3(c)=1). Hitz and Montazeri remark, that class c can still be more or less cohesive. The number of edges of graph G_c can range between $n-1$ (minimum cohesion) and $n(n-1)/2$ (maximum cohesion), where n is the number of vertices of G_c. In other words, the discriminative power of measures counting the number of connected components (such as LCOM3 or LCOM4) is limited, because a connected component can show various degrees of connectivity. Therefore, Hitz and Montazeri proposed measure C, which is a normalized count of the number of edges of G_c. Measure C can be used to further discriminate classes for which graph G_c has only one connected component. However, using two measures to completely determine the cohesion of a class has the drawback that cohesion is no longer defined on an interval scale, but only on an ordinal scale. In addition, measure C is not necessarily a better cohesion measure since it may not be possible to define classes with fully connected components.

TCC and LCC are also measures which take indirect connections into account, LCC even in two different ways. First, both measures count pairs of "connected" methods, i.e., methods which directly or indirectly use a common attribute. Method m uses an attribute a indirectly, if a is used by a method which is directly or indirectly invoked by m. Therefore, TCC and LCC take indirect method invocations into account. TCC counts the number of pairs of connected methods. It is therefore similar to measure C, which counts the number of pairs of "similar" methods. LCC counts the number of directly or indirectly connected pairs of methods, and this is the second way in which indirect connections are accounted for by LCC. This again is related to the idea of counting connected components in LCOM3 or LCOM4: Consider a graph G where vertices are methods and there are edges between connected methods. Then, "two methods m and n are indirectly connected" is equivalent to "methods m and n lie within the same connected component of graph G". The condition "each method is directly or indirectly connected to every other method" is equivalent to "graph G consists of only one connected component". A low value of LCC corresponds with a large number of connected components of G. In that respect, LCC is conceptually similar to LCOM3 and LCOM4.

LCOM5 counts for each attribute how many methods access the attribute. Only direct connections between methods and attributes are considered. In a completely cohesive class, each attribute is accessed by every method. Whether such a design is desirable is unknown.

The RCI measures are a count of interactions between elements in the class. In a completely cohesive class, each element interacts with every other element. Because the interaction relationship is transitive, there need not be a direct interaction between all pairs of elements in order to have a maximum RCI. As a consequence, RCI does not have the drawback of LCOM5 that direct interactions between all elements are required to get a maximum value.

We summarize the results of this discussion:

- Indirect connections appear to be a better criterion than direct connections when indicators for when to split up a class are needed.
- With direct connections, each element of a class needs to be directly connected to every other element in order for the class to have maximum cohesion. This appears to be an unrealistic requirement.
- Measures accounting for indirect connections are less discriminative; maximum cohesion can be attained for a larger number of classes.

3.2.4. Inheritance

For the analysis of cohesion of a class c, we have several options available concerning the attributes and methods c has inherited. Two straightforward options are:

1. exclude inherited attributes and methods from the analysis, or
2. include inherited attributes and methods in the analysis.

These two options form the distinction between class and inheritance cohesion in the framework by Eder *et al.* (see Section 3.1.1). A child class c represents an extension of its parent class d. If we exclude inherited attributes and methods, we analyse to what degree this extension represents a single semantic concept. If we include inherited attributes and methods, we analyse whether class c as a whole still represents a single semantic concept. These are two quite different aspects, and both should be considered.

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

Bieman and Kang offer a third option for the analysis of cohesion [3]:

3. include inherited attributes, but exclude inherited methods from the analysis.

Bieman and Kang do not provide any rationale for this option.

A fourth alternative would be to exclude inherited attributes but include inherited methods. This of course makes little sense, as inherited methods can only access inherited attributes.

With the exception of measures TCC, LCC and ICH, the influence of inheritance apparently has not been addressed in the definition of the reviewed cohesion measures. In the original definition of the RCI measures, inheritance is not addressed, because these measures were defined in context of object-based systems.

3.2.5. Access methods and constructors

In object-oriented design, classes usually have "access methods". An access method provides read or write access to an attribute of the class. Access methods typically reference only one attribute, namely the one they provide access to. Thus, many pairs of access methods can be built, which do not use any common attributes. This constitutes a problem for measures which count such pairs (i.e., LCOM1, LCOM2, and LCOM3).

In addition, if other methods of the class use the access methods, they may no longer need to directly reference any attributes at all. Therefore, the presence of access methods artificially decreases the class cohesion for measures based on method-attributes references. In the definitions of LCOM4 and C, this problem has been solved by adding method invocations to graph G_c (see Section 3.1.3). In the definitions of TCC and LCC, this problem is circumvented by introducing "indirectly" used attributes: if a method m invokes an access method, m indirectly uses the attributes accessed by the methods.

Constructor methods provide the class attributes with initial values and therefore access most or all of the class attributes. The presence of such a method constitutes a problem for measures counting "similar" or "connected" methods and indirect connections (LCOM3, LCOM4 and LCC). As explain in Section 3.1.4, the constructor method creates an indirect connection between any two methods which use at least one attribute, and artificially increases cohesion. Destructors are less problematic, because they do not provide attributes with values and therefore do not need to reference all attributes.

3.2.6. Summary and conclusions

From the above discussion we can see that there exists a variety of decisions to be made during the definition of a cohesion measure. It is important that decisions are based on the intended application of the measure if the measure is to be useful. When no decision for a particular aspect can be made, all alternatives should be investigated empirically. A second observation is that because the different aspects of cohesion are widely independent of each other, a large number of cohesion measures could be defined - this defines the problem space for cohesion measurement research in object-oriented systems.

4. A unified framework for cohesion measurement

In this section, a new framework for cohesion in object-oriented systems is proposed. The framework is defined on the basis of the issues identified by comparing the various approaches to measure cohesion (Section 3.2) and the discussion of existing measures. The objective of the unified framework is to support the comparison and selection of existing cohesion measures with respect to a particular measurement goal. In addition, the framework should provide guidelines to support the definition of new measures with respect to a particular measurement goal when there are no existing measures available. The framework, if used as intended, will

- ensure that measure definitions are based on explicit decisions and well understood properties,
- ensure that all relevant alternatives have been considered for each decision made,
- highlight dimensions of cohesion for which there are few or no measures defined.

The framework consists of five criteria, each criterion determining one basic aspect of the resulting measure. First, we describe each criterion: what decisions have to be made, what are the available options, how is the criterion reflected by the cohesion measures in Section 3.1. We then briefly discuss in Section 4.2 how the framework can be used to derive cohesion measures. For each criterion, we have to choose one or more of the available options which will be strongly influenced by the stated measurement goal. Note that these criteria are not sufficient in isolation; other aspects such as properties of measures (e.g., those proposed in [7]) and results from empirical validation studies must be also considered. Due to space constraints, the influence of these aspects cannot be addressed here.

4.1. Framework criteria

4.1.1. Type of connection

By type of connection we mean the mechanism that makes a class cohesive. In Table 1 we summarize types of connections used by the measures in Section 3.1.

A connection within a class is a link between elements of the class (attributes, methods, or data declarations). For each type of connection, the elements are listed in the columns "Element 1" and "Element 2". Column "Description" explains the type of connection. Column "Measures" lists for each type of connection, which of the reviewed measures use that type of connection. The

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

Table 1. Types of connection

#	Element 1	Element 2	Description	Measures
1	method m of a class c	attribute a of class c	m references a	LCOM5
2	method m of class c	method m' of class c	m invokes m'	ICH, LCOM4, C
3	method m of class c	method m' of class c, $m \neq m'$	m and m' directly reference an attribute a of class c in common ("similar methods")	LCOM1, LCOM2, LCOM3, C, LCOM4
4	method m of class c	method m' of class c, $m \neq m'$	m and m' directly or indirectly reference an attribute a of class c in common ("connected methods")	TCC, LCC
5	data declaration in class c	data declaration in class c	data-data interaction	RCI, NRCI, ORCI, PRCI
6	method m of class c	data declaration in class c	data-method interaction	RCI, NRCI, ORCI, PRCI

numbers in column "#" are used later to reference the types of connections.

4.1.2. Domain of the measure

The domain of the measure specifies the objects to be measured (methods, classes etc.). Table 2 shows possible domains for the cohesion measures, and for each domain, the measures from Section 3.1 having that domain.

Table 2. Mapping of measures to domains

Domain	Measures
attribute	-
method	ICH
class	LCOM1, LCOM2, LCOM3, C, LCOM4, LCOM5, TCC, LCC, ICH, RCI, NRCI, PRCI, ORCI
set of classes	ICH
system	-

As we see, most measures are defined at the class level. Measures defined at the attribute and method level are also conceivable. These measures count the number of connections a method or attribute has to other elements of the class. Measures defined on the class level can be scaled up to sets of classes or the whole system.

4.1.3. Direct or indirect connections

We have to decide whether to count direct connections only or also indirect connections. For example, consider a method m_1 which is "similar" to a method m_2 (connection type #3), which in turn is similar to method m_3. Then methods m_1 and m_2 are directly connected through a connection of type #3, as are methods m_2 and m_3. Methods m_1 and m_3 are indirectly connected.

Table 3 shows which measures in Section 3.1 count direct connections only and which also count indirect connections.

Table 3. Measures counting direct and indirect connections

Type	Measures
direct	LCOM1, C, LCOM2, LCOM5, TCC, ICH
indirect	LCOM3, LCOM4, LCC, RCI, NRCI, ORCI, PRCI

4.1.4. Inheritance

Two aspects are to be considered with respect to inheritance:

- How do we assign methods and attributes to classes?
- For method invocation: shall we consider static or polymorphic invocations?

The aspects can be dealt with in the order they are listed here.

How to assign methods and attributes to classes

As we found in the review of the cohesion measures, we have two options available concerning the attributes and methods c has inherited for the analysis of cohesion of a class c:

1. Exclude inherited attributes and methods from the analysis.
   A child class c represents an extension of its parent class d. If we exclude inherited attributes and methods, we analyse to what degree this extension represents a single semantic concept.
2. Include inherited attributes and methods in the analysis.
   If we include inherited attributes and methods, we analyse whether class c as a whole still represents a single semantic concept.

From the measures defined in Section 3.1, ICH conforms to the first option, for TCC and LCC, both

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

options have been suggested. In the definition of the other measures, inheritance apparently has not been addressed.

Polymorphism

The next question is how to deal with polymorphism. This will be relevant only if the chosen type of connection involves method invocations (types #2 and #4), for the special case that a method of a class c polymorphistically invokes a method of its ancestor(s). We have two options:

- Account for polymorphism, i.e., for a method m, we consider connections between m and all methods m' that can possibly be invoked in the implementation of m through polymorphism and dynamic binding.
- Do not account for polymorphism, i.e., for a method m, we count connections between m and methods m' that are statically invoked.

Table 4 shows which measures in Section 3.1 account for polymorphism and which do not. Only measures counting connections of types #2 and #4 are considered in the table.

Table 4. Mapping of measures to options for accounting for polymorphism

Type	Measure
account for polymorphism	ICH
do not account for polymorphism	LCOM4, C, TCC, LCC

4.1.5. Access methods and constructors

As we have seen in the review of the measures, access methods and constructors may artificially increase or decrease the values for cohesion measures. How to account for access methods and constructors should be a conscious decision in the definition of a cohesion measure and is therefore part of the framework.

Access methods

Access methods cause problems for measures which count references to attributes (connection types #1 and #3). Instead of referencing an attribute directly, the access method may be used, which is not accounted for by these types of connections. Thus, the number of references to attributes is artificially decreased. A solution to this problem is to count the invocation of an access method as reference to the attribute. However, this solution may be difficult to implement in practice because it is not always possible to recognize access methods automatically.

Access methods also cause problems for measures that count pairs of methods which use common attributes (connection types #3 and #4). Because access methods usually access only one attribute, many pairs of methods that do not reference a common attribute can be formed using access methods. Thus, the cohesion is artificially decreased. A solution to this problem is to exclude access methods from the analysis.

The available options for how to deal with access methods are summarized in Table 5. Column "Connections" indicates the types of connections for which the respective option is applicable.

Table 5. Options to account for access methods

Option	Description	Connections
1	Do nothing (treat access methods as regular methods)	All types
2	Consider the invocation of an access method as a reference to that attribute	#1, #3
3	Exclude access methods from the analysis	#3, #4

The measures as defined in Section 3.1 all conform to option 1.

Constructors

Constructors cause problems for measures that count pairs of methods which use common attributes (connection types #3 and #4). Constructors typically reference all attributes. This artificially increases the cohesion of the class, because it generates many pairs of methods that use a common attribute. A solution to this problem is to exclude constructors from the analysis. We thus have two options how to account for constructors, which are summarized in Table 6.

Table 6. Options to account for constructors

Option	Description	Connection
4	Do nothing (treat constructors as regular methods)	All types
5	Exclude constructors from the analysis	#3, #4

The measures in Section 3.1 all conform to option 4, except measures TCC and LCC, which take option 5.

4.2. Application of the framework

We apply the framework to select existing measures or to derive new measures for a given measurement goal. Note that the framework is not intended to be used as a means to search cohesion measures in an ad-hoc manner, or to generate an exhaustive set of theoretically possible cohesion measures. Applying the framework implies following two steps:

1. For each criterion of the framework, choose one or more of the available options basing each decision on the objective of measurement.
2. Choose the existing measures accordingly or, if none exist to match the decisions made, construct new cohesion measures. Remember that properties such as those presented in [7] can also be used to guide the definition and theoretical validation of new measures. For instance, Briand *et al.* suggest that a cohesion measure should be

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE

normalized to allow meaningful comparison of the cohesion of classes which have different sizes [7].

As a result of applying the framework criteria, the set of connections of interest for a class will be identified. For the definition of the measure, we then need two figures: the number of actual connections of interest present, and the maximum number of possible connections of interest. We can thus define the cohesion measure as

$$\frac{\text{number of actual connections of interest}}{\text{maximum number of possible connections of interest}}.$$

This yields a normalized cohesion measure ranging between 0 and 1.

5. Conclusions

We have provided a framework for the comparison, evaluation, and definition of cohesion measures in object-oriented systems. This framework is intended to be exhaustive and integrates new ideas with existing measurement frameworks in the literature. Thus, detailed guidance is provided so that cohesion measures may be defined in a consistent and operational way and existing measures may be selected based on explicit criteria. We conclude:

- The measures generated with this framework are proportions of the maximum possible number of connections within classes. This leads to the highest level of measurement, the ratio level, which means the most powerful types of statistical analysis techniques can be performed.
- These measures, however, are not guaranteed to be useful. To be useful, the measures must be indicators of an external quality attribute of interest specified in the measurement goal. We believe that measures of internal product attributes have no inherent significance in isolation. They become useful only if they are related to some external quality attribute [4].
- Existing measures have been classified according to the options available for each criterion of the framework. This classification allows existing measures to be compared and their potential use identified. The classification has shown that some particular options of the framework criteria have no or only few corresponding measures proposed.

We have also used this framework to review the state-of-the-art in object-oriented cohesion measurement (full details are provided in [2]).

References

[1] L. Briand, J. Daly, J. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *Technical Report* ISERN-96-14, 1996.

[2] L. Briand, J. Daly, J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", *Technical Report* ISERN-97-05, 1997.

[3] J.M. Bieman, B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System", in *Proc. ACM Symp. Software Reusability (SSR'94)*, 259-262, 1995.

[4] L. Briand, K. El Emam, S. Morasca, "Theoretical and Empirical Validation of Software Product Measures", *Technical Report* ISERN-95-03, 1995.

[5] L. Briand, S. Morasca, V. Basili, "Measuring and Assessing Maintainability at the End of High-Level Design", *IEEE Conference on Software Maintenance*, Montreal, Canada, September 1993.

[6] L. Briand, S. Morasca, V. Basili, "Defining and Validating High-Level Design Metrics", *Technical Report, University of Maryland, CS-TR 3301*, 1994.

[7] L. Briand, S. Morasca, V. Basili, "Property-Based Software Engineering Measurement", *IEEE Transactions of Software Engineering*, 22 (1), 68-86, 1996.

[8] D.N. Card, V.E. Church, W.W. Agresti, "An Empirical Study of Software Design Practices", *IEEE Transactions on Software Engineering* 12 (2), 264-271, 1986.

[9] D.N. Card, G.T. Page, F.E. McGarry, "Criteria for Software Modularization", *Proceedings IEEE Eighth International Conference on Software Engineering*, 372-377, 1985.

[10] S.R. Chidamber, C.F. Kemerer, "Towards a Metrics Suite for Object Oriented design", in A. Paepcke, (ed.) *Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications* (OOPSLA'91), SIGPLAN Notices, 26 (11), 197-211, 1991.

[11] S.R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 20 (6), 476-493, 1994.

[12] J. Eder, G. Kappel, M. Schrefl, "Coupling and Cohesion in Object-Oriented Systems", *Technical Report*, University of Klagenfurt, 1994.

[13] D.W. Emble, S.N. Woodfield, "Cohesion and Coupling for Abstract Data Types", *6th International Phoenix Conference on Computers and Communications*, Arizona, 1987.

[14] B. Henderson-Sellers, "Software Metrics", *Prentice Hall*, Hemel Hempstaed, U.K., 1996.

[15] M. Hitz, B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented systems", in *Proc. Int. Symposium on Applied Corporate Computing*, Monterrey, Mexico, October 1995.

[16] Y.-S. Lee, B.-S. Liang, S.-F. Wu, F.-J. Wang, "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow", in *Proc. International Conference on Software Quality*, Maribor, Slovenia, 1995.

[17] G. Myers, "Composite/Structured Design", *Van Nostrand Reinhold*, 1978.

Proceedings of the 4th International Software Metrics Symposium (METRICS '97)
0-8186-8093-8/97 $10.00 © 1997 IEEE