The Conceptual Cohesion of Classes

Andrian Marcus, Denys Poshyvanyk
Department of Computer Science
Wayne State University
Detroit Michigan 48202
313 577 5408
amarcus@wayne.edu, denys@cs.wayne.edu

Abstract

While often defined in informal ways, software cohesion reflects important properties of modules in a software system. Cohesion measurement has been used for quality assessment, fault proneness prediction, software modularization, etc. Existing approaches to cohesion measurement in Object-Oriented software are largely based on the structural information of the source code, such as attribute references in methods. These measures reflect particular interpretations of cohesion and try to capture different aspects of cohesion and no single cohesion metric or suite is accepted as standard measurement for cohesion.

The paper proposes a new set of measures for the cohesion of individual classes within an OO software system, based on the analysis of the semantic information embedded in the source code, such as comments and identifiers. A case study on open source software is presented, which compares the new measures with an extensive set of existing metrics. The differences and similarities among the approaches and results are discussed and analyzed.

1. Introduction

Software cohesion can be defined as a measure of the degree to which elements of a module belong together [5]. Cohesion is also regarded from a functional point of view; in this view, a cohesive module is a crisp abstraction of a concept or feature from the problem domain, usually described in the requirements or specifications. Such definitions, while very intuitive, are quite vague and make cohesion measurement a difficult task, leaving too much room for interpretation. In Object-Oriented (OO) software systems, cohesion is usually measured at class level and lately, many different OO cohesion metrics have been proposed (see Section 2 for details), which try to capture different aspects of cohesion, or which reflect a particular interpretation of cohesion. As of today, there is no one cohesion metric that is accepted as a standard. There is little research [7] on assessing the differences and similarities among these metrics.

Software cohesion metrics can be used for different purposes including assessment of design quality [4, 9], prediction of software quality and fault proneness [17, 34], modularization of software [10, 27], identification of reusable of components [19, 25], etc. Most of the existing OO metrics suites are based primarily on the structural aspects of source code (e.g., attribute references in methods). These measures capture the degree to which the elements of a class belong together from a structural point of view, but give no clues whether the class is cohesive from a functional point of view. While other metrics were proposed by researchers (see Section 2 for details) to capture other aspects of cohesion, only few such metrics address the functional aspect of cohesion [18].

We propose a new set of measures for class cohesion based on information retrieval (IR) techniques, which identifies and captures properties shared between members of a class that take into account not only syntactic but also semantic information. Our measure of cohesion can be classified as measuring the information strength of a class within the context of the entire system.

The following section summarizes the related work on other metrics for OO cohesion. Section 3 describes our approach and the proposed measures in detail. Section 4 presents a theoretical evaluation of the proposed measures and an empirical one through a case study on measuring the cohesion of classes in existing open source software. The results are compared with other metrics computed on the same software; differences and similarities are analyzed and discussed. Section 5 concludes the paper summarizing our results and discusses future work in this area of research.

2. Related work

There are several different approaches to measure cohesion in OO systems. Based on the underlying mechanisms used to measure the cohesion of a class one can distinguish: structural metrics [5, 6, 13, 22, 23,

26, 36], the most popular class of cohesion metrics, semantic metrics [18], information entropy-based metrics [1], slice-based metrics [31], metrics based on data mining [32], and metrics for specific types of applications like knowledge-based [24] and distributed systems [14].

The class of structural metrics is the most investigated category of cohesion metrics and includes: LCOM1 (lack of cohesion in methods) [13], LCOM2 [13], LCOM3 [23], LCOM4 [23], C_o (connectivity) [23], LCOM5 [22], Coh [7], TCC (tight class cohesion) [5], LCC (loose class cohesion) [5], ICH (information-flow-based cohesion) [26].

The dominating philosophy behind this category of metrics considers class variable referencing and sharing between methods as contributing to the degree to which the methods of a class belong together. Most structural metrics define and measure relationships among the methods of a class based on this principle. Cohesion is seen dependent on the number of pair of methods that share instance or class variables, one way or another. The differences among the structural metrics are based on the definition of the relationships among methods, system representation, and counting mechanism.

Somewhat different in this class of metrics are LCOM5 and Coh, which consider that cohesion is directly proportional to the number of instance variables in a class that are referenced by the methods in that class. Briand defined a unified framework for cohesion measurement in OO systems [7], that classifies and discusses all these metrics.

Recently, other structural cohesion metrics have been proposed trying to improve existing metrics by considering the effects of the dependent instance variables whose values are computed from other instance variables in the class [12, 36].

While different from each other, all structural metrics capture the same aspects of cohesion, which relate to the data flow between the methods of a class. Other interpretations of cohesion generated different measures and metrics like the Logical Relatedness of Methods (LORM) [18], which is being used to measure the cohesion of a class. Both metrics are based on natural language processing and associate domain level concepts with elements of a source. The philosophy behind this class of metrics, where our proposed measures fall into, is that a cohesive class is a crisp implementation of a problem or solution domain concept. Hence if the methods of a class are conceptually related to each other, the class is cohesive. The difficult problem here is how to define and measure conceptual relationships.

Other cohesion metrics exploit relationships that underline slicing. A large-scale empirical investigation of slice-based metrics [31] indicated that the slice-based cohesion metrics provide complementary views of cohesion to the structural metrics.

A couple of specialized cohesion metrics were proposed for different types of applications. Among those are cohesion metrics for knowledge-based systems [24] and dynamic cohesion metrics for distributed applications [14].

From a measuring methodology point of view, two other cohesion metrics are of interest here since they are also based on an information retrieval approach. Patel et al. [33] proposed a composite cohesion metric that measures the information strength of a module. This measure is based on a vector representation of the frequencies of occurrences of data types in a module. The approach measures the cohesion of individual subprograms of a system based on the relationships to each other in this vector space. Maletic and Marcus [27] defined a file level cohesion metric based on the same type of information we are using for our proposed metrics here. Even though these metrics were not specifically designed for the measurement of cohesion in OO software, they could be extended to measure cohesion in OO systems.

3. An information retrieval approach to class cohesion measurement

OO analysis and design methods try to decompose the problem addressed by the software system development into classes, in an attempt to control complexity. High cohesion for classes and low coupling among classes are design principles aimed at reducing the system complexity. The most desirable type of cohesion for a class is model cohesion [16], such that the class represents a single, semantically meaningful concept. This is the type of cohesion we are trying to measure in our approach.

The designers and the programmers of a software system rarely think about a class as a set of method–attribute interactions. Most often they think about the class as a set of responsibilities that approximate the concept from the problem domain implemented by the class. This type of information is recorded in the source code through identifiers and comments. Analysis of this type of information, referred to as semantic information, is useful for a variety of software development and evolution tasks [2, 3, 11, 27, 29, 30].

Among the existing cohesion metrics for OO software LORM [18] is the only one that uses this type of information to measure the conceptual similarity of the methods in a class, as determined by the representation of the class methods by a semantic network. LORM uses natural language processing techniques for the analysis needed to measure the conceptual similarity of methods.

We are proposing here a different approach that uses the same type of information and is based on a similar interpretation of cohesion. The underlying mechanism used to extract and analyze the semantic information from the source code is based on Latent Semantic Indexing (LSI) [15], an advanced information retrieval method. Any other IR method could be used in this approach like a vector space model or a Bayes classifier, which were used before to support software maintenance tasks [3]. We chose LSI since we already have a positive experience in using it to address other software maintenance tasks such as concept location [30], identification of abstract data types in legacy source code [27], clone detection in software [28], and recovery of traceability links between software and documentation [29]. See [27, 29, 30] for a detailed description of the use of LSI in the context of software based corpus.

The basic usage of LSI in measuring the conceptual cohesion of classes is similar to some extent to our previous work. The source code under analysis is converted into a text corpus, such that from each method only identifiers and comments are extracted. Each method is a document in this corpus and LSI is used to map each document to a vector in a multidimensional space determined by the terms that occur in the vocabulary of the software. This representation is similar to that used in existing search engines such as Google (www.google.com). Once each method is represented as a vector, a similarity measure between any two methods can be defined as the cosine between their corresponding vectors. This similarity measure will express how much relevant semantic information is shared among the two methods, in the context of the entire system.

By computing the degree of similarity between methods of a class we can determine whether a class represents a single semantic abstraction (or concept). This information is then correlated to a new measure of cohesion we call Conceptual Cohesion of Classes (C3).

3.1. System representation

With the IR based underlying mechanism, in order to define and compute the C3 metric, we introduce a graph based system representation, similar to those used to compute other cohesion metrics.

We consider an OO system as a set of classes $C = \{c_1, c_2 \ldots c_n\}$. The total number of classes in the system C is $n = |C|$.

A class has a set of methods. For each class $c \in C$, $M(c) = \{m_1, \ldots, m_k\}$ is the set of methods of class c.

An OO system C is represented as a set of connected graphs $G_C = \{G_1, .., G_n\}$ with G_i representing class c_i. Each class $c_i \in C$ is also represented by a graph $G_i \in G_C$ such that $G_i = (V_i, E_i)$, where $V_i = M(c_i)$ is a set of vertices corresponding to the methods in class c_i and $E_i \subset V_i x V_i$ is a set of weighted edges that connect pairs of methods from the class.

Definition 1. (Conceptual similarity between methods – CSM)

For every class $c_i \in C$, all the edges in E_i are weighted. For each edge $(m_k, m_j) \in E_i$, we define the weight of that edge $CSM(m_k, m_j)$, as the conceptual similarity between the methods m_k and m_j.

The *conceptual similarity between methods* m_k and m_j, $CSM(m_k, m_j)$ is computed as the cosine between the vectors corresponding to m_k and m_j in the semantic space constructed by the IR method (in this case LSI).

$$CSM(m_k, m_j) = \frac{vm_k^T vm_j}{|vm_k|_2 \times |vm_j|_2},$$

where vm_k and vm_j are the vectors corresponding to the $m_k, m_j \in M(c_i)$ methods.

For each class $c \in C$ we have a maximum of $N = C_n^2$ distinct edges between different nodes, where $n = |M(c)|$.

3.2. The conceptual cohesion of classes (C3)

With this system representation we define a set of measures that approximate the cohesion of a class in an OO software system by measuring the degree to which the methods in a class are related conceptually.

Defintion 2. (Average conceptual similarity of methods in a class – ACSM)

The average conceptual similarity of the methods in a class $c \in C$ is:

$$ACSM(c) = \frac{1}{N} \times \sum_{i=1}^{N} CSM(m_i, m_j),$$

where $(m_i, m_j) \in E$, $i \neq j$, $m_i, m_j \in M(c)$, and N is the number of distinct edges in G, defined in def. 1.

In our view, ACSM(c) defines degree to which methods of a class belong together conceptually and thus it can be used as basis for computing the conceptual cohesion of classes.

Definition 3. (Conceptual cohesion of a class – C3)

For a class $c \in C$, the conceptual cohesion of c, C3(c) is defined as following:

$$C3(c) = \begin{cases} ACSM(c) & if \quad ACSM(c) > 0 \\ else \quad 0 \end{cases}$$

Based on the above definitions, $C3(c) \in [0, 1]$ $\forall c \in C$. If a class $c \in C$ is cohesive then C3(c) should be closer to one meaning that all methods in the class are strongly related conceptually with each other (i.e., the CSM for each pair of methods is close to one). In this case, the class most likely implements a single

concept or a very small group of related concepts (related in the context of the software system).

If the methods inside the class have low conceptual similarity values between them (CSM close to or less than zero), then the methods most likely participate in the implementation of different concepts and C3(c) will be close to zero.

3.3. An example of measuring C3

To better understand the C3 metric, consider a class $c \in C$ with five methods m1, m2, m3, m4, m5. The conceptual similarities between the methods in the class are shown in Table 1. For the computation of ACSM we consider all pairs of different methods, thus ACSM(c) = 0.5. Since the value is positive, C3(c) = ACSM(c) = 0.5. This particular value for C3 does not indicate high cohesion for class c nor a low one, but the CSM values from Table 1 show that m1 and m3, m2 and m4, m2 and m5, and m4 and m5 are closely related respectively (i.e., the CSM between each pair is larger than C3). As one can see in this example, CSM is not a transitive measure. Since C3 is an average measure, we could have situations when some pairs of methods are highly related and other are not and the average is around 0.5.

With that in mind, we refine the C3 to measure the influence of the difference between the highly related and unrelated pairs of methods on the cohesion of the class.

Table 1. Conceptual similarities between the methods in class c. ACSM(c) = 0.5.

	m1	m2	m3	m4	m5
m1	1	0.21	0.72	0.33	0.42
m2		1	0.28	0.91	0.66
m3			1	0.37	0.27
m4				1	0.89
m5					1

3.4. Lack of conceptual similarity between methods (LCSM)

In order to capture the influence of highly related methods in a class with a low C3 cohesion we define a new measure based on the counting mechanism utilized in LCOM2 [13]. In our case, of course, the metrics does not take into account intersections of methods based on common attribute usage, but it counts intersections of method pairs based on the CSM value between them.

Definition 4. (Lack of conceptual similarity between methods – LCSM)

Consider a class $c \in C$ represented by graph G = (V, E) as defined in section 3.1. $V = M(c) = \{m_1, m_2, ..., m_n\}$ is the set of nodes in the graph with n = |M(c)| methods. Only edges between pairs of methods with CSM higher than the average are considered: $(m_i, m_j) \in E \Leftrightarrow CSM(m_i, m_j) > ACSM(c)$, $m_i \neq m_j$.

Let $M_i = \{m_j \mid (m_i, m_j) \in E, m_i \neq m_j\}$ be the set of neighbor methods of m_i (with which m_i has a higher CSM value than the average).

Let $P = \{(M_i, M_j) \mid M_i \cap M_j = \emptyset\}$. If all n sets $M_1, \ldots, M_n$ are Ø, then let P = Ø.

Let $Q = \{(M_i, M_j) \mid M_i \cap M_j \neq \emptyset\}$.

With these measures, we define the lack of conceptual similarity between methods for a class c as:

$$LCSM(c) = \begin{cases} |P| - |Q| & if \quad |P| > |Q| \\ else \quad 0 \end{cases}$$

Just as LCOM2, LCSM is an inverse measure of cohesion, which means that a higher value for LCSM indicates lower cohesion. Note that the LCSM metric for a class where |P| = |Q| will be zero. This does not necessarily imply maximal cohesion, since within the set of classes with LCSM = 0, some may be more cohesive than others, as indicated by C3. LCSM is intended to complement C3, therefore for a more complete assessment of the cohesion of a class, both should be computed.

3.5. An example of measuring LCSM

Consider the same class c described in section 3.3 with C3(c) = 0.5. For each method of the class c, we compute M_i based on definition 4: M1 = {m3}, M2 = {m4, m5}, M3 = {m1}, M4 = {m2, m5}, M5 = {m2, m4}. Table 1 shows us the intersection among all pairs of sets $M_i \cap M_j$ in class c. Based on the intersection P = {(M1, M2); (M1, M3); (M1, M4); (M1, M5); (M2, M3); (M3, M4); (M3, M5)} and |P| = 7. Q = {(M2, M4); (M2, M5); (M4, M5)} and |Q| = 3. Thus, LCSM(c) = 7-3 = 4.

Table 2. Intersection results for method sets

	M1	M2	M3	M4	M5
M1		**Ø**	**Ø**	**Ø**	**Ø**
M2			**Ø**	**m5**	**m4**
M3				**Ø**	**Ø**
M4					**m4**
M5					

The two results combined indicate a lower value for the cohesion of class c from the example. In another situation, class c' could have had more highly related methods than in this case (i.e., four pairs) and less unrelated method pairs with the same C3(c') value (i.e., 0.5). Assume Table 2 would indicate 6 pairs of method sets with non empty intersection and only 4 with an empty intersection. The LCSM(c') in that case would

be 0. The combined measures will indicate that c' is more cohesive than c.

3.6. Measuring methodology and tool support

The measuring methodology for the proposed cohesion metrics is described in Figure 1. The following steps are necessary to compute the C3 and LCSM metrics:

- Preprocessing and parsing of the source code to produce a text corpus. Comments and identifiers from each method are extracted and processed. A document in the corpus is created for each method in every class.
- An IR method is used to index the corpus and create an equivalent semantic space.
- Based on the IR indexing conceptual similarities are computed between each pair of methods.
- Based on the conceptual similarity measures, C3 and LCSM are computed for each class.

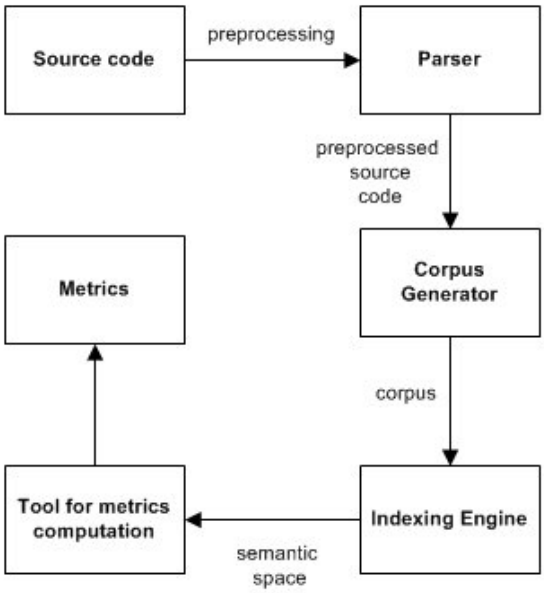

Figure 1. Measuring methodology and tools

We implemented a tool to compute C3 and LCSM for C++ software projects in MS Visual Studio .NET, based on the above methodology. Our source code parser component is based on the "Visual C++ Object Extensibility Model". Using project information retrieved from Visual Studio .NET, the tool retrieves parts of source code that are used to produce a corpus. The extracted comments and identifier are processed in a similar fashion we used in [30], by elimination of stop words and splitting identifiers that follow predefined coding standards. The corpus is indexed by the indexing engine, which is an implementation of LSI. We use the cosine between vectors in the LSI space to compute conceptual relations.

3.7. Limitations of the proposed metrics

Both C3 and LCSM metrics greatly depend upon reasonable naming conventions for identifiers and relevant comments contained in the source code. When these are missing, the only hope for measuring any aspects of cohesion rests on the structural metrics. Section 4.1 presents a case study in which we computed C3 and LCSM for the same software once considering the comments and once only considering identifiers. We are currently working on a set heuristics that would indicate the circumstances when the comments and identifiers will not help in measuring cohesion.

Another limitation of C3 and LCSM is that they do not take into account polymorphism and inheritance. They only consider methods of a class that are implemented or overloaded in the class. Method invocation, parameters, attribute references, and types are of interest only at identifier level. Each occurrence of an identifier as an invocation, attribute reference, or type specification within the body of a method contributes to the vector representation of the method in the semantic space. The vector will get 'closer' to the ones where the same identifiers are used.

Finally, as most of the other cohesion measures, C3 and LCSM do not make distinction between constructors, accessors, and other method stereotypes. Some of these methods can artificially increase or decrease cohesion.

4. Assessment of C3 and LCSM

For newly proposed metrics, empirical and theoretical evaluations are needed [7, 8].

The mechanism used to measure C3 and LCSM is general enough to accommodate measurement of various components and elements of a software system regardless of the programming paradigm or language used to build the system.

We consider the following properties of cohesion metrics as important [8]: non-negativity (i.e., cohesion should be non-negative) and normalization (i.e., the measure is independent of the size of the class), null and maximal value (i.e., a class can have no cohesion or can have maximum cohesion), and monotonicity.

Both C3 and LCSM comply with the non-negativity property. However, only C3 is normalized (i.e., for a class c, $C3(c) \in [0, 1]$). LCSM is not normalized, because its value depends on the number of methods in the class, since the counting mechanism is similar to that used by LCOM2. Normalization provides the mechanism for meaningful comparisons between the cohesion of different classes.

If none of the methods of the class share conceptually related content (i.e., identifiers and comments), they are considered to have different semantic context and C3(c) may be close to zero. If methods share the same vocabulary, then the conceptual similarity between methods is high and C3(c) should be close to one (i.e., if each method is a clone of the other, C3 will be one). On the other hand, if there are no empty method sets intersections for LCSM or the number of non-empty intersections is greater than number of empty intersections, then LCSM(c) = 0. The maximum value of LCSM in this case is C_n^2, where n is the number of methods in class c. The maximum value is possible in the case when all methods in the class are conceptually unrelated indicating complete absence of cohesion.

While we will not prove monotonicity for C3 and LCSM, intuitively, if the value of the conceptual relationships between two methods is increased, then C3 will obviously increase since it is based on the average of all such values. The similar situation will result in the increase (or not) in the number of neighbors of a method in the class. This in turn will result in a lower (or equal) value of LCSM, which means higher (or equal) cohesion.

4.1. Case study

In order to evaluate the proposed metrics against existing structural metrics we performed a case study on open source software.

4.1.1. Case study design. The goal of the case study is to determine how the proposed metrics correlate to the traditional structural ones. We also tried to see whether the presence or absence of the comments in the source code significantly affects C3 and LCSM metrics.

We selected the following structural cohesion metrics to compare with C3 and LCSM: LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, C, ICH, TCC and LCC. The goal was to establish which structural metrics correlate well with the conceptual cohesion metrics. The assumption is that if two metrics correlate well, then they measure the same aspects of cohesion. Our choice of metrics is not random, since these structural metrics were extensively studied [4, 7, 9, 20, 35] and compared to each other and to other metrics.

For the case study we computed the structural and the conceptual cohesion metrics for the open source project WinMerge 2.0.2 (sourceforge.net/projects/winmerge/), which is commonly used for visual differencing and merging of files and directories. The software consists of 173 *.cpp and *. h files with 51,457 lines of code and 11,533 lines of comments.

The structural metrics were computed using the Columbus tool [21] and the conceptual metrics were computed with our prototype tool.

Columbus computed metrics for 69 classes containing 624 methods. We excluded abstract classes with pure virtual functions from the analysis since they have no implementation. Thus, we considered the structural metrics for only 34 classes with 522 methods. Our tool computed C3 and LCSM metrics for the 34 classes with 522 methods as well. In order to study the effect of comments in the source code on the C3 and LCSM metrics, we also computed these metrics for WinMerge without using the comments in the source code; these two metrics are referred to as C3' and LCSM' respectively.

Table 3. Statistics for cohesion measures for WinMerge application. C3' and LCSM' are the values for C3 and LCSM computed for WinMerge without using the comments in the code.

Metric	Max	75%	Med	25%	Min	Mean	Std. Dev
C3	**0.98**	**0.69**	**0.52**	**0.39**	**0.19**	**0.55**	**0.22**
C3'	**0.84**	**0.63**	**0.58**	**0.31**	**0.18**	**0.52**	**0.21**
LCSM	**7**	**1.5**	**0**	**0**	**0**	**0.97**	**1.56**
LCSM'	**9**	**2**	**1**	**0**	**0**	**1.7**	**2.76**
LCOM1	1867	63.5	23	6.5	0	174.51	413.74
LCOM2	1843	61	19	4	0	163.34	398.05
LCOM3	54	10	5	3	0	9.2	11
LCOM4	33	10	5	3	0	7.34	7.53
LCOM5	1.13	0.9	0.73	0	0	0.52	0.44
ICH	102	4	0	0	0	9.71	24.54
TCC	1	0.17	0	0	0	0.17	0.31
LCC	1	0.24	0	0	0	0.2	0.34
Coh	0.67	0.27	0.13	0	0	0.16	0.18

4.1.2. Results. For each metric we also computed a set of descriptive statistical values, at system level: the maximum values, interquartile ranges, median, minimum, mean value, and standard deviation. This data is used to give an overall picture of the differences and similarities among all these metrics, at system level. The statistics for structural and conceptual cohesion measures are provided in Table 3.

For more precise information, we computed Pearson's correlation coefficient between every pair of metrics, for each class. The correlation coefficient measures the strength of the linear relationship between two variables, which in our case are two metrics.

The value of the coefficient is between -1 and +1, inclusively. A value of -1 would indicate a perfect negative correlation between the two variables. Correspondingly, a value of +1 would indicate a perfect positive correlation between the two variables. If the correlation coefficient is 0, then there is no linear relationship between the two variables. In this case study we consider correlation of 0.1 to be trivial, 0.1 – 0.3 minor, 0.3 – 0.5 moderate, 0.5 – 0.7 large, 0.7 – 0.9

Table 4. Correlations of cohesion metrics for WinMerge.

	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	ICH	TCC	LCC	Coh	C3	LCSM	C3'	LCSM'
LCOM1	1.00	**1.00**	**0.84**	**0.49**	0.21	**0.86**	-0.11	-0.03	-0.26	-0.35	-0.23	-0.29	-0.21
LCOM2		1.00	**0.86**	**0.51**	0.19	**0.84**	-0.11	-0.05	-0.27	-0.33	-0.23	-0.26	-0.21
LCOM3			1.00	**0.83**	-0.06	**0.61**	-0.28	-0.26	-0.42	-0.08	-0.19	-0.01	-0.12
LCOM4				1.00	-0.34	0.20	-0.37	-0.35	**-0.51**	0.14	-0.14	0.21	-0.05
LCOM5					1.00	0.37	0.22	0.25	**0.54**	**-0.65**	-0.15	**-0.52**	-0.22
ICH						1.00	-0.02	0.10	-0.14	**-0.49**	-0.24	**-0.51**	-0.23
TCC							1.00	**0.97**	**0.63**	-0.16	-0.14	-0.17	-0.09
LCC								1.00	**0.55**	-0.24	-0.19	-0.31	-0.13
Coh									1.00	-0.23	0.14	-0.31	-0.09
C3										1.00	0.23	**0.79**	0.31
LCSM											1.00	0.17	**0.70**

very large, and 0.9 – 1 almost perfect. The correlation coefficients between the cohesion metrics measured on WinMerge, with and without comments, are presented in Table 4.

4.1.3. Discussion. The statistics in Table 3 show us that the values for C3 and C3' are very close to each other. This is a clear indication that in the case of WinMerge the comments in the source code, which amount to approximately 20% of the text, do not influence much the computation of C3. The values for LCSM and LCSM' are less conclusive in this respect, but the differences are still not major. The standard deviation across all the metrics is quite similar (20-30% of the maximum value).

The more interesting results though are provided by the analysis of the correlation data from Table 4.

LCOM1, LCOM2, LCOM3, and LCOM4 are strongly correlated and thus they are measuring similar properties of cohesion. This is not surprising since all these metrics are based on counting instance variable usage in the methods of a class. It is interesting however that LCOM5 does not correlate with LCOM1-LCOM4, although it is just a variation on counting how methods access the attributes of a class. The Coh metric, which is an extension of LCOM5, moderately correlates with LCOM4 and LCOM5, which is also surprising. We expected a stronger correlation among these metrics. An interesting correlation is found among ICH and LCOM1-LCOM3, although it is somewhat counterintuitive considering their definitions. ICH is information flow-based cohesion measure based on the information strength (i.e., method invocations weighted by the number of parameters invoked) among the methods of a class. Even though the approaches between ICH and LCOM1-3 are different, they seem to capture the same aspects of cohesion. In the end, these results are in line with and support previous empirical studies that compared these structural metrics [20].

To reach the goals of the case study, we focused on the analysis of the correlations with C3 and LCSM. The analysis reveals significant correlations between C3 and ICH, and C3 and LCOM5. The first pair is not very surprising because ICH uses the number of invocations of other methods weighted by the number of parameters in its formula. The method invocations will increase the frequency of shared terms between methods, thus implicitly the value of C3. The correlation coefficient between C3 and ICH is negative since ICH is an inverse measure for cohesion.

LCOM5 is based on counting the number of methods referencing attributes in the class. The idea is akin to the way LSI counts term frequencies in documents, and such attribute references are terms in the methods.

On the other hand, we did not find significant correlation between any structural metric and LCSM. In fact LCSM does not correlate with C3 either, although they use the same information extracted from the source code, but the counting mechanism is different. It seems that LCSM captures aspects on cohesion that are not addressed by any metrics in the study. While we expected this to be true with respect with C3, it is somewhat of a surprise, since we expect more significant correlation with LCOM2, considering the common counting mechanism. We will conduct more case studies in the future to further confirm these results on other software systems.

Since the quality of the comments in the source is a cause for concern when measuring C3 and LCSM, we compared C3 with C3' and LCSM with LCSM'. We found significant correlation between C3 and C3' as well as LCSM and LCSM'. For C3' we observed significant correlation with ICH and LCOM5 as well, similar with C3.

To gain more insight into how our metrics differ from some of the structural ones, we analyzed the classes with high structural and low conceptual cohesion and vice versa. Henderson-Sellers [22] noted that: "It is after all possible to have a class with high internal, syntactic cohesion but little semantic cohesion". We selected several classes based on high LCOM2 values (i.e., indicating low cohesion) and high C3 values and vice versa. The classes selected for further analysis based on values for C3 and LCOM2 are shown in Table 5. Overall, a class is considered to have low LCOM2 values if it is in the first 15% of classes with lowest values for the metric. We apply the same threshold for identifying the classes with high values for C3.

Table 5. Classes under analysis

Class name	C3	LCOM2
IVSSItem	0.64	528
IVSSDatabase	0.635	136
IVSSItemOld	0.632	465
BCMenuData	0.434	0
CDirDoc	0.294	0
RescanSuppress	0.392	1

The analysis of the classes in Table 5 yields very interesting results. For example, the IVSSItem class is a wrapper class that does not have data members, only methods that wrap the implementation for the OLE automation on the client side. High values for LCOM2 in methods are easily explained in this case. The intersection of every pair of methods in this class is null, because the class does not contain any attribute. For the IVSSItem class that has 33 methods, LCOM2 = 528. The high C3 value is also understandable, because the implementation of every method contains invocations of the InvokeHelper method of the derived class COleDispatchDriver and a similar subset of identifier names for local variables. Wrappers tend to group together methods that are conceptually similar. The IVSSDatabase and IVSSItemOld classes follow the same pattern since they also implement wrappers for the COleDispatchDriver interface. In conclusion, in these situations (i.e., wrappers) it seems that C3 is a more suitable measure for cohesion than LCOM2.

From the other group of investigated classes, BCMenuData is a class that implements a "property container" for menu items that are drawn using an "Office XP" like style. It is a small class with a set of accessor functions. LCOM2 is 0, meaning that the number of intersecting sets is more than the number of non-intersecting ones. Close examination of the class supports the fact that the class represents a single meaningful abstraction however values of C3 do not capture this fact due to the large number of unique identifier names used in these accessors. As mentioned above, accessor methods, just like constructors may influence significantly the measurement of C3.

The CDirDoc class is an example of a class with concealed cohesion, which means that the class includes some attributes and methods that might create another class. Close analysis revealed that the class handles the following activities: creating and closing of a new document, representing "right-left" panel abstraction in the "view-merge" application, keeping track of updating time, status and content, as well as choosing different view modes. It has only several attributes like pointer to CDirView class and a container of CMergeDoc classes. Those attributes are referenced in most methods of the class. Thus LCOM2 for the CDirDoc is 0. On the other hand, the value of C3 for CDirDoc is 0.294 which shows low conceptual similarity of methods inside the class. Detailed analysis shows that the class implements a set of concepts that could be refactored into separate classes implementing each concept at a time. The low LCOM2 value would indicate a difficult refactoring since it may create high coupling. However, when considering the low number of attributes of the class, this is not a major issue. While it is hard to generalize, in situation when a class has few attributes and many methods by comparison, low LCOM2 value and high low C3 value may indicate lack of cohesion after all.

The RescanSuppress class implements an abstraction of a simple lock that prevents objects of type CMergeDocs from rescanning within its lifetime unless the clear() method is called. It is a small class with three attributes and three methods - constructor, destructor and the clear() method. Although the class represents a crisp abstraction from a user point of view, the small value for C3 can be explained due to the small number of identifiers and their intersections within method implementations of the class. This is a situation where C3 showed its limits.

Finally, we also analyzed classes with the maximum and minimum C3 values. The classes with the highest C3 values (i.e., 0.9812 and 0.9187 respectively) are IVSS and BCMenuMemDC. The LCOM2 values for those two classes are 6 and 5 respectively. The IVSS class is also a wrapper class as described earlier. The BCMenuMemDC class turned out to be a crisp abstraction from both perspectives: structural and conceptual. While no strong correlation was found between LCOM2 and C3, the same observation is true for other classes with high C3 and structural cohesion (based on LCOM2) that we analyzed: WaitStatusCursor, CPropSyntax, CColorButton, and CDiffContext.

The analysis of classes with the lowest C3 values revealed CMainFrame and CMergeDoc that have C3 values of 0.1863 and 0.1975 respectively. These classes also have very low LCOM2 values, 1470 and 814 respectively. Analysis of the source code implementing the classes showed that CMainFrame is one of the largest classes in the system having 1397 LOC and 804 lines of comments in 76 methods. It is clear from the source code that this class implements multiple unrelated abstractions and could be restructured into several more cohesive classes. CMergeDoc follows the same pattern as CMainFrame. It has 1212 LOC and 402 lines of comments in 58 methods implementing several concepts.

Overall, it seems that LCOM2 and C3 values for the classes significantly correlate among the highest and lowest values respectively.

4.1.4. Threats to validity. Several issues affected the results of our case study and limit our interpretation.

We have demonstrated that some of the structural and conceptual measures investigated have significant correlation between them. Such statistical relationships do not necessarily demonstrate a casual relationship, but rather provide empirical evidence of it. Only controlled experiments, where the measures would be varied in a controlled manner and all other factors would be held constant, could demonstrate causality.

The analysis of the results at this stage in our research showed how the conceptual cohesion metrics can complement structural metrics in some cases. However, only the values of C3 and LCOM2 were used to draw these conclusions. This may affect the results if another structural metric is used in conjunction with C3. Further analysis should be done to find the combinations of metrics that can capture different aspects of cohesion and that will best complement each other in a variety of cases.

The size and the quality of the WinMerge source also have an impact on the results. While the structural metrics are not much affected, the construction of the corpus and the indexing are dependent on these factors.

5. Conclusions and future work

The paper presented a new set of metrics for measuring the conceptual cohesion of classes. The metrics are measured using semantic information embedded in the source code (i.e., comments and identifiers) via an IR approach. A case study on open source software provided evidence of unsurprising correlations between C3 and ICH and LCOM5 respectively, meaning that those metrics may capture similar aspects of cohesion. On the other hand we did not find any significant correlation between LCSM and any structural metric. However, LCSM seem to complement well C3 in some situations. Further studies are needed in order to determine more subtle relationships between LCSM and other cohesion metrics. We found that C3 and LCOM2 can help identify special cases like wrappers or classes that have several concepts implemented that can be refactored into a set of classes. According to the results of the case study, C3 and LCSM can also be applied to the software with sparse or missing comments.

More case studies are planned on software written in other languages. We need to further investigate the effect of certain method stereotypes on these cohesion metrics and of the class interfaces, which were not captured here. In addition, we plan to extend the suite of conceptual cohesion metrics with new ones based on different counting mechanism (i.e., like LCOM3). Comparison with metrics in other categories (e.g., semantic metrics) is also desirable. In the long run we want to identify groups of metrics, which used in conjunction best capture the cohesion of classes.

6. Acknowledgements

This research was supported in part by a grant from the National Science Foundation (CCF-0438970).

7. References

[1] Allen, E. B., Khoshgoftaar, T. M., and Chen, Y., "Measuring coupling and cohesion of software modules: an information-theory approach", in Proc. of 7th International Software Metrics Symposium, April 4-6 2001, pp. 124-134.

[2] Anquetil, N. and Lethbridge, T., "Assessing the Relevance of Identifier Names in a Legacy Software System", in Proceedings of Annual IBM Centers for Advanced Studies Conference (CASCON'98), December 1998, pp. 213-222.

[3] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E., "Recovering traceability links between code and documentation", *IEEE Transactions on Software Engineering*, vol. 28, no. 10, October 2002, pp. 970-983.

[4] Bansiya, J. and Davis, C. G., "A hierarchical model for object-oriented design quality assessment", *IEEE Transactions on Software Engineering*, vol. 28, no. 1, January 2002, pp. 4-17.

[5] Bieman, J. and Kang, B.-K., "Cohesion and reuse in an object-oriented system", in Proceedings of ACM Symposium on Software Reusability (SSR'95), April 1995, pp. 259-262.

[6] Briand, L. C., Daly, J. W., Porter, V., and Wüst, J., "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", in Proc. of International Software Metrics Symposium, Bethesda, MD, Nov. 21 1998, pp. 43-53.

[7] Briand, L. C., Daly, J. W., and Wüst, J., "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", *Empirical Software Engineering*, vol. 3, no. 1, 1998, pp. 65-117.

[8] Briand, L. C., Morasca, S., and Basili, V. R., "Property-Based Software Engineering Measurements", *IEEE Transactions on Software Engineering*, vol. 22, no. 1, January 1996, pp. 68-85.

[9] Briand, L. C., Wüst, J., Daly, J. W., and Porter, V. D., "Exploring the relationship between design measures and software quality in object-oriented systems", *Journal of System and Software*, vol. 51, no. 3, May 2000, pp. 245-273.

[10] Brito e Abreu, F. and Goulao, M., "Coupling and cohesion as modularization drivers: are we being over-persuaded?" in Proceedings of 5th European Conference on Software Maintenance and Reengineering, 2001, pp. 47-57.

[11] Caprile, B. and Tonella, P., "Restructuring program identifier names", in Proceedings of International Conference on Software Maintenance, Oct. 11-14 2000, pp. 97-107.

[12] Chae, H. S., Kwon, Y. R., and Bae, D. H., "Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables", *IEEE Transactions on Software Engineering*, vol. 30, no. 11, November 2004, pp. 826-832.

[13] Chidamber, S. R. and Kemerer, C. F., "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, 1994, pp. 476-493.

[14] Cho, E. S., Kim, C. J., Kim, D. D., and Rhew, S. Y., "Static and dynamic metrics for effective object clustering", in Proceedings of Asia Pacific International Conference on Software Engineering, 1998, pp. 78 - 85.

[15] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R., "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, vol. 41, 1990, pp. 391-407.

[16] Eder, J., Kappel, G., and Schreft, M., "Coupling and Cohesion in Object-Oriented systems", University of Klagenfurt, Technical Report 1994.

[17] El-Emam, K., "Object-Oriented Metrics: A Review of Theory and Practice", in *Advances in software engineering*, Springer-Verlag, New York, 2002, pp. 23-50.

[18] Etzkorn, L. and Delugach, H., "Towards a semantic metrics suite for object-oriented design", in Proceedings of 34th International Conference on Technology of Object-Oriented Languages and Systems, July 30 2000, pp. 71 - 80.

[19] Etzkorn, L. H. and Davis, C. G., "Automatically Identifying Reusable OO Legacy Code", *IEEE Computer*, vol. 30, no. 10, October 1997, pp. 66-72.

[20] Etzkorn, L. H., Gholston, S. E., Fortune, J. L., Stein, C. E., Utley, D., Farrington, P. A., and Cox, G. W., "A comparison of cohesion metrics for object-oriented systems", *Information and Software Technology*, vol. 46, no. 10, August 2004, pp. 677-687.

[21] Ferenc, R., Siket, I., and Gyimóthy, T., "Extracting facts from open source software", in Proceedings of 20th International Conference on Software Maintenance (ICSM'04), September 11-14 2004, pp. 60-69.

[22] Henderson-Sellers, B., *Software Metrics*, U. K., Prentice Hall, 1996.

[23] Hitz, M. and Montazeri, B., "Measuring Coupling and Cohesion in Object-Oriented Systems", in Proceedings of International Symposium on Applied Corporate Computing, Monterrey, Mexico, October 1995.

[24] Kramer, S. and Kaindl, H., "Coupling and cohesion metrics for knowledge-based systems using frames and rules", *ACM Transactions on Software Engineering and Methodology*, vol. 13, no. 3, July 2004, pp. 332-358.

[25] Lee, J. K., Jung, S. J., Kim, S. D., Jang, W. H., and Ham, D. H., "Component identification method with coupling and cohesion", in Proceedings of Eighth Asia-Pacific Software Engineering Conference, December 2001, pp. 79-86.

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

[27] Maletic, J. I. and Marcus, A., "Supporting Program Comprehension Using Semantic and Structural Information", in Proceedings of 23rd International Conference on Software Engineering, Toronto, Canada, May 12-19 2001, pp. 103-112.

[28] Marcus, A. and Maletic, J. I., "Identification of High-Level Concept Clones in Source Code", in Proceedings of Automated Software Engineering (ASE'01), San Diego, CA, November 26-29 2001, pp. 107-114.

[29] Marcus, A. and Maletic, J. I., "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", in Proceedings of 25th IEEE/ACM International Conference on Software Engineering, Portland, OR, May 3-10 2003, pp. 125-137.

[30] Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J. I., "An information retrieval approach to concept location in source code", in Proceedings of 11th Working Conference on Reverse Engineering, November 8-12 2004, pp. 8-12.

[31] Meyers, T. M. and Binkley, D., "Slice-based cohesion metrics and software intervention", in Proceedings of 11th Working Conference on Reverse Engineering (WCRE'04), Nov. 8-12 2004, pp. 256-265.

[32] Montes de Oca, C. and Carver, D. L., "Identification of data cohesive subsystems using data mining techniques", in Proceedings of International Conference on Software Maintenance (ICSM'98), November 1998, pp. p. 16-23.

[33] Patel, S., Chu, W., and Baxter, R., "A Measure For Composite Module Cohesion", in Proceedings of International Conference on Software Engineering (ICSE'92), May 11-15 1992, pp. 38-48.

[34] Quah, T.-S. and Thwin, M. M. T., "Application of neural networks for software quality prediction using object-oriented metrics", in Proceedings of International Conference on Software Maintenance, September 22-26 2003, pp. 116-125.

[35] Succi, G., Pedrycz, W., Djokic, S., Zuliani, P., and Russo, B., "An Empirical Exploration of the Distributions of the Chidamber and Kemerer Object-Oriented Metrics Suite", *Empirical Software Engineering*, vol. 10, no. 1, January 2005, pp. 81-104.

[36] Zhou, Y., Xu, B., Zhao, J., and Yang, H., "ICBMC: an improved cohesion measure for classes", in Proceedings of International Conference on Software Maintenance, October 3-6 2002, pp. 44-53.