High Precision Cohesion Metric

N. KAYARVIZHY[1], S. KANMANI[2], R.V. UTHARIARAJ[3]
[1]Assistant Professor, Department of Computer Science and Engineering
AMC Engineering College
12th K.M., Bannerghatta Road, Bangalore – 560083
[2]Professor, Department of Information Technology
Pondicherry Engineering College
Puducherry – 605014
[3]Professor and Director, Ramanujam Computing Centre,
Anna University, Chennai – 25, Tamil Nadu
INDIA
[1]kayarvizhy@gmail.com, [2]kanmani@pec.edu, [3]rhymend@annauniv.edu

Abstract: - Metrics have been used to measure many attributes of software. For object oriented software, cohesion indicates the level of binding of the class elements. A class with high cohesion is one of the desirable properties of a good object oriented design. A highly cohesive class is less prone to faults and is easy to develop and maintain. Several object oriented cohesion metrics have been proposed in the literature. These metrics have provided a lot of valuable insight into the mechanism of cohesion and how best to capture it. However they do suffer from certain criticisms. In this paper, we propose a new cohesion metric named as High Precision Cohesion Metric (HPCM). HPCM addresses the drawbacks present in the existing cohesion metrics. We introduce two novel concepts - link strength and average attributes used in a class and apply them to arrive at the proposed metric. The metric is presented using the unified cohesion framework to avoid ambiguity. The metric is validated using theoretical approach suggested in the unified framework for cohesion metrics. Empirical validation is also carried out on this metric using data from open source Java projects. The newly proposed High Precision Cohesion Metric overcomes the shortfalls of the earlier proposed cohesion metrics.

Key-Words: - Object Oriented Metrics, Cohesion, High Precision, Link Strength

1 Introduction

Object oriented design and development continues to be the most widely used methodology for designing high quality software. Ensuring the quality of such systems is of prime interest. This is done through various approaches at different phases in the life cycle in software development. Eliminating the defects that affect quality in early stages of software development life cycle, like design, saves cost and effort. Object oriented design-level metrics and their associated quality prediction techniques attempt to ensure quality during the design and early coding phase. This has been a prime area of research.

Cohesion metrics indicate how well the class elements bind with each other. Since a class consists of two basic sets of elements, attributes and methods, all of the cohesion metrics revolve around the usage of these. A high cohesion value indicates that the class is well structured and provides the stated functionality with the help of well knit attributes and methods [1]. Conversely a low cohesive value indicates a class that may need to be split or redesigned. A non-cohesive class is a maintenance nightmare and is prone to faults [2] [3]. Mens [4] has stated that the cohesion metrics are too coarse and need to be complemented with finer grained factors. After analyzing existing cohesion metrics, we found that the metrics do not capture precise information which can be used to discriminate classes based on their cohesion values. Preventive actions on classes which are tagged as non cohesive are costly and hence there is a need to group classes based on a highly precise cohesion value. We propose a new cohesion metric that has high precision and hence capable of distinguishing classes with similar but not identical cohesiveness. This property sets it apart from the earlier proposed metrics and manages to provide a wider range of values to fit the classes.

This paper is organized as follows. Section 2 describes in short the significant cohesion metrics that have been proposed in the literature. This is followed by Section 3 that describes the drawbacks of the existing cohesion metrics and hence the motivation behind the new cohesion metric. Section 4 introduces the proposed metric and presents it in the unified framework proposed by Briand et al. developed for cohesion metrics. Section 5 highlights the applicability criteria of the new metric. Section 6 deals with mathematical evaluation of the proposed metric. This is followed by an empirical validation in Section 7 with open source object oriented programs in Java. Section 8 summarizes the conclusions derived from this study

2 Related Work

Several cohesion measures have been proposed in the literature. Chidamber and Kemerer in 1991 [5] proposed the first cohesion metric. They presented an inverse metric, thus measuring the lack of cohesion (LCOM1). This metric simply measures the number of pairs of methods that do not have any common attribute.

Chidamber and Kemerer revised their metric in 1994 and came up with an altered version LCOM2 [6], which found the difference between the number of pairs of methods not sharing any attribute and those sharing at least one attribute. Li and Henry in 1995 [7] proposed their version of lack of cohesion, LCOM3 as the number of connected components in a graph with methods as node and edges between each pairs of methods with at least one common attribute. Hitz and Montazeri in 1996 [8] enhanced the LCOM3 to include edges for method to method invocations as well and named it LCOM4. Henderson-Sellers [9] came up with LCOM5 which included the number of distinct attributes accessed by each method. The first direct cohesion metric, Connectivity (Co) was introduced by Hitz and Montazeri [10]. Briand et al in 1998 proposed Coh [18] which uses the concept of number of attributes used by each method. Bieman and Kang in 1995 [11] introduced TCC (Tight class cohesion) and LCC (Loose class cohesion) which captured the methods connected through attributes directly and indirectly. Badri in 2004 introduced variations on TCC and LCC naming them DC_D and DC_I [12] which includes method invocations also, directly or in the call trace respectively. The metrics that were proposed further were based on the similarity between methods based on the number of common attributes used between them. Bonja and Kidanmariam in 2006 defined Class Cohesion (CC) based on this similarity of methods [13]. The concept was further enhanced by Fernandez and Pena in 2006 [14] in their metric Sensitive Class Cohesion Metric (SCOM). Bansiya et al. [15] in 1999 defined Cohesion among Methods in a class (CAMC) as a modified version of Co. Counsell et al in 2006 introduced Normalized Hamming Distance (NHD) and Scaled Normalized Hamming Distance (SNHD) [16] based on how many attributes each method accesses. Al Dallal and Briand in 2009 came up with Low-level design Similarity based Class Cohesion (LSCC) [17] which finds out how many methods access each attribute.

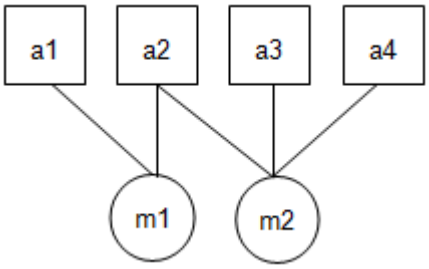


Fig. 1 Example of a Class with methods and attributes

3 Motivation

While the existing metrics capture cohesion and are used widely they have their drawbacks as well. In each case we explain the method to compute the metric and provide an example that highlights the drawbacks of that metric. A method attribute interaction diagram is provided in Fig. 1. The methods are represented as circular nodes and the attributes are represented as square nodes. A line between a method and an attribute indicates that the method accesses that attribute. In the sample class in Fig. 1 we have 2 methods and 4 attributes. Method m1 uses attributes a1 and a2 while method m2 uses attributes a2, a3 and a4. This gives the connection patterns of a class.

3.1 LCOM1 and LCOM2

LCOM1 measures the number of pairs of methods that do not share any common attribute. LCOM2 is computed as the number of pairs of methods that do not share any common attribute minus the number of method pairs that share at least one attribute. For all method pairs I_i and I_j in a class, we have.

$$P = \{|I_i, I_j|, I_i \cap I_j = \emptyset\}$$

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

$$LCOM1 = P$$

$$LCOM2 = \begin{cases} |P| - |Q|, & if\,|P| > |Q| \\ 0, & otherwise \end{cases}$$

Since LCOM1 and LCOM2 were the first cohesion metrics proposed, they fall short on many accounts. The first and foremost is the lack of a normalized form, as the values can grow exponentially and are directly proportional to the number of methods in a class.

This makes it very difficult to compare the cohesion between a pair of classes. Consider the two classes in Fig. 2. We can see that the class 2 has a lot more methods compared to the class 1. Though it seems that both the classes have relatively similar cohesion, the values are skewed due to the number of methods in the two classes. We have $LCOM1\,(class1) = LCOM2\,(class1) = 1$ and $LCOM1\,(class2) = LCOM2\,(class2) = 6$.

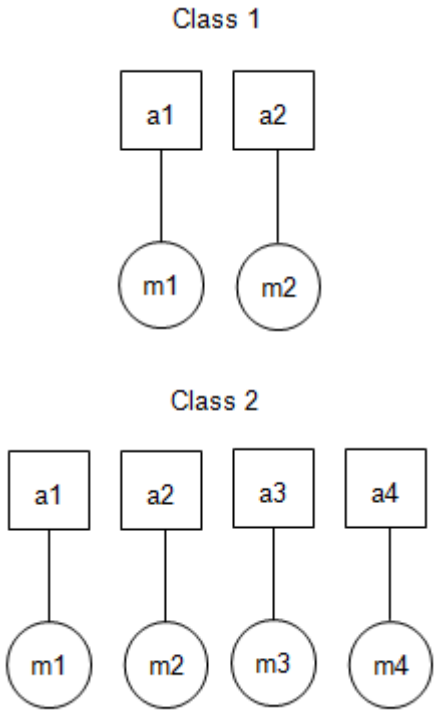


Fig. 2 LCOM1 and LCOM2 examples

3.2 LCOM3 and LCOM4

To compute LCOM3 a graph is constructed with methods as nodes and edges between methods sharing at least one attribute. LCOM3 is given as the number of connected components in the graph. LCOM4 is also computed similarly except that during the construction of the graph, edges are also added if one method calls another method. In LCOM3 and LCOM4 the drawback related to range of values has been improved. This is because both these metrics count the connected components.
This restricts their range of values compared to LCOM1 and LCOM2 which measured the method pairs. However the problem of totally different values for classes with similar cohesion still exists. For the example shown in Fig. 2 we have $LCOM3\,(class1) = LCOM4\,(class1) = 2$ and $LCOM3\,(class2) = LCOM4\,(class2) = 4$. This is not very intuitive. But a better example to highlight the drawback of LCOM3 and LCOM4 is given in Fig. 3. The class 3 contains 7 attributes and 6 methods. Each method accesses a unique attribute along with one common attribute a4. In essence there is almost no cohesion between the methods and each of them behaves like an isolated island. The common attribute could be a debug variable or a static counter variable. But we get $LCOM3\,(class3) = LCOM4\,(class3) = 1$ which is the best possible cohesion value for these metrics.

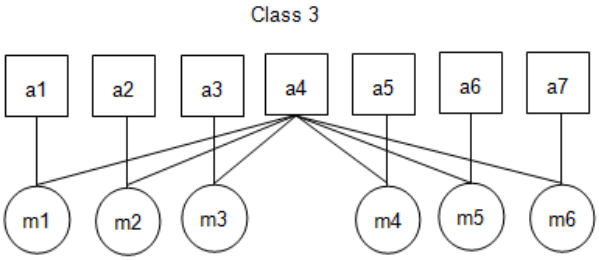


Fig. 3 Example for LCOM3 and LCOM4

3.3 TCC, LCC, DC_D, DC_I

TCC and LCC provide another way to measure the cohesion of a class by looking at relations between methods. Two methods ‘a’ and ‘b’ are related if they both access the same class level variable or if the call trees starting at ‘a’ and ‘b’ access the same class level variable. Once the graph is drawn with edges depicting relations, two methods are said to be directly connected if they have an edge between them. If they are connected through other methods then they are indirectly connected. If there are N methods, NDC represents number of direct connections; NIC represents the number of indirect connections, then

$$TCC = \frac{NDC}{(N * (N-1))}$$

$$LCC = \frac{(NDC + NIC)}{(N * (N-1))}$$

For DC_D, DC_I we consider an additional relationship of method invocation. So methods 'a' and 'b' are said to be connected if the two methods directly or indirectly invoke a third method. If NTC represents the number of transitive connections, then

$$DC_D = \frac{(NDC + NTC)}{(N * (N-1))}$$

$$DC_I = \frac{(NDC + NIC + NTC)}{(N * (N-1))}$$

The value for TCC, LCC, DC_D and DC_I is 1 for the example in Fig. 3. The reason behind the same value is that all these metrics provide highest cohesion between the sharing methods of the class even if just one attribute is shared across them.

3.4 CC and SCOM

CC and SCOM are different from the earlier cohesion metrics. They do not consider just a single common attribute sufficient for providing highest cohesion value between pairs of methods. Instead they consider the level of similarity between the pair of methods. Similarity between methods is calculated using the number of common attributes used between the pairs of methods. CC and SCOM differ in the denominator used to divide the common attributes. CC uses the distinct attributes between the methods whereas SCOM uses the minimum number of attributes used in the two methods. If N is the number of methods in a class, I_i and I_j refer to the method pairs, CC and SCOM are given by

$$CC = \frac{\sum_{i,j} \frac{|I_i \cap I_j|}{|I_i \cup I_j|}}{(N * (N-1))}$$

$$SCOM = \frac{\sum_{i,j} \frac{|I_i \cap I_j|}{Min(|I_i|, |I_j|)}}{(N * (N-1))}$$

Though both CC and SCOM deliver a far better measure for cohesion compared to the earlier metrics, they still fall short on a few parameters. When there are many method pairs that access very few attributes of the class, but have sufficient commonality, they tend to bloat the cohesion value. See class 4 in Fig. 4. The class has 5 attributes and 5 methods. Attribute a1 is accessed by 4 methods while the other attributes are accessed by method m5 alone. Careful observation would tell us that the class has a poor cohesion since the majority of the attributes are not shared across the methods but both CC and SCOM give better than average value. We have $CC\ (class4) = SCOM\ (class4) = 0.6$ (1 is max cohesion possible). So the actual reality of cohesiveness in the class is not fully captured by the metrics.

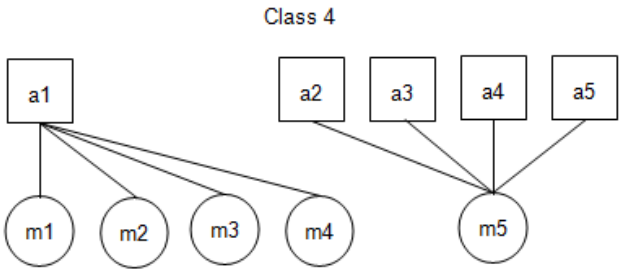


Fig. 4 Example for SCOM and CC

All the above reasons contributed to the motivation behind the need to explore a better metric, which would suit all cases and capture cohesion realistically

4 High Precision Cohesion Metric

We use Briand et al.'s Unified Framework for Cohesion (UFC) in 1998 [18] to present the new cohesion metric. First we propose two new concepts that will form the basic building blocks for the new cohesion metric.

4.1 Average Attribute Usage

The Average Attribute Usage (AAU) is a separate metric which computes the average number of attributes used by each method of the class. We consider only public, non-inherited methods of a class. As per UFC, the total of such methods are arrived at using the following approach. Let 'c' be the class in consideration. Then $M_I(c)$ is the set of non-inherited, overriding or newly implemented methods of c. Further $M_{pub}(c)$ is the set of public methods of c. Public non-inherited or overridden methods are given by $M_I(c) \cap M_{pub}(c)$.

The total number of methods in our case is the cardinality of such a set. It is given by $|M_I(c) \cap M_{pub}(c)|$. The attributes referenced by a method is given by $AR(m)$where m is the method. Hence total attributes referenced is given by $\sum AR(m)$, for each 'm' in the set $M_I(c) \cap M_{pub}(c)$. Hence the AAU is given as

$$AAU = \frac{\sum AR(m)}{|M_I(c) \cap M_{pub}(c)|}$$

The AAU for class 4 in Fig. 4 is 1.6.

4.2 Link Strength

The Link Strength (LS) is based on the AAU. Link Strength for a pair of methods m1 and m2 is given by

$$LS_{m1m2} = \begin{cases} \frac{|AR(m1) \cap AR(m2)|}{AAU}, & if\ |AR(m1) \cap AR(m2)| \leq AAU \\ 1, & if\ |AR(m1) \cap AR(m2)| > AAU \end{cases}$$

LCOM4 considers the methods as nodes of a graph. Two nodes are connected with an edge if the underlying methods have at least one attribute in common. Consider the example of class 5 in Fig. 5. The class consists of 6 attributes and 3 methods. A LCOM4 graph is drawn to depict the same in Fig. 6

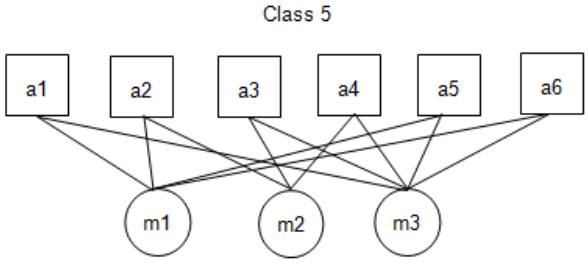


Fig. 5 Example for Link Strength

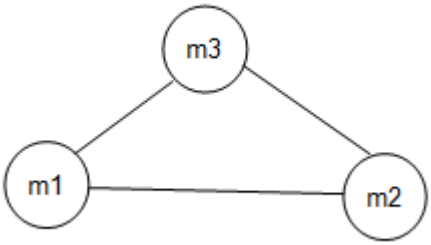


Fig. 6 LCOM4 Graph for Class 5

As expected the graph shows a single connected component giving an LCOM4 value of 1. On observing the class we find that method m1 and method m2 share a single attribute a2. Method m2 and Method m3 share two attributes a3 and a4. Method pairs m1 and m3 share three attributes a1, a5, a6. So the linkage between m1 and m2 is not the same as that of m2 and m3 which in turn is different from the link between m1 and m3. The Link Strength metric is specifically meant to capture this difference in the strength of the links between methods. The graph for LCOM4 metric has been modified to include link strength in Fig. 7. It is clear that though all three methods are linked, the level or strength of their links is different and this is captured in LS

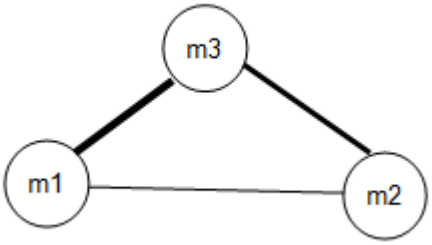


Fig. 7 Link Strength for Class 5

4.3 Definition of HPCM

The new cohesion metric has been named High Precision Cohesion Metric (HPCM) to denote its capability to distinguish and assign precise cohesion values for a wide variety of classes with various method-attribute interactions. It depends on the earlier proposed metrics AAU and LS. The formula for HPCM is given

$$HPCM(c) = \frac{2 * \sum LS_{m_i m_j}}{|M_I(c) \cap M_{pub}(c)| * (|M_I(c) \cap M_{pub}(c)| - 1)}$$

HPCM is an average of LS of all method pairs in the class. To find the average we first find the total of all LS in the numerator which is done by $LS_{m_1 m_2}$. Here all public non inherited, overridden and newly implemented methods are considered, similar to AAU and their link strengths are summed up. The denominator denotes the total available method pairs and is given by a simple formula of n * (n - 1)/2 where 'n' is the total number of methods.

In our case the total number of methods is given by $M_I(c) \cap M_{pub}(c)$ and hence results in the denominator of $|M_I(c) \cap M_{pub}(c)| * (|M_I(c) \cap M_{pub}(c)| - 1)$. The final division by 2 gets back to the numerator. Below is an example showing the calculation of HPCM for the class 5 given in Fig. 5. The first step would be to decide how many methods are to be considered for the metric. We see that $M_I(c) \cap M_{pub}(c) = 3$. The next step would be to find AAU.

The first method uses 4 attributes, the second uses 3 attributes and the third uses 5. Thus the total attribute usage is 12 and the average usage would be (4 + 3 + 5)/3 = 4. Hence AAU = 4. Having found AAU, we can proceed to find the link strength between each pair of methods. In our case since the number of methods considered is 3, we have 3 * (3 - 1)/2 = 3 method pairs. Below we show the calculation steps for the first method pair m1 and m2. Table 1 shows the link strength values for all method pairs.

$$LS_{m_1m_2} = \frac{|AR(m_1) \cap AR(m_2)|}{AAU}$$
$$= {}^{1}/_{4}$$
$$= 0.25$$

Table 1 Link Strength of method pairs for class in Fig. 5

Method Pair	Common Attributes	Link Strength
m_1m_2	1	0.25
m_1m_3	3	0.75
m_2m_3	2	0.50
Total	6.00	1.50

Finally we proceed to find the High Precision Cohesion Metric.

$$HPCM = \frac{2 * \sum LS_{m_im_j}}{|M_I(c) \cap M_{pub}(c)| * (|M_I(c) \cap M_{pub}(c)| - 1)}$$
$$= \frac{2 * 1.50}{3 * (3 - 1)}$$
$$= \frac{1.50}{3}$$
$$= 0.50$$

5 Criteria of Application

Metrics tend to be plagued by poor definition, assumptions and lack of application criteria. This causes difficulty in using the metric and can result in multiple interpretations by different users. In this section we put forward the criteria of application for HPCM.

5.1 Class Level Design Metric

Metrics can be specified at various levels like system, class or method level. HPCM is a class level metric. Metrics can also be classified based on the phase in which they are computed in a life cycle of the software product. HPCM is a design phase metric.

5.2. Inheritance

We only consider public methods which are directly implemented in this class. This also includes methods that are inherited and overridden. In short we avoid methods that are inherited but not overridden and those that are private. Similarly we only consider attributes that are introduced by the class and avoid the attributes that are got through inheritance.

5.3. Methods not accessing any attribute

Two kinds of methods do not access any class attribute. Static members accessing only static attributes are not allowed to access any class attributes. Similarly pure virtual methods not implemented in this class will not access any attribute. We do not consider such methods while computing HPCM.

5.4. Attributes not accessed by any method

This is allowed only in inheritance where the child class might use an attribute not used by its parent. However that attribute should not be considered while computing the cohesion of the parent class. We do not consider such attributes while computing HPCM.

5.5. Access Methods

The get/set methods typically deal with a single attribute and are used as access methods to set or get the value of that attribute. They should not be considered for computing HPCM as they would skew the cohesion value.

5.6. Constructors and Destructor

Constructor and destructor methods by design access many attributes for initializing or freeing them. Since they do not contribute to the cohesion we suggest excluding them for computing HPCM.

5.7. Method Invocations

In HPCM we do not consider method invocations as part of the relationships between methods. If method M_i calls M_j it does not add to the link strength between them. However we consider the transitive access of attributes that result due to method invocations. Consider that method M_i calls method M_j and if M_j accesses an attribute a_k, then we can consider that method M_i transitively accesses attribute a_k and is taken into account while measuring link strength.

6 Mathematical Validation

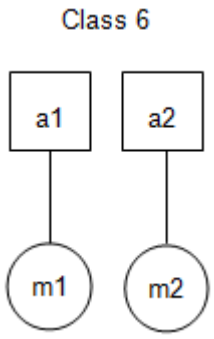
We use the four properties advocated as part of theoretical validation by Briand et al. in 1998 [18] in his Unified Framework for Cohesion.

6.1. Non-Negativity and Normalization.

The minimum value of HPCM is 0. This happens when there are no attributes or when there are no common attributes between the pairs of methods. The maximum value of HPCM is 1, when all methods share at least AAU attributes between them. Thus $HPCM(c) \in [0;1]$ where 'c' is the class of an object oriented system C.

6.2. Null value and maximum value.

Let c be the class of an object oriented system C and IR is the set of all interactions in the class. The cohesion of a class is null if IR is empty, that is there are no shared attributes between the methods. Refer to class 6 in Fig. 8. We see that IR results in an empty set or link strength is 0 for all the method pairs and hence the HPCM gets a corresponding null value of 0 i.e. $HPCM(class6) = 0$. Alternatively the class 7 in Fig. 8 results in a set with all possible members in the set IR. In our case we refer to links with full possible strength. This gives the maximum possible value for HPCM i.e. $HPCM(class7) = 1$.



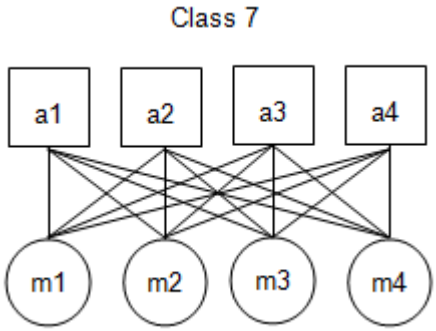


Fig. 8 Example for null and maximum value of HPCM

6.3. Monotonicity

Consider a class c in the object oriented system C. If we modify the class c to form a new class c' which is identical to c except that few more attributes are shared by methods. So, on adding a few more relationships to the existing class relationships we have better link strength with increase in numerator. Hence HPCM also increases. $HPCM(c) \leq HPCM(c')$. So HPCM satisfies the monotonicity property.

6.4. Merging unconnected classes

Let C be an object oriented system and $c1, c2 \in C$. Let c' be the class which is the union of c1 and c2. Let C' be the object-oriented system which is identical to C except that c1 and c2 are replaced by c'. If no relationships exist between classes c1 and c2 in C, then $max(HPCM(c1), HPCM(c2)) \geq HPCM(c')$.

7 Empirical Validations

In this section we present the analyses that we did to discover the relationship of HPCM with other cohesion metrics and its ability as an accurate indicator of faults in classes. We performed two types of analyses. The first analysis was targeted towards finding whether HPCM was contributing new information in finding the cohesion of classes. The second analysis was focused on building models to validate HPCM's ability to predict faults.

7.1. Data Set

We considered four open source projects written in Java from the sonar source repository website [19] for our validation. The projects were chosen such that they had close to 20 classes each for a fair analysis. The projects we chose were Spojo, Maven Plugin, XDoclet, and Sonar Plugin.

7.2. Cohesion Metrics considered

The following cohesion metrics were taken for this empirical validation - LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, TCC, LCC, CC, SCOM, and the proposed HPCM.

Table 2 Descriptive statistics for the cohesion metrics

Metrics	Min	25Q	Mean	Median	75Q	Max	StdDev
LCOM1	0.00	3.00	72.94	13.00	63.00	683.00	139.02
LCOM2	0.00	1.00	62.64	7.00	44.00	663.00	133.33
LCOM3	1.00	2.00	4.06	3.00	4.00	19.00	3.80
LCOM4	1.00	1.00	2.88	2.00	3.00	17.00	2.94
LCOM5	0.00	0.63	0.75	0.79	0.91	2.00	0.33
TCC	0.00	0.12	0.29	0.17	0.33	1.00	0.27
LCC	0.00	0.13	0.35	0.24	0.50	1.00	0.33
SCOM	0.00	0.12	0.29	0.19	0.40	1.00	0.27
CC	0.00	0.06	0.20	0.13	0.31	1.00	0.22
HPCM	0.00	0.02	0.20	0.11	0.25	1.00	0.26

7.3. Data Collection Procedure

To compute the cohesion metrics, we used an automated tool [20]. This tool was developed by us to aid in our research work. The tool takes in source code of object oriented programs as input and computes the relevant metrics. We added the ability to compute HPCM in addition to the already existing list of object oriented metrics in the tool. The bug data was available online in the Sonar source website. The bugs were classified as major and minor. For our study we concentrated only on the major bugs. The source code of the projects considered were downloaded and used to extract the metrics. Table 2 gives the snapshot of the metrics computed with the minimum, 25 percentage quartile, mean, median, 75 percentage quartile, max and standard deviation.

7.4. Correlation Results

Correlation studies are required to evaluate whether two variables are related and if they are trending. Pearson coefficient gives the correlation between two variables X and Y in the range of +1 to -1. A value of 1 indicates perfect positive correlation which means that the variables are tracking the same information. A value of -1 indicates perfect negative correlation.

It means that the variables are tracking the same information but in opposite direction. A value of 0 indicates zero correlation which means that the variables are independent and are tracking different information. We calculated the Pearson coefficients between all pairs of cohesion metrics and their p-values. p-values indicate statistical significance, which is the probability that the coefficient is different from zero by chance. When p-value is closer to zero, it means that the coefficient is highly significant. Table 3 shows the Pearson coefficients for the cohesion metrics. High correlation coefficients which also have significant p-value ($p<0.0001$) have been highlighted. Correlations between LCOM1, LCOM2, LCOM3 and LCOM4, between LCOM5, TCC, LCC, SCOM and CC are greater than 0.7. If a metric has low Pearson coefficients, it indicates that the metric is capturing unique information regarding cohesion compared to other metrics. We see that HPCM is capturing unique information.

Table 3 Pearson Coefficients for Cohesion Metrics

Metrics	LCOM2	LCOM3	LCOM4	LCOM5	TCC	LCC	SCOM	CC	HPCM
LCOM1	**0.99**	**0.91**	**0.81**	0.24	-0.33	-0.33	-0.36	-0.31	-0.14
LCOM2	1.00	**0.92**	**0.82**	0.24	-0.33	-0.33	-0.36	-0.31	-0.16
LCOM3		1.00	**0.82**	0.30	-0.45	-0.46	-0.49	-0.43	-0.27
LCOM4			1.00	0.25	-0.39	-0.40	-0.41	-0.35	-0.15
LCOM5				1.00	-0.64	-0.58	**-0.71**	**-0.75**	-0.53
TCC					1.00	**0.96**	**0.95**	**0.86**	0.55
LCC						1.00	**0.91**	**0.74**	0.41
SCOM							1.00	**0.91**	0.53
CC								1.00	0.31

7.5 Univariate Models

Precise fault prediction is considered as an important criterion in a metric's indicator of quality [2], [21] [22] [23]. In this analysis, we evaluate how each cohesion metric fares individually in their capability of predicting faults. Since the focus of this study is on metrics we did not try to evaluate many fault prediction models. We chose the widely accepted and used linear regression for the univariate analysis. Linear regression maps the cohesion values to the bug data, where Y is the dependent variable (bugs), X is the independent variable (metrics), 'a' is the constant term called intercept and 'b' is the coefficient.

$$Y = a + bX$$

Linear regression models can be evaluated based on a number of indicators. We use Root Mean Square error (RMSE). RMSE measures the differences between the values predicted by the linear regression model and the actual values. The lower the RMSE the better the fit of the model. The equation for RMSE is given below where x_m is the predicted value by the model and x_a is the actual value and 'n' is the number of observations.

$$RMSE = \sqrt{\frac{\sum(x_m - x_a)^2}{n}}$$

The results of the univariate linear regression are given in the Table 4. HPCM is a significant contributor with the RMSE of 0.37.

Table 4 Results of Univariate Linear Regression

Metric	Coefficient	Constant	RMSE
LCOM1	.001	.54	.51
LCOM2	.001	.54	.51
LCOM3	.05	.39	.48
LCOM4	.05	.46	.53
LCOM5	.72	.07	.43
TCC	-1.19	.95	.38
LCC	-.91	.92	.39
SCOM	-1.19	.95	.39
CC	-1.46	.89	.41
HPCM	-1.32	.87	.37

7.6. Multivariate Models

The goal of this analysis is to confirm that when HPCM is included, the fault prediction accuracy improves in models developed using other cohesion metrics. We developed pairs of multivariate models, one with all the cohesion metrics except HPCM and another including HPCM.

We chose three different multivariate models - linear regression, Bayesian network and Decision trees. Multivariate Linear Regression maps more than one independent variable to a dependent variable. 'Y' is the dependent variable and 'a' is the constant term. b1,b2...bn are the coefficients for the independent variables X1,X2...Xn respectively.

$$Y = a + b_1X_1 + b_2X_2 + \cdots b_nX_n$$

Table 5 shows the details of the prediction with and without HPCM for multivariate linear regression. We find that by adding HPCM the root mean square error improves from 3.85 to 2.90.

Table 5 Results of Multivariate Linear Regression

Model	RMSE	
	Without HPCM	With HPCM
Multivariate Linear Regression	3.85	2.90

Bayesian Network is a probabilistic graphical model that represents a set of random variable and their conditional dependencies. We have used it to map the cohesion metrics and bug data. On providing the cohesion metrics the model will output the probability of occurrence of bug. Decision Trees represents decision making with branches relating to decisions and leaves representing the result. For each set of input the tree is traversed and the corresponding leaf is chosen. Table 6 shows the details of the prediction with and without HPCM for Decision tree and Bayesian network. The prediction accuracy of models with HPCM is better compared to the prediction accuracy of models without HPCM.

Table 6 Results of Bayesian and Decision Tree models

Model	Prediction Accuracy	
	Without HPCM	With HPCM
Decision Tree	71	78
Bayesian Net	50	60

7.7. Threats to Validity

The empirical validation that we have done in this section is prone to some limitations. We have considered open source code in Java. This data set may not be representative of real time large object oriented software systems. A thorough validation has to consider programs of different sizes and types and from different domains. Another obvious threat is modeling faults with cohesion metrics alone. There could be other factors like coupling which would also contribute to quality. However if we consider that the effect of external attributes is constant, our validation holds.

8 Conclusions

In this study, we have identified the problems with the existing cohesion metrics. We then presented a new cohesion metric called High Precision Cohesion Metric (HPCM) which attempts to address the short comings of the earlier metrics. The Unified Framework for Cohesion was used to present the metric to avoid ambiguity. The criteria of application of the metric were also provided. We subjected the metric to a rigorous mathematical validation advocated by Briand et al. This was followed by empirical validations with open source java projects. HPCM fared well both as a standalone metric in univariate models as well as a contributing metric in multivariate models. Future study areas include applying the HPCM along with other metrics to evaluate software quality and see the effectiveness of the metric in commercial software. The same concepts of HPCM can be used to present a complimentary metric for coupling between classes.

References:

[1] Z. Chen, Y. Zhou, B. Xu, "A novel approach to measuring class cohesion based on dependence analysis" *Proceedings of the International Conference on Software Maintenance*, 2002, pp. 377-384.

[2] L. C. Briand, C. Bunse, J. Daly, "A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs" *IEEE Transactions on Software Engineering*, Vol. 27, 2001, pp. 513-530.

[3] J. Bieman, L. Ott, "Measuring functional cohesion" *IEEE Transactions on Software Engineering*, Vol. 20, 1994, pp. 644-657.

[4] T. Mens, S. Demeyer, "Future trends in software evolution metrics" *Proceedings of IWPSE2001, ACM*, 2002, pp. 83-86.

[5] S. R. Chidamber, C. F. Kemerer, "Towards a metrics suite for object-oriented design" *Proceedings of Conference on Object-Oriented Programming Systems, Languages and Applications*, 1991, pp. 476-493.

[6] S. R Chidamber, C. F. Kemerer, "A metrics suite for object-oriented design" *IEEE Transactions on Software Engineering*, 1994, pp. 476-493.

[7] W. Li, S. Henry, "Object-oriented metrics that predict maintainability" *Journal of Systems and Software*, 1995, pp. 111-122.

[8] M. Hitz, B. Montazeri, "Chidamber and Kemerer metric suite - a measurement theory perspective" *IEEE Transactions on Software Engineering*, 1996, pp. 267-271.

[9] B. S. Henderson, "Object-oriented Metrics: Measure of Complexity" *New Jersey, Prentice Hall*, 1996, pp. 142-147.

[10] M. Hitz, B. Montazeri, "Measuring coupling and cohesion in object oriented systems" *Proceedings of the Int. Symposium on Applied Corporate Computing*, 1995, pp. 25-27.

[11] M. M. Bieman, B. K. Kang, W. Melo, "Cohesion and reuse in an object oriented system" *Proceedings of the symposium on software reliability*, 1995, pp. 259-262.

[12] L. Badri, M. Badri, "A Proposal of a new class cohesion criterion, an empirical study" *Journal of Object Technolog,* Vol. 3. No. 4. 2004, pp. 145-159.

[13] C. Bonja, E. Kidanmariam, "Metrics for class cohesion and similarity between methods" *Proceedings of the 44th Annual ACM Southeast Regional Conference*, 2006, pp. 91-95.

[14] L. Fernandez, R. Pena "A sensitive metric of class cohesion" *International Journal of Information Theories and Applications*, Vol. 13, 2006, pp. 82-91.

[15] J. Bansiya, L. Etzkorn, C. Davis, W. Li "A class cohesion metric for object-oriented designs" *Journal of Object Oriented Program*, Vol. 11, 1999, pp. 47-52.

[16] S. Counsell, S. Swift, J. Crampton "The interpretation and utility of three cohesion metrics for object-oriented design" *ACM Transactions on Software Engineering and Methodology*, Vol. 15, 2006, pp. 123-149.

[17] A. J. Dallal, L. Briand, "A Precise method-method interaction based cohesion metric for objectoriented classes" *Simula Research Laboratory, Simula Technical Report*, 2009.

[18] L.C. Briand, J. Daly, J. Wuest, "A unified framework for cohesion measurement in object-oriented systems" *Empirical Software Engineering, An International Journal*, 1998, pp. 65-117

[19] SonarSource, "Sonar Project Source and Bug Repository" http://nemo.sonarsource.org (visited September 2012)

[20] N. Kayarvizhy, S. Kanamani, "An Automated Tool for Computing Object Oriented Metrics using XML" *Proceedings of International Conference on Advances in Computing and Communication ACC2011, Springer,* 2011, Vol. 191, pp. 69-79.

[21] T. Gyimothy, R. Ferenc, I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction" *IEEE Transactions on Software Engineering*, Vol. 3, 2009, pp. 897-910.

[22] K. Aggarwal, Y. Singh, A. Kaur, R. Malhotra, "Investigating effect of design metrics on fault proneness in object-oriented systems" *Journal of Object Technology*, Vol. 6, 2007, pp. 127-141.

[23] A. Marcus, D. Poshyvanyk, R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems" *IEEE Transactions on Software Engineering*, Vol. 34, 2008, pp. 287-300.