A Proposal for Normalized Lack of Cohesion in Method (LCOM) Metric Using Field Experiment

Ezekiel Okike

School of Computer Studies, Kampala International University ,
Kampala, Uganda 256, Uganda

Abstract

Chidamber and Kemerer first defined a cohesion measure for object-oriented software – the Lack of Cohesion in Methods (LCOM) metric. One of the critique of the LCOM metric is that the metric does not yield normalized or standardized values, and as such, the metric does not seem appealing to a section of the software engineering community. This paper presents an approach for normalizing the LCOM metric so that most practioners would find it as useful as its variant measures such as Tight Class Cohesion (TCC), Low Class Cohesion (LCC), Degree of Cohesion in a Class based on direct relation between its public relations (DCD) and that based on indirect methods (DCI). Data for this study was gathered from three industrial systems. System 1 has 34 classes, System 2 has 383 classes and System 3 has 1055 classes. The main objectives of the study were to apply different normalization approaches in order to determine the best for the LCOM metric. Three normalization techniques namely Sigmoid normalization, Bowless normalization, and Bestfit normalization were used in the study of the selected test systems. The result of the study showed that the Bestfit approach seem to be the best LCOM normalization approach.

Keywords: *Class Cohesion, LCOM Metric, Normalization, Software Measurement.*

1. Introduction

The Lack of Cohesion in Methods (LCOM) metric was proposed in [6,7] as a measure of cohesion in the object oriented paradigm.

The term cohesion is defined as the “intramodular functional relatedness” in software [1]. This definition, considers the cohesion of each module in isolation: how tightly bound or related its internal elements are. Hence, cohesion as an attribute of software modules capture the degree of association of elements within a module, and the programming paradigm used determines what is an element and what is a module. In the object-oriented paradigm, for instance, a module is a class and hence cohesion refers to the relatedness among the methods of a class. Cohesion may be categorized ranging from the weakest form to the strongest form in the following order: coincidental, logical, temporal, procedural, communicational, sequential and functional. Further discussion on these categories are already presented in [1, 2].

A module exhibits one of these forms of cohesion depending on the skill of the designer. However, functional cohesion is generally accepted as the best form of cohesion in software design. Functional cohesion is the most desirable because it performs exactly one action or achieves a single goal. Such a module is highly reusable, relatively easy to understand (because you know what it does) and is maintainable. In this paper, the term “cohesion” refers to functional cohesion. Several measures of cohesion have been defined in both the procedural and object-oriented paradigms. Most of the cohesion measures defined in the object-oriented paradigm are inspired from the Lack of Cohesion in methods (LCOM) metric defined by Chidamber and Kemerer.

1.2. The problem.

The LCOM metric defined by Chidamber and Kemerer is often critised due to the presence of outliers and for not being a normalized measure [3,4,8,11,14]. Based on this, a section of the software engineering community seem not to find the metric appealing in usage.

In this paper, the Lack of Cohesion in methods (LCOM) metric is analysed, and a Bestfit normalized LCOM metric is proposed. The rest of the paper is organized as follows: Section 2 presents a summary of the approaches to measuring cohesion in procedural and object-oriented programs. Section 3 examines the Chidamber and Kemerer LCOM metric. Section 4 presents the application of three normalization techniques on the LCOM metric. Section 5 presents the result of the study of which the Bestfit LCOM metric is suggested as a normalized metric to be used with the LCOM metric.

2. Measuring Cohesion in Procedural and Object oriented Programs

2.1 Measuring cohesion in procedural programs

Procedural programs are those with procedure and data declared independently. Examples of purely procedure oriented languages include C, Pascal, Ada83, Fortran and so on. In this case, the module is a procedure and an element is either a global value which is visible to all the modules or a local value which is visible only to the module where it is declared. As noted in [2], the approaches taken to measure cohesiveness of this kind of programs have generally tried to evaluate cohesion on a procedure by procedure basis, and the notational measure is one of "functional strength" of procedure, meaning the degree to which data and procedures contribute to performing the basic function. In other words the complexity is defined in the control flow. Among the best known measures of cohesion in the procedural paradigm are discussed in [4] and [5].

2.2 Measuring cohesion in object-oriented systems

In the Object Oriented languages, the complexity is defined in the relationship between the classes and their methods. Several measures exist for measuring cohesion in Object-Oriented systems [7,8,9,10,11,12, 13]. Most of the existing cohesion measures in the object-oriented paradigm are inspired from the Lack of Cohesion in Methods (LCOM) metric [6,7]. Some examples include LCOM3, Connectivity model, LCOM5, Tight Class Cohesion (TCC), and Low Class Cohesion (LCC), Degree of Cohesion in class based on direct relation between its public methods (DCD) and that based on indirect methods (DCI), Optimistic Class cohesion (OCC) and Pessimistic Class Cohesion (PCC).

3. The Lack of Cohesion in Methods (LCOM) Metric.

The LCOM metric is based on the number of disjoint sets of instance variables that are used by the method. Its definition is given as follows [6,7].

Definition 1.

Consider a class C1 with n methods M_1, M_2,…,Mn. Let {Ii}= set of instance variables used by method Mi. There are n such sets {Ii},…,{In}. Let $P = \{ (Ii, Ij) \mid Ii \cap Ij = \phi\}$ and $Q = \{ (Ii, Ij) \mid Ii \cap Ij \neq \phi \}$. If all n sets { I_1}, …,{In} are ϕ then let $P = \phi$

$$LCOM = \{ |P|- |Q|, \text{ if } |P| > |Q|$$

$$= 0, \text{ otherwise}$$

Example: Consider a class C with three methods M_1, M_2 and M_3. Let $\{I_1\} = \{a,b,c,d,e\}$ and $\{I_2\} = \{a,b,e\}$ and $\{I_3\} = \{x,y,z\}$. $\{I_1\} \cap \{I_2\}$ is nonempty, but $\{I_1\} \cap \{I_3\}$ and $\{I_2\} \cap \{I_3\}$ are null sets. LCOM is (the number of null intersections – number of non empty intersections), which in this case is 1.

The theoretical basis of LCOM uses the notion of degree of similarity of methods. The degree of similarity of two methods M_1 and M_2 in class C_1 is given by:

$$\sigma( ) = \{I_1\} \cap \{I_2\}$$

where $\{I_1\}$ and $\{I_2\}$ are sets of instance variables used by M_1 and M_2 . The LCOM is a count of the number of method pairs whose similarity is 0 (i.e, σ( ) is a null set) minus the count of method pairs whose similarity is not zero. The larger the number of similar methods, the more cohesive the class, which is consistent with the traditional notions of cohesion that measure the inter relatedness between portions of a program. If none of the methods of a class display any instance behaviour, i.e. do not use any instance variables, they have no similarity and the LCOM value for the class will be zero. The LCOM value provides a measure of the relative disparate nature of methods in the class. A smaller number of disjoint pairs (elements of set P) implies greater similarity of methods. LCOM is intimately tied to the instance variables and methods of a class, and therefore is a measure of the attributes of an object class.

In this definition, it is not stated whether inherited methods and attributes are included or not. Hence, a refinement is provided as follows [15]:

Definition 2.

Let P = ϕ, if AR (m) = ϕ $\forall$ m $\in$ M_I (c)

= $\{\{m_1,m_2\} \mid m_1,m_2 \in M_I(c) \wedge m_1 \neq m_2 \wedge AR(m_1) \cap AR(m_2) \cap A_I(c) = \phi\}$, else

Let Q = $\{\{m_1,m_2\} \mid m_1,m_2 \in M_I(c) \wedge m_1 \neq m_2 \wedge AR(m_1) \cap AR(m_2) \cap A_I(c) \neq \phi\}$

Then LCOM2(c) = { $|P| - |Q|$, if $|P| > |Q|$

= 0, otherwise

Where M_I are methods in the class c and A_I are the attributes (or instance variables) in the class c ; AR denote attribute reference

In this definition, only methods M implemented in class c are considered; and only references to attributes AR implemented in class c are counted.

The definition of LCOM2 has been widely discussed in the literature [7,8,10,11,15]. LCOM2 of many classes are set to be zero although different cohesions are expected.

3.1 Remarks

In general the Lack of Cohesion in Methods (LCOM) measures the dissimilarity of methods in a class by instancevariable or attributes. Chidamber and Kemerer's interpretation of the metric is that LCOM = 0 indicates acohesive class. However, for LCOM >0, it implies thatinstance variables belong to disjoint sets. Such a class maybe split into 2 or more classes to make it cohesive.

Consider the case of an n-sequentially linked methods as shown in figure 3.1 below where n methods are sequentially linked by shared instance variables.

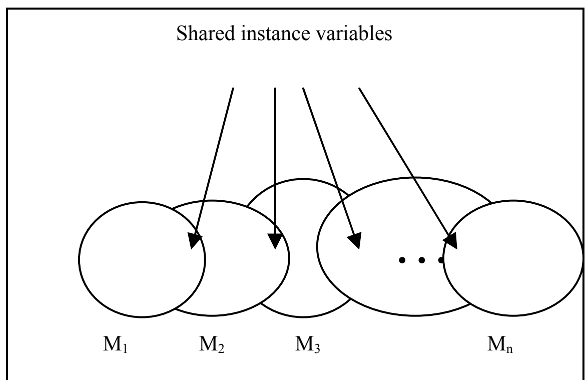

Fig. 3.1. n-Sequentially liked methods

In this special case of sequential cohesion:

$$P = \binom{n}{2} - (n-1) \quad (1)$$

$$Q = n - 1 \quad (2)$$

so that LCOM

$$P - Q = \left| \binom{n}{2} - 2(n-1) \right|_{+} \quad (3)$$

where $[k]^+$ equals k, if k>0 and 0 otherwise [8].

From (1) and (2)

$$P - Q = \binom{n}{2} - (n-1) - (n-1)$$

$$= \binom{n}{2} - 2n - 2$$

$$= \binom{n}{2} - 2(n-1)$$

$$= \frac{n!}{(n-2)!2!} - 2(n-1) \qquad (4)$$

From (4), for $n < 5$, LCOM = 0 indicating that classes with less than 5 methods are equally cohesive. For $n \geq 5$, $1 <$ LCOM $< n$, suggesting that classes with 5 or more methods need to be split [8,18].

3.2 Class design and LCOM computation

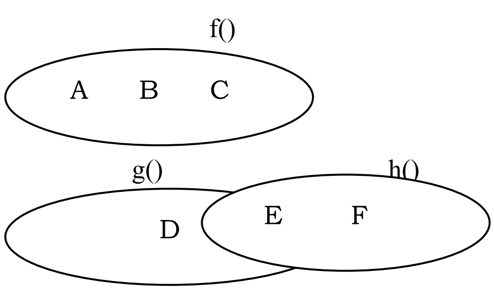

Fig. 3.2. Class design showing LCOM computation

Source:[8]

```
Class x {
  Int A, B, C, D, E, F;
  Void f() {…uses A,B, C …}
  Void g () {…uses D, E…}
```

Figure 3.2 presents a class x written in C++.

The Lack of Cohesion in Methods (LCOM) for class x = 1, calculated as follows:

There are two pairs of methods accessing *no* common instance variables namely (<f, g>, <f, h>). Hence P = 2. One pair of methods shares variable E, namely, <g, h>. Hence, Q = 1. Therefore, LCOM is 2 - 1 =1.

4. Normalizing LCOM metric

4.1 The Method

Chidamber and Kemerer's suit of metrics were used in the study of 1055 classes from three industrial systems. Metric values were computed for Lack of Cohesion in methods (LCOM), Coupling Between Object Classes (CBO), Response For a Class (RFC),Weighted Methods Per Class (WMC), Depth of Inheritance (DIT) and Number of Children (NOC) were used in the study. Two other metrics used in this experiment which are not part of the Chidamber and Kemerer metrics are: Number of Public Methods (NPM) and Afferent Coupling (CA).

Specifically cohesion was measured using the LCOM metric. Coupling was measured using CBO, RFC, and CA. Size was measured using WMC, and NPM. Inheritance was measured using DIT. Descriptive statistics was used to analyze results as already presented in [2]. In order to demonstrate the need for a normalized LCOM metric, 11 classes from system 3 most of which were outliers were used. Of particular note are the LCOM values of these classes as shown in table 4.1 below:

4.2 Outliers in LCOM metric

Table 4.1 below shows LCOM values with outliers. LCOM = 0 indicates a cohesive class [6,7], likewise LCOM= [0,1] [2,19]. Chidamber and Kemerer suggested that when a class is not cohesive, the class should be split into two or more classes to make the class cohesive. In this study, splitting outlier classes only reduced the outlier values and never made such classes cohesive, and there were no evidences to suggest such classes were improperly designed. This is illustrated by taking data from the first eight classes of system 3 in the experimental study as shown in figures 4.2 (a) and (b). The fourth class with LCOM value 196 was used as highlighted in figure 4.2 (a). below. The class was split into two classes (Classmetrics and Classinfo) to see if the outlier value 196 would become a cohesive class in the range values of 0 or 1. The result as shown in figure 4.2 (b) indicate that the outlier value was only reduced to 63 in Classmetrics while the LCOM value for Classinfo was 34. Both values are still outliers, implying the classes were still un cohesive even after splitting.

Table 4.1. Illustration of LCOM values with outliers and not standardized (un normalized)

Class Name	WMC	CBO	RFC	LCOM	CA	NPM
XlmException	5	0	6	2	1	5
HandlerBase	14	2	16	91	0	14
SAXDriver	33	2	86	370	0	31
XmlHandler	13	0	13	78	3	13
XmlParser	118	1	182	6075	1	27
JspXMLParser	2	5	18	0	1	1
CompiledException	3	0	5	1	2	0
JspServlet$Page	11	14	154	0	1	0
JspServlet$MapEntry	1	1	2	0	1	0
JspMsg	0	0	0	0	9	0
JspFactoryImpl$1	2	0	3	1	1	1

Source: [19]

Figure 4.2 (a) and (b) show the output after running a Chidamber and Kemerer metric tool ckjm (discussed in [19]) on the selected classes.

After the word ckjm is the class name being analyzed by the tool followed by the corresponding metrics for the class: WMC, DIT, NOC, CBO, RFC, LCOM, CA , NPM in that order. The metric values after the class name correspond to these metrics respectively.

```
[eokike@visitor2 build]$ java -jar ckjm-1.6.jar /tmp/gr/spinellis/ckjm/*.class
gr.spinellis.ckjm.ClassMetricsContainer 3 1 0 3 18 0 2 2
        .spinellis.ckjm.MethodVisitor 11 1 0 21 40 0 1 8
        spinellis.ckjm.CkjmOutputHandler 1 1 0 1 1 0 3 1
        .spinellis.ckjm.ClassMetrics 24 1 0 0 33 196 6 23
        gr.spinellis.ckjm.MetricsFilter 7 1 0 6 30 11 2 5
        gr.spinellis.ckjm.ClassVisitor 13 1 0 14 71 34 2 9
        gr.spinellis.ckjm.ClassMap 3 1 0 1 21 0 0 2
        gr.spinellis.ckjm.PrintPlainResults 2 1 0 2 8 0 1 2
        [eokike@visitor2 build]$
```

Figure 4.2 (a). Spliting an outlier class

```
        .spinellis.ckjm.ClassMetrics 14 1 0 0 16 63 1 13
        JavaParser$ButtonHandler 3 1 0 2 5 3 1 1
        Classidentifier 3 1 0 0 7 0 1 3
        JavaParser 5 1 0 3 59 2 1 3
        gr.spinellis.ckjm.Classinfo 12 1 0 1 27 34 0 12
        JavaParser$1   0 1 0 0 0 0 2 0
```

Figure 4.2 (b). Result of splitting an outlier class

Sources: [19]

In [19], there is no evidence to suggest that a class whose LCOM=1 was improperly designed and therefore needs splitting. Hence, it is suggested that the basis for splitting a class should be whenever the number of methods (Number of Public Methods NPM) is greater than or equal to 5 [9,19].

4.3 Normalization of software measures

4.3.1 Related Work

The normalization of measures is an important aspect of software measurement. Measures with outlier values need to be normalized in order to get numbers between [0,1]. However, as stated in [16] the normalization of a measure u to a normalized measure u' can lead to a completely new measure with other properties.

In [18] a measure IS is defined in order to capture the complexity of a module derived from the interactions (coupling) with other modules. The measure IS was the original measure. In order to get numbers between 0 and 1 the measure CM was defined as follows:

$$CM = 1 - \frac{1}{1+IS}.$$

where CM = Module Complexity, IS = Complexity from coupling

In this normalization, the obtained values for CM are in the range [0,1].

In neural networks a Sigmoid neural layer uses the Sigmoid function to determine its activation. The sigmoid function y(u) is defined as follows:

$y(u) = 1/(1+e^{-u})$; (the usual value of e =2.7183, u= x_values)

One thing about the sigmoid layer is that only positive values are returned between 0.5 and 1. The Sigmoid function is so called because it looks like an **s** as shown in figure 4.1.

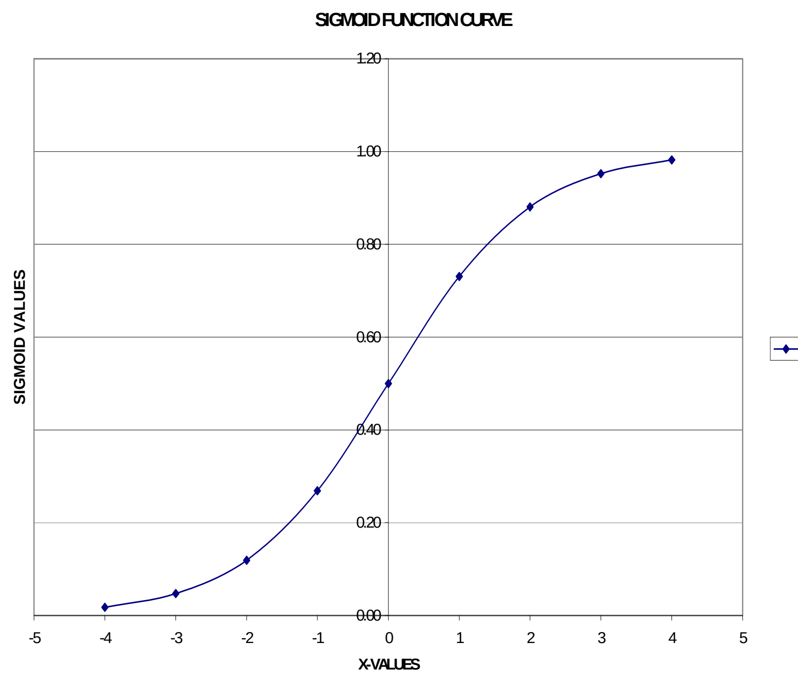

Figure 4.1 Sigmoid function layer

When applied to software measurement, a sigmoid layer for the attribute being measured may be identified only if the expected transformations are in the range [0.5,1].

Although both the approach in [18] and the Sigmoid approach may be used to normalize a measure between the ranges [0,1] and [0.5,1] respectively, in [16] the following important questions were suggested in order to decide whether to use a normalization approach:

(i) Do the measure being normalized measure the same qualitative aspect ?

(ii) Is the normalizing measure really a normalization of the measure being normalized ?

(iii) After normalization, does the normalization agree with the interpretations of the original measure ?

5. Normalization Approaches for LCOM Metric

5.1 Bowless normalized LCOM

Using the approach in [18], a Bowles normalized LCOM (BLCOM) may be defined as follows:

BLCOM= 1-(1/1+LCOM).

The measure BLCOM is a numerical modification of the measure LCOM without changing the empirical meaning of the measure. The result of applying BLCOM on outliered LCOM values obtained from system 3 of this study is shown in table 5.1. Hence 0≤BLCOM≤1 is valid.

Table 5.1. Illustration of BowlessNormalized LCOM (BLCOM)

Class Name	WMC	LCOM	BLCOM	NPM
XlmException	5	2	0.666667	5
HandlerBase	14	91	0.9891304	14
SAXDriver	33	370	0.9973046	31
XmlHandler	13	78	0.9873418	13
XmlParser	118	6075	0.9998354	27
JspXMLParser	2	0	0	1
CompiledException	3	1	0.5	0
JspServlet$Page	11	0	0	0
JspServlet$MapEntry	1	0	0	0
JspMsg	0	0	0	0
JspFactoryImpl$1	2	1	0.5	1

$$0 \leq BLCOM \leq 1$$

Source: [19]

5.2 Sigmoid normalized LCOM (SLCOM)

Using a sigmoid function as explained in section 4.2 (fig 4.1), a sigmoid function layer ,

$y_{(u)} = 1/(1+e^{-u})$

is defined.

Applied to the LCOM metric, a sigmoid normalized LCOM may be defined as follows:

SLCOM= $y_{(LCOM)}= 1/(1+e^{-LCOM})$.

The result of applying SLCOM on outliered LCOM values obtained from system 3 of this study is shown in table 5.2. Hence 0.5≤SLCOM≤1 is valid.

Table 5.2. Illustration of Sigmoid Normalized LCOM (SLCOM)

Class Name	WMC	LCOM	SLCOM	NPM
XlmEx ception	5	2	0.880798	5
Handler Base	14	91	1	14
SAXDriver	33	370	1	31
Xml Handler	13	78	1	13
XmlParser	118	6075	1	27
JspXML Parser	2	0	0.5	1
Compiled Exception	3	1	0.73106	0
JspServle t$Page	11	0	0.5	0
JspServlet$ MapEntry	1	0	0.5	0
JspMsg	0	0	0.5	0
JspFactory Impl$1	2	1	0.73106	1

$$0.5 \leq SLCOM \leq 1$$

Source: [19]

5.3 Bestfit normalized LCOM

The definition of LCOM as presented in section 3 suggests that both the Bowles normalized LCOM and the Sigmoid normalized LCOM may not appropriately represent Chidamber and Kemerer's interpretation of their LCOM metric. Although the former are normalized values, however, in line with Chidamber and Kemerer's definition, a Bestfit normalized LCOM (BFLCOM) may be defined as follows [19]:

BFLCOM=0, LCOM=0

BFLCOM= 1/LCOM, otherwise.

Using the BFLCOM metric, it was possible to eliminate outliers from LCOM values in agreement with the usual definition of LCOM and its interpretation as shown in table 5.3.

Table 5.3. Illustration of Bestfit Normalized LCOM (BFLCOM)

Class Name	WMC	LCOM	BFLCOM	NPM
Xlm Exception	5	2	0.5	5
HandlerBase	14	91	0.0109	14
SAXDriver	33	370	0.0027	31
XmlHandler	13	78	0.128	13
XmlParser	118	6075	0.0001	27
JspXMLParser	2	0	0	1
Compiled Exception	3	1	1	0
JspServlet $Page	11	0	0	0
JspServlet$ MapEntry	1	0	0	0
JspMsg	0	0	0	0
JspFactory Impl$1	2	1	1	1

$$0 \leq BFCOM \leq 1,$$

Source: [19]

5.4 Discussion: Comparison of Normalized LCOM values

A close observation of the considered normalized LCOM metrics reveal some interesting configurations. Bowles LCOM (BLCOM) is a generalized measure. Although its values are in the range [0,1], its transformations with respect to LCOM may not be exact. Consider table 5.1 and transformations of LCOM values from 0 to 0, and from 1 to 0.5. If a modified interpretation of LCOM = [0,1] is accepted [19], then the transformation of LCOM to BLCOM for value 1 should be 1.

The Sigmoid LCOM (SLCOM) also has the same problem when applied to LCOM. SLCOM range [0.5,1] does not necessarily represent LCOM range [0,1]. However, Bestfit LCOM (BFLCOM) range [0,1] gives exact transformation for LCOM range [0,1] table 5.3. This comparative analysis is shown in table 5.4 . From this table BFLCOM seem to be the appropriate normalization of the LCOM metric; considering the appropriate transformations of 0 to 0, and 1 to 1 while eliminating outlier values. This result suggest that the LCOM metric may be used with the BFLCOM metric to obtained normalized values much like the other variants measures of cohesion whose ranges are [0,1] such as Connectivity (CO) metric [9], Tightclass cohesion/Low class cohesion (TCC/LCC) [11], Degree of cohesion in class based on direct relation between its public (D_{CD}) or that based on indirect methods (D_{CI}) [12] , Optimistic class cohesion (OCC) and pessimistic class cohesion (PCC) [13].

Table 5.4. Comparative normalized LCOM metric

Class Name	LCOM	BLCOM	SLCOM	BFLCOM
Xlm Exception	2	0.666667	0.880798	0.5
HandlerBase	91	0.9891304	1	0.0109
SAXDriver	370	0.9973046	1	0.0027
XmlHandler	78	0.9873418	1	0.128
XmlParser	6075	0.9998354	1	0.0001
JspXMLParser	0	0	0.5	0
Compiled Exception	1	0.5	0.7310	1
JspServlet $Page	0	0	0.5	0
JspServlet$ MapEntry	0	0	0.5	0
JspMsg	0	0	0.5	0
JspFactory Impl$1	1	0.5	0.7310	1

References

[1] E. Yourdon and L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design,* Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

[2] E. Okike, "A pedagogical Evaluation and Discusion about the Lack of Cohesion in methods (LCOM) Metric Using field Experiment", *International journal of computer science issues,* Vol. 7, issue 2, No. 3, March 2010. Pp 36-43

[3] M. F. Shumway . "Measuring Class Cohesion in Java". Master of Science Thesis. Department of computer science, Colorado state university, Technical Report CD - 97-113, 1997.

[4] J. M. Bieman and L. M. Ott, . "Measuring Functional Cohesion", IEEE *Transactions on Software Engineering, vol.* 20, no. 8, pp. 644-658, August 1994.

[5] J. M. Bieman and B. K. Kang, 998. Measuring Design – level Cohesion. *IEEE Transactions on Software Engineering,* vol. 20, no. 2, pp. 111-124, February 1998.

[6] S. R. Chidamber and C. F. Kemerer, "Towards a Metric Suite for Object Oriented Design", *Object Oriented Programming Systems, Languages and Applications, Special Issue of SIGPLAN Notices, vol.* 26, no.10, pp. 197-211, October 1991.

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

[8] W. Li and S. Henry, "Object Oriented Metrics that Predict Maintainability". *Journal of Systems and Software, vol.* 23, pp. 111-122, February1993.

[9] M. Hitz and B. Montazeri, "Chidamber and Kenmerer 's mtric Suite: A Measurement Theory Perspective", *IEEE Transactions on Software Engineering, vol.* 22. no. 4, pp.267-270, April 1996.

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

[11] J. M. Bieman and B. K. Kang, "Cohesion and Reuse in an Object Oriented System", *Proceedings of the Symposium on Software Reusability* (SSR '95), Seattle: WA. Pp. 259-262, April 1995.

[12] L. Badri and M. Badri, "A proposal of a New Class cohesion Criterion: An Empirical Study" *Journal of Object Technology, vol.* 3, no. 4, pp. 145-159, April 2004.

[13] H. Aman, K. Yamasski, and M. Noda, "A Proposal of Class Cohesion Metrics using sizes of cohesive parts", *Knowledge based Software Engineering.* T. Welzer et al. Eds. IOS press, pp. 102-107, September 2002.

[14] B. S. Gupta, "A Critique of Cohesion Measures in he Object Oriented Paradigm", M.S Thesis, Department of Computer Science, Michigan Technological University. iii+ 42pp, 1997.

[15] L. C. Briand, J. Daly and J. Wust, "A Unified Framework for Cohesion Measurement in Object Oriented Systems", *Empirical Software Engineering, vol.* 3, no.1, pp. 67-117, 1998.

[16] H. Zuse, "*A framework for Software Measuremen",* New York*:* Walter de Gruyter, 1988.

[17] V. R. Basili, L. C. Briand, and W. Melo, "A validation of Object Oriented Design Metrics as quality indicators", *IEEE Transaction On Software Engineering, vol.* 22 , no.10, pp. 751- 761, 1996.

[18] A. J. Bowles, Effects of Design complexity on Software Maintenance", Dissertation, Northwestern University, Evanston, Illinois, 1983.

[19] E. U. Okike "Measuring class cohesion in object-oriented systems using Chidamber and Kemerar metrics and Java as case study. Ph.D thesis. Department of Computer Science, University of Ibadan, xvii + 133pp, 2007.

Ezekiel U. Okike received the BSc degree in computer science from the University of Ibadan Nigeria in 1992, the Master of Information Science (MInfSc) in 1995 and PhD in computer science in 2007 all from the same University. He has been a lecturer in the Department of Computer Science, University of Ibadan since 1999 to date. Since September, 2008 to date, he has been on leave as a senior lecturer and Dean of the School of Computer Studies, Kampala International University, Uganda. His current research interests are in the areas of software engineering, software metrics, compilers and programming languages. He is a member of IEEE computer and communication societies.