IMPLEMENTATION AND VALIDATION OF OBJECT-ORIENTED DESIGN-LEVEL COHESION METRICS

ABUBAKAR ADAM

COMPUTER SCIENCE

JANUARY 2005

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **ABUBAKAR ADAM** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Thesis Committee

_____
Dr. Jarallah AlGhamdi (Chairman)

_____
Dr. Krishna Rao (Member)

_____
Dr. Moataz Ahmed (Member)

_____
Department Chairman

_____
Dean of Graduate Studies

_____
Date

DEDICATION

This thesis is dedicated to my parent: Late Alhaji Adamu and Hajiya Titi Yusuf.

&

To my wonderful step-mother: Hajiya Habiba Adamu

ACKNOWLEDGEMENT

I would like to begin by thanking Allah, Lord of the worlds, for His infinite mercy, blessing and forgiveness. I thank King Fahd University of Petroleum and Minerals for supporting this research.

My deepest appreciation goes to my thesis advisor, Dr. Jarallah Al-Ghamdi, who has an attitude and the substance of being a good scholar and a good advisor. His continual support and constructive criticism is highly appreciated; I do not know what would have become of this thesis without his support. I am very thankful to the members of my thesis committee: Dr. Krishna Rao and Dr. Moataz Ahmed for their valuable contributions. In particular Dr. Moataz's contributions have been invaluable for this research. My thank goes to the Software Metrics Research Group (SMRG); I found the weekly meetings highly informative.

I would like to thank my friends and colleagues in KFUPM for their companionship especially Rufai, Sohel and Abid for their concern and brotherly support in the course of this research. Immense appreciation goes to Yushau Balarabe, Yau Garba and Hamisu Shehu for their assistance; and to Hassan (Abu Hamza) for helping with some of the statistical analysis.

Finally, I thank the members of my family for their care, love and prayers I believe this is what kept me going.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Thesis Abstract

NAME: Abubakar Adam

TITLE: IMPLEMENTATION AND VALIDATION OF OBJECT-ORIENTED DESIGN-LEVEL COHESION METRICS

MAJOR FIELD: COMPUTER SCIENCE

DATE OF DEGREE: JANUARY 2005

This thesis presents a number of classification criteria that may serve as an evaluation scheme for object-oriented cohesion metrics. Based on these criteria, we present a critical survey of the state-of-the-art of object-oriented cohesion metrics. The thesis also proposes and theoretically validates an object-oriented high-level cohesion metric. Rigorous experiment was conducted in order to tie cohesion to the number of defects in a software system. Results show that cohesion alone does not give enough information regarding the number of defects in a class. The results also show that there are inconsistencies in the definition of the cohesion metrics. The thesis also presents three defects prediction models that are built using regression analysis and Artificial Neural Network (ANN).

خلاصة الرسالة

الاســـم : أبوبكر آدم

عنوان الرسالة : تطبيق صحة مقاييس تماسك الكائنات في مرحلة التصميم.

التخصص : علوم الحاسب الآلي

تاريخ التخرج : يناير 2005

يقدم هذا البحث عدد من معايير التصنيف التي يمكن أن يستفاد منها كوسيلة تقييم لمقاييس مقدار تماسك الكائنات (Object-oriented cohesion). وبناءاً على هذه المعايير، نستعرض دراسة نقدية لأحدث مقاييس تماسك الكائنات. يقترح هذا البحث مقياس جديد لمقدار التماسك وإثبات صحتها نظرياً. أُجريت تجربة عملية دقيقة من أجل ربط مقدار التماسك مع عدد من الأخطاء المنتجة في أنظمة البرمجة. أظهرت النتائج أن مقدار التماسك لوحده قد يعطي معلومات غير كاملة بالنسبة لعدد الأخطاء في الفئة (class). كما أوضحت النتائج عدة تناقضات في تعريف مقاييس التماسك الموجودة. كذلك يستعرض هذا البحث ثلاث نماذج لكشف الأخطاء وهذه النماذج مبنية على تحليل الانحسار التدريجي (Regression Analysis) والشبكة العصبية الاصطناعية (Artificial Neural Network).

درجة الماجستير في العلوم

جامعة الملك فهد لبترول والمعادن

يناير 2005

CHAPTER 1

1 INTRODUCTION

Object-oriented technology is one of the most widely used paradigms for developing software systems. Researchers assert that OO practice assures good quality software. Software quality refers to ease of understandability, maintainability, and reuse. Through the years, many software attributes have been identified that have relation, in one way or the other, with the quality of the artifact being produced. Such attributes include: size, complexity, coupling, and cohesion. Several metrics have been proposed with a view to accurately capture these attributes so that software of high quality is produced. Unfortunately not many of these metrics are exposed to rigorous theoretical and experimental validation in order to determine how effective they are at capturing what they claim to capture. Software metrics simply give a value that represents some software attribute such as coupling and cohesion. These values are difficult to interpret because they do not directly give information that relates to external quality attributes (such as maintainability, reusability, and fault-proneness) which people are more interested in. One way of interpreting the values obtained from such metrics is to associate the values to a quality attribute that is externally visible. In this research, we set out to conduct a rigorous research for cohesion metrics and to tie the values obtained from the existing cohesion metrics to a software attribute that is externally visible. In this chapter, we discuss the concept of object-oriented cohesion and its relationship with coupling in sections 1.1 and 1.2, respectively. In section 1.3 we present a brief introduction to UML. Section 1.4 and 1.5 presents the motivation behind this research and the main contributions of the work, respectively.

1.1 Cohesion

Cohesion is an internal software attribute that depicts how well connected the components of a software module are. This can be determined by knowing the extent to which the individual components of a module are required to perform the same task [37]. In a highly cohesive module all the components performance are tailored towards the requirement of a single function. On the contrary, a low cohesive module has some elements that have little relationships with others, which is an indication that the module may provide several unrelated functions [26]. If a module is highly cohesive then it is easy to develop and maintain because it does not have much dependence on the components of other modules as such it is less error-prone.

Measures such as coupling equally serve as quality indicators; coupling and cohesion are terms used to define module interconnectedness. Coupling is a measure of how strongly one module is connected to, have knowledge of, or relies on other modules [60].While cohesion addresses intra-module connectedness, coupling addresses inter-module connectedness. In general, coupling should be minimized while cohesion should be maximized [33][60]. In object-oriented paradigm, however, coupling should not be completely minimized because some level of dependence is required for instance dependence due to inheritance is required. This concept is explained in the following section.

1.2 Relationship between coupling and cohesion

One of the most difficult tasks to achieve in object-oriented design is to come up with well designed classes; classes that are easy to understand, easy to maintain and easy to reuse. Two important factors that affect the design of classes are coupling and

cohesion. Coupling and cohesion are highly related. Bad cohesion usually leads to bad coupling because they have a highly interdependent influence [60].

1.2.1 Low Coupling

Coupling is a measure of how strongly one module is connected to, have knowledge of, or relies on other modules. A class with low coupling is not dependent on too many other classes. On the other hand, a class with high coupling (or strong) coupling relies on many other classes. Such classes may be undesirable due to the following reasons [60]:

- Changes in related classes force local changes.
- Harder to understand in isolation
- Harder to reuse because its use requires the additional presence of the classes on which it is dependent.

Hence, low coupling is a principle to keep in mind during all design decisions; it is an underlying goal to continually consider. Low coupling encourages assigning a responsibility so that its placement does not increase the coupling to such a level that it leads to the negative results that high coupling can produce. Low coupling supports the design of classes that are more independent, which reduces the impact of change. The extreme case of low coupling is not desirable i.e. when there is no coupling between classes at all or when it is extremely low. If low coupling is taken to excess, it yields a poor design because it leads to a few not cohesive, bloated, and complex active objects that do all the work [60].

1.2.2 High Cohesion

Cohesion, in object-oriented terms, is a measure of how strongly related and focused the responsibilities of a module are. The issue to consider here is how to keep

complexity manageable. A class with low cohesion does many unrelated things, or does too much work. Such classes are undesirable; they suffer from the following problems:

- Hard to comprehend
- Hard to reuse
- Hard to maintain
- Delicate; constantly effected by change

In general, the relationship between coupling and cohesion is that coupling should be low while cohesion is kept high.

1.3 UML

In this section we introduce the Unified Modeling Language (UML), which is a language used for modeling the design of software products. UML offers a lot of advantages to software developers; it makes communication across development team simple; developers can come up with models that are language independent and that are easy to understand and interpret.

"The Unified Modeling Language (UML) is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system." [13]. The building blocks of UML include the following [13]:

1. Things
2. Relationships
3. Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

1.3.1 Things

There are four kinds of things in the UML:

- Structural things
- Behavioral things
- Grouping things
- Annotational things

1.3.2 Relationships

There are four kinds of relationships in the UML:

- Dependency
- Association
- Generalization
- Realization

1.3.3 Diagrams

UML diagrams are simply projections into system; they are used to visualize systems from different perspectives[13]. The UML includes nine diagrams, these are [13][70]:

Class diagram: shows the classes of the system, their interrelationships (such as aggregation, and association), and the operations and attributes of the classes. These diagrams are the most common diagrams found in modeling object-oriented systems. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modeling and detailed design modeling.

Object diagram (sometimes referred to as instance diagrams): shows a set of objects and their relationships. Object diagrams are useful for exploring "real world" examples of objects and the relationships between them. Although UML class diagrams are very good at describing this very information some people find them too abstract – a UML object diagram can be a good option for explaining complex relationships between classes.

Use case diagram: shows a set of use cases and actors and their relationships. UML Use Case Diagrams can be used to describe the functionality of a system in a horizontal way. That is, rather than merely representing the details of individual features of your system, UCDs can be used to show all of its available functionality. It is important to note, though, that UCDs are fundamentally different from sequence diagrams or flow charts because they do not make any attempt to represent the order or number of times that the systems actions and sub-actions should be executed.

UCDs have 4 major elements: The *actors* that the system you are describing interacts with, the *system* itself, the *use cases*, or services, that the system knows how to perform, and the lines that represent *relationships* between these elements

Sequence diagram: this is a kind of interaction diagram that emphasizes the time ordering of messages. The UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes.

Collaboration diagram: this also is an interaction diagram but its emphasis is on the structural organization of the objects that send and receive messages. UML

Collaboration diagrams (interaction diagrams) illustrate the relationship and interaction between software objects

Statechart diagram: shows a state machine, consisting of states, transitions, events, and activities. Statechart diagrams address the dynamic view of a system. A statechart diagram is a view of a *state machine* that models the changing behavior of a state. Statechart diagrams show the various states that an object goes through, as well as the events that cause a transition from one state to another.

Activity diagram: is a special kind of a statechart diagram that shows the flow from activity to activity within a system. Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects. Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

Component diagram: shows the organizations and dependencies among a set of components. Component diagrams address the static implementation view of a system. They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations. Today in software engineering we have team-based development efforts, where everyone has to work on different component. That's important to have a component diagram in modeling process of the system. A component diagram describes the organization of the physical components in a system.

Deployment diagram: shows the configuration of run-time processing nodes and the components that live on them. They are related to component diagrams in that a node typically encloses one or more components. A UML deployment diagram depicts a static view of the run-time configuration of processing nodes and the components that run on those nodes. In other words, deployment diagrams show the hardware for your system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another

1.4 Motivation

Software engineering researchers have attached importance to having high cohesion in the modules of software products, as briefly discussed above. They have asserted that highly cohesive program components are desirable because they lead to better external attributes such as reusability, comprehensibility, maintainability etc. According to Fenton in [37], designs that possess high module cohesion and low module coupling are assumed to lead to more reliable and maintainable code. In order to be certain about these claims, we need to have good understanding of the cohesion of software systems and for this to be achieved; we need to have effective means of measuring it so that it can easily be tied to software quality attribute. Our motivation in this research is to determine the predictive power of the existing cohesion metrics with respect to an external quality attribute that is easily be understood.

1.5 Main Contributions

The main contributions of this thesis work are:

- Conducting a critical survey of existing cohesion metrics.
- Developing a classification for the existing cohesion metrics
- Proposing a class cohesion metric and theoretically validating it

- Implementing the proposed metric as well as the existing cohesion metrics.
- Investigating whether cohesion metrics can be predictors of fault density.
- Building defect prediction models using ANN and regression analysis and evaluating the performance the models.

1.6 Organization of the Thesis

The rest of this thesis is organized as follows: Chapter 2 presents the literature survey of the existing cohesion metrics. Chapter 3 presents a critical analysis of the cohesion metrics using some attributes that the cohesion metrics have in common. In chapter 4 we discuss the new proposed metric, theoretically validate it and empirically show that there are inconsistencies in the definitions of the existing cohesion metrics. In chapter 5, we present defect prediction models built using regression analysis and Artificial Neural Network; the performance of the models are compared. Finally we conclude in chapter 6.

CHAPTER 2

2 LITERATURE REVIEW

2.1 Introduction

In this chapter we discuss the existing cohesion metrics found in literature that are proposed by software researches. While section 2.2 covers cohesion metrics in procedural programs, section 2.3 discusses cohesion metrics in object oriented paradigm.

2.2 Background

By definition, cohesion captures the degree of interdependence among elements of the same module [73]. As explained in Chapter 1, modules with strong cohesion are easier to maintain and they greatly improve the possibility of reuse. The definition of cohesion can have two interpretations; a module is said to be cohesive if (i) its elements are tailored towards one functionality and (ii) the module is self-contained; i.e. it does not rely on other modules for its function to be achieved. The programming paradigm in question determines what a module is and what an element is. In procedural paradigm, elements of module are statements, and sub functions. In object-oriented paradigm, the counterparts of module are classes and methods. The elements of a method are statements and attributes since they are accessed either directly or via access functions in the methods. The elements of an object class are methods and instance variables [33]. In the following sections we discussed some of the proposed metrics in procedural paradigm and those in object-oriented paradigm found in the literature.

2.3 Cohesion Metrics in procedural programs

A procedural program is composed of one or more units or modules and each module is composed of one or more procedures (e.g of procedural programs include programs written in C and FORTRAN).

2.3.1 The SFC and WFC Metrics

Bieman and Ott [12], proposed three cohesion metrics. These measures are: Strong Functional Cohesion (SFC(p)), Weak Functional Cohesion (WFC(p)) and Adhesiveness of a procedure (A(p)).

Strong Functional Cohesion (SFC) is defined as the ratio of super-glue tokens to the total number of data tokens in a procedure p. The SFC is a measure of the minimal functional cohesion in a procedure; the metric is given by the following formula.

$$\mathrm{SFC(p)} = \frac{|\ Supergluetokens\ |}{|\ tokens\ |}$$

The weak functional cohesion (WFC) is defined as the ratio of glue tokens to the total number of tokens in a procedure p.

$$\mathrm{WFC(p)} = \frac{|\ gluetokens\ |}{|\ tokens\ |}$$

The third measure they proposed is *Adhesiveness*, this is related to the number of slices that each token "glues" together. The *Adhesiveness* of a procedure p is defined as follows:

$$A(p) = \frac{\sum_{tokens} \frac{|\ programslicescontainingagluetoken\ |}{|\ programslices\ |}}{|\ tokens\ | * |\ programslices\ |}$$

A *program slice* is a set of program statements which include references to a particular program variable. A *glue token* is a token which is used in more than one program slice that includes a certain statement. A *super glue token* unites all the program slices at some statements. The measures capture the number of program slices having glue or super glue tokens as a proportion of total program slices. Note that a procedure having no cohesion would have no glue tokens. However, a procedure having perfect cohesion would have super glue tokens at every statement.

2.3.2 The DLC and DFC Metrics

Bieman and Kang proposed two design level cohesion metrics [10]: DLC (Design Level Cohesion) and DFC (Design Functional Cohesion).

An ordinal scale of cohesion measures is defined: Coincidental, Conditional, Iterative, Communicative, Sequential, and Functional. Each pair of output tokens in a module is evaluated for the strongest cohesion the pair exhibits. The minimum of such value over all output token pairs gives the *Design Level Cohesion* (DLC). *Design Functional Cohesion* (DFC), on the other hand, is a slice based measure which averages adhesiveness of output token slices corresponding with the interface pints of the module.

2.4 Cohesion Metrics in Object Oriented Programs

In this section we describe the cohesion metrics that were proposed to measure cohesion in object oriented programs.

2.4.1 The Degree of Method and Class Cohesion of Eder et al.

Eder et al. [33] have extended the concept of coupling and cohesion developed oringinally for procedural-oriented systems to object-oriented sytems. They distinguised between three types of cohesion in an object-oriented systems: method, class and inheritance cohesion. For each type, various degrees of cohesion are defined. In this section we succinctly explain the degrees of each type of cohesion.

Method Cohesion: the elements of a method are statements, local variables and attributes of the method's class. Eder et al. defined seven degrees of method cohesion as given below from weakest to strongest [33]:

- *Concidental:* the elements of a method have nothing in common besides being within the same method.
- *Logical:* elements with similar functionality such as input/output handling are collected in one method
- *Temporal:* the elements of a method have logical cohesion and are performed at the same time.
- *Procedural:* the elements of a method are connected by some control flow.
- *Communicational:* the elements of a method are connected by some control flow and operate on the same set of data.
- *Sequential:* the elements of a method have communicational cohesion and are connected by a sequential control flow.

- *Functional:* the elements of a method have sequential cohesion, and all elements contribute to a single task in the problem domain. Functional cohesion fully supports the principle of locality and thus minimizes maintenance efforts.

Class Cohesion: class cohesion addresses the relationships between the elements of a class. The elements of a class are its non-inherited methods and non-inherited attributes. The following are the five degrees of class cohesion from weakest to strongest:

- *Seperable:* the objects of a class represent multiple unrelated data abstractions
- *Multifaceted:* the objects of a class represent multiple related data abstractions. The relation is caused by at least one method of the class which uses all these data abstractions.
- *Non-delegated:* there exist attributes which do not describe the whole data abstraction represented by a class, but only a component of it.
- *Concealed:* there exist some useful data abstraction concealed in the data abstraction represented by the class. Consequently, the class includes some attributes and methods which might make another class.
- *Model:* the class represents a single, semantically meaningful concept.

Inheritance: This is similar to class cohesion, but it is a bit different in that it considers all methods and attributes in a class including those that are inherited.

2.4.2 The LCOM1 and LCOM2 Metrics

Chidamber and Kemerer [28] use the notion of degree of similarity of methods to propose a cohesion metric, Lack of Cohesion Measure (LCOM). The definition of this metric is given below.

Definition 2.1:

Consider a class C with *n* methods M_1, M_2,….., M_n. Let $\{I_i\}$ = set of instance variables used by method M_i. There are *n* such sets, i.e., $\{I_1\}$, $\{I_2\}$,….., $\{I_n\}$. LCOM1(C) = the number of disjoint sets formed by the intersection of *n* sets. In other words, LCOM1 is the number of pairs of methods with no common attributes references.

Example 2.1: Let m1, m2, m3 and A1, A2, A3, A4 in Figure 1 represent the methods and attributes in class C.

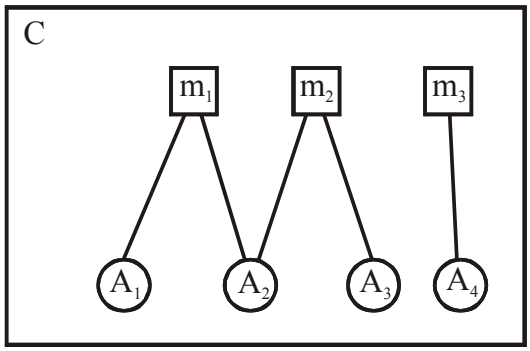


Figure 1: A class C with three methods and four attributes

From the example given in Figure 1, LCOM1 = 2. Note that LCOM1 is an inverse cohesion measure. A high value of LCOM1 indicates low cohesion and vice versa.

In [29], Chidamber and Kemerer have given the following new definition for *LCOM*. Let the new LCOM be LCOM2.

Definition 2.2:

Consider a class C with methods M_1, M_2,….., M_n. Let $\{I_i\}$ = set of instance variables used by method M_i. There are n such sets, i.e., $\{1_i\}$, $\{I_2\}$,….., $\{I_n\}$. Let $P = \{ (I_i, I_j) \mid I_i \cap I_j = \varnothing\}$ and $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varnothing\}$. If all n sets $\{1_i\}$, $\{I_2\}$,….., $\{I_n\}$ are $\varnothing$ then let $P = \varnothing$.

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

In other words, P is the number of pairs of methods without shared attributess and Q is the pairs of methods with shared attributes.

Using the example given in Figure 1, we have, P = 2 and Q = 1 thus LCOM2 = 1.

2.4.3 The LCOM3, LCOM4 and Co Metrics

Hitz and Montazeri evaluated the metrics suit for object-oriented design put forward by Chidamber and Kemerer in [29] by applying the principle of measurement theory. One of the metrics evaluated is Lack of Cohesion in Methods (LCOM). They proposed alternative definitions for the LCOM metric [49][50], as presented in the following definitions.

Definition 2.3:

Let X denote a class, I_x the set of its attributes, and M_x the set of its methods. Consider a simple undirected graph $G_x(V, E)$ with $V = M_x$ and $E = \{(m, n) \in V \times V \mid \exists I \in I_x: (m \text{ accesses } i) \wedge (n \text{ accesses } i)\}$.

LCOM3(C) = Number of connected components of G_x.

This definition is illustrated in Figure 2 using the class *C* given in Figure 1. From the figure we can see that the graph G_x has two connected components. Thus LCOM3 = 2.

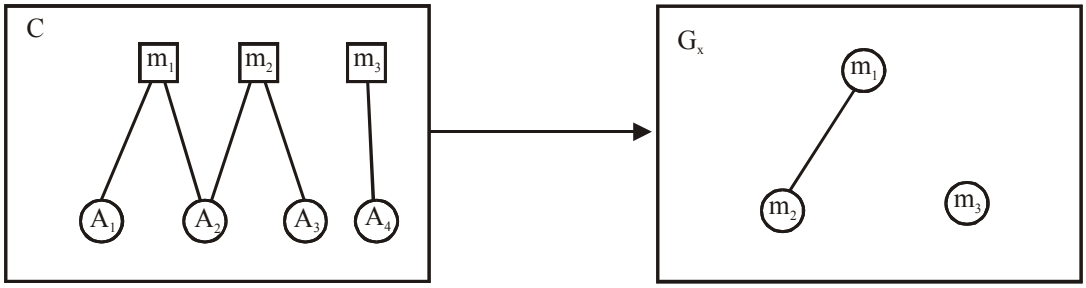


Figure 2: A class C and G_x

Hitz and Montazeri identified a problem with the *access methods* for LCOM3. An access method provides read or write access to an attribute of the class. Access methods typically reference only one attribute, namely the one they provide access to. If other methods of the class use the access methods, they may no longer need to directly reference any attribute at all. These methods are then isolated vertices in graph G_x. Thus, the presence of access methods artificially decreases the class cohesion as measured by LCOM3. To remedy this problem, Hitz and Montazeri proposed a second version of their LCOM measure. In this version, the definition of G_x is changed as follows: there is also an edge between vertices representing methods m_1 and m_2, if m_1 invokes m_2 or vice versa.

Definition 2.4:

Let X denote a class, I_x the set of its attributes, and M_x the set of its methods. Consider a simple undirected graph $G_x(V, E)$ with $V = M_x$ and $E = \{(m, n) \in V \times V \mid (\exists I \in I_x: (m \text{ accesses } i) \wedge (n \text{ accesses } i)) \vee (m \text{ invokes } n) \vee (n \text{ invokes } m)\}$.

$LCOM4$(C) = Number of connected components of G_x.

See Figure 3 for the illustration of this definition. From the figure we can see that the graph G_x has only one connected component. Thus, LCOM4 = 1.

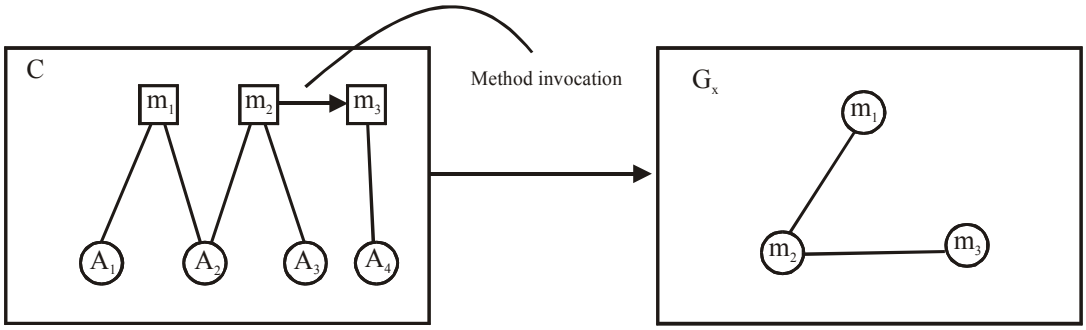


Figure 3: A class C and G_x with interaction among methods

In the case where G_x consists of only one connected component, i.e., LCOM = 1, the number of edges $|E|$ ranges between $|V| - 1$ (minimum cohesion) and $|V|.(|V| - 1)/2$ (maximum cohesion). Hitz and Montazeri defined a measure C ("connectivity") [50] which further discriminates classes having LCOM4 = 1 by taking into account the number of edges of the connected component.

Hitz and Montazeri defined *C* (Let it be Co in order to differentiate the measure from C used for classes in our examples) as follows:

$$\text{Co(c)} = 2 \cdot \frac{|E_c| - (|V_c| - 1)}{(|V_c| - 1).(|V_c| - 2)}$$

Where E_c and V_c are the edges and vertices of the connection graph of the class *c*.

From the example given in Figure 3, we have $E_c = 2$ and $V_c = 3$. Hence, Co(C) = 0

2.4.4 The TCC and LCC Metrics

The approach by Bieman and Kang [11] is also based on that of Chidamber and Kemerer's. They consider pairs of methods that use common attributes. They have defined two different cohesion measures based on the direct and indirect connectivity between pairs of methods. Two methods that use one or more common attributes are said to be *directly connected*. Whereas two methods that are connected through other directly connected methods are called *indirectly connected*. The indirect connection relation is the transitive closure of the direct connection relation. Thus, a method M_1 is indirectly connected with a method M_n if there is a sequence of methods M_2, M_3, … M_{n-1} such that $M_1 \, \delta \, M_2$, … $M_{n-1} \, \delta \, M_n$. Where $M_i \, \delta \, M_j$ represents a direct connection.

Let *NDC*(*C*) be the number of pairs of directly connected methods of a class *C*, *NIC*(*C*) be the number of pairs of indirectly connected methods of *C* and *NP*(*C*) be the maximum possible number of connections in *C*. It is clear that for a class with *N* methods, $NP(C) = N\,(N-1) / 2$.

Tight Class Cohesion (*TCC*) is defined to be a ratio of the number of pairs of directly connected methods in a class, *NDC*(*C*), to the maximum possible number of connections in a class, *NP*(*C*).

$$TCC(C) = \frac{NDC(C)}{NP(C)}$$

Loose Class Cohesion (*LCC*) is defined to be a ratio of the sum of the number of pairs of directly connected methods, *NDC*(*C*), and number of pairs of indirectly connected methods, *NIC*(*C*), in a class *C* to the maximum possible number of connections in *C*, *NP*(*C*).

$$LCC(C) = \frac{NDC(C) + NIC(C)}{NP(C)}$$

With respect to inheritance, Bieman and Kang have stated three options for the analysis of cohesion of a class [11]:

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

Bieman and Kang identified a problem with constructor methods for *TCC* and *LCC*. A class constructor is an initialization function. It generally accesses all attributes in the class, and thus, shares attributes with virtually all other methods. Constructors create connections between methods even if the methods do not have any other relationships.

Therefore, the presence of a constructor method artificially increases cohesion as measured by *TCC* and *LCC*. Bieman and Kang have therefore recommended excluding constructors (and also destructors) from the analysis of cohesion [11].

To illustrate these two metrics, consider the class given in Figure 4. From the figure, we have:

NP(C) = 3, NDC(C) = 2 and NIC(C) = 1, thus TCC = 2/3 and LCC = 1.

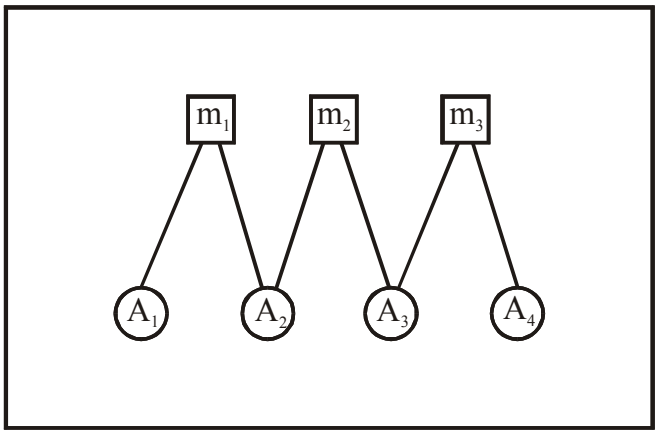


Figure 4: A class C with three methods and four attributes

2.4.5 The LCOM5 Metric

Henderson-Sellers et al.[47] also based their work on the metric suite of Chidamber and Kemerer [29]. The suite is evaluated from a mathematical point of view and a new formulation for the LCOM metric was defined. Their definition is based on the following properties:

- The measure yields 0, if each method of the class references every attribute of the class (this situation is called “perfect cohesion” by Henderson-Sellers”).

- The measure yields 1, if each method of the class references only a single attribute.
- Values between 0 and 1 are to be interpreted as percentages of the perfect value.

We call their definition LCOM5 and it is defined as follows:

Definition 2.5:

Consider a set of methods $\{M_i\}$ $(i = 1,\ldots.., m)$ of a class C accessing a set of attributes $\{A_j\}$ $(j = 1,\ldots.., a)$. Let the number of methods which access an attribute A_j be $\mu(A_j)$ and total number of attributes in $\{A_j\}$ is a.

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

This definition is illustrated in Figure 5.

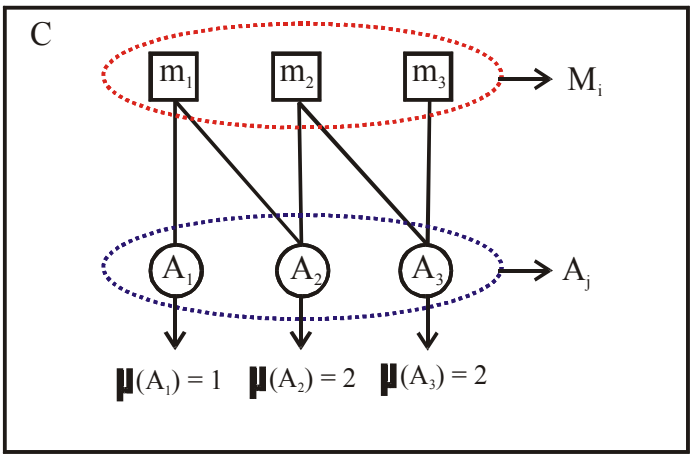


Figure 5: A class C with three methods and three attributes

From the example, we have: $m = 3$ and $a = 3$, Therefore

$$\text{LCOM5} = \frac{\frac{1}{3}(1+2+2)-3}{1-3}$$

thus LCOM5 = 2/3.

2.4.6 The RCI Metric

Briand et al. proposed a cohesion measure in [15] that is based on the visualization of a class as a collection of *data declarations* and *methods*. Data declarations are (i) local type declarations, (ii) the class itself (as an implicit public type), and (iii) public/private attributes (including constants). Briand et al. defined two types of interactions, *DD-interactions* (declaration-declaration interactions) and *DM-Interactions* (declaration-method interactions).

DD-interaction: A data declaration *a* DD-interacts with another data declaration *b*, if a change in *a*'s declaration or use may cause the need for a change in *b*'s declaration or use. We say that there is a *DD-interaction* between *a* and *b*. The following are examples of *DD-interactions*:

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

DD-interactions need not be confined to one class. There can be DD-interactions between attributes and types of different classes. The DD-interaction relationship is

transitive. If *a* DD-interacts with *b* and *b* DD-interacts with *c*, then *a* DD-interacts with *c*.

DM-interaction: Data declarations can also interact with methods. There is a *DM-interaction* between a data declaration *a* and method *m* either

- if *a* DD-interacts with at least one data declaration of *m* (Data declarations of methods include their parameters, return type and local variables), or
- if *a* is an attribute and *m* uses/accesses it.

Briand et al. defined *CI*(*C*) (*CI* means Cohesive Interactions) to be the set of all DD- and DM-interactions present in the class *C* and *Max*(*C*) to be the set of all possible DD- and DM-interactions that can be established in class *C*. *RCI* can be defined as follows:

$$RCI(C) = \frac{|\,CI(C)\,|}{|\,Max(C)\,|}$$

Consider a class with four methods and five attributes as shown in Figure 6, from the figure we have, |CI(C)| = 8 and |Max(C)| = 20. Hence RCI = 8/20 = 2/5.

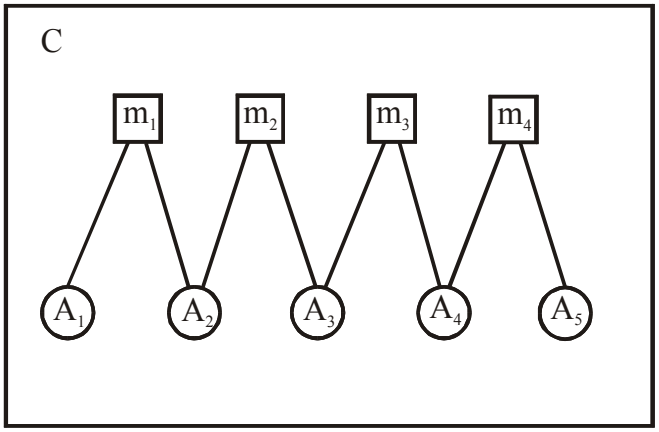


Figure 6: A class C with four methods and five attributes

2.4.7 The CAMC Metric

In 1999, Bansiya et al. [6] proposed a design metric to evaluate cohesion among methods of a class early in the analysis and the design phase. The metric evaluates the consistency of methods in a class' interface using the parameter lists of the methods. The metric can be applied on class declarations that only contain method prototypes (method types and parameter types). They call their metric CAMC (Cohesion Among Methods of Classes).

The CAMC metric is based on the assumption that the parameters of a method reasonably define the types of interaction that methods may implement. To compute the CAMC metric value, an overall union (T) of all object types in the parameters of the methods of a class is determined. A set M_i of parameter object types for each method is also determined. An intersection (set P_i) of M_i with the union set T is computed for all methods in the class. A ratio of the size of the intersection (P_i) set to the size of the union set (T) is computed for all methods. The summation of all intersection sets P_i is divided by product of the number of methods and the size of the

union set T, to give a value for the CAMC metric. Mathematically, the metric is defined as follows:

$$CAMC = \frac{\sum_{i=1}^{n} |P_i|}{|T| \times n}$$

Where

N is number of methods in the class

M_i is the set of parameters of method *i*

T is the union of M_i, for every *i* = 1 to n

P_i is the intersection of set M_i with T i.e. $P_i = Mi \cap T$

The metric value ranges between 0 and 1.0. A value of 1.0 represents maximum cohesion and 0 represents a completely un-cohesive class.

2.4.8 The CBMC Metric

In 2000, Chae et al. highlighted two problems with the existing cohesion metrics.They noted that the existing cohesion measures do not [26]:

1. take into account the properties of special methods like access methods, constructors etc. thus fail to properly reflect the actual cohesiveness of classes.
2. consider the patterns of the interactions among members, they are simply based on counting the number of the attributes refernced by methods or the number of method pairs with shared attributes.

In order to cope with these problems, they proposed a new metric called CBMC (Cohesion Based on Member Connectivity) whose definition is given below. Their metric is based on two things: *connectivity factor* and *structure factor*.

Definition 2.6:

The **CBMC** for a class C, CBMC(C), is defined to be the connectivity factor of its reference graph, $F_c(G_r(C))$, *scaled by the structure factor of its reference graph*, $F_c(G_r(C))$

$$CBMC(C) = F_c(G_r(C)) \times F_s(G_r(C)) = F_c(G_r(C)) \times \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i(C))$$

Where $F_c(G_r(C)) = \frac{|M_g(G_r)|}{|M_n(G_r)|}$ is the connectivity factor(represents the degree of the connectivity among the members).

and $F_s(G_r(C)) = \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i)$ is the structure factor

M_g and M_n are the set of glue methods and normal methods respectively. Glue methods are the minimum number of methods without which the reference graph will be divided into sub-graphs. G_r^i is one of the *n* children of G_r in the *structure tree*; CBMC denotes the cohesion of a component G_r^i.

Example 2.2:

Consider the class shown in Figure 7,

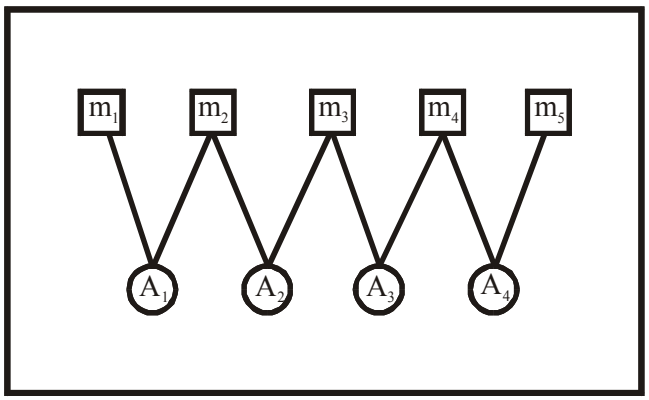


Figure 7: A class C with five methods and four attributes

To compute the CBMC of the class in Figure 7, we need to construct its structure tree first, which is shown in Figure 8.

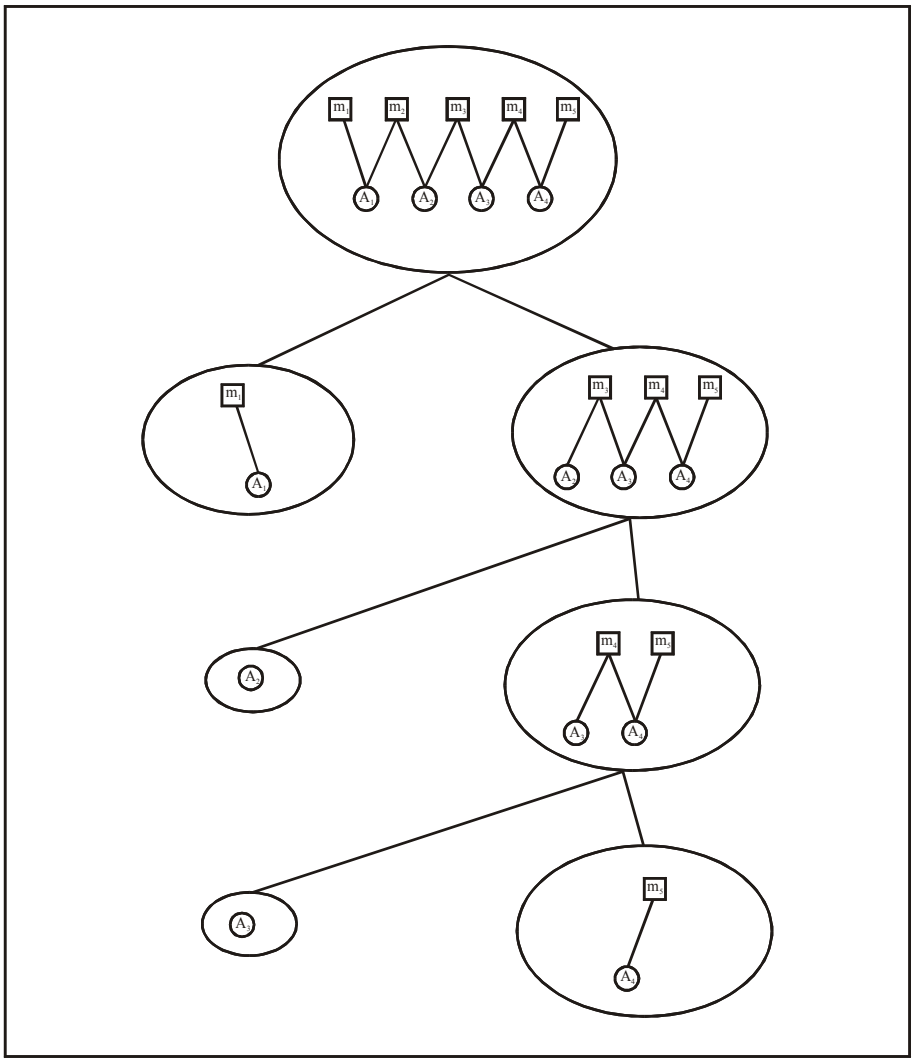


Figure 8: The structure tree of the class in Figure 7

$$CBMC(C) = F_c(G_r(C)) \times F_s(G_r(C)) = F_c(G_r(C)) \times \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i(C))$$

$$CBMC(C) = \frac{1}{5} \times \frac{1}{2}\left[CBMC(G^1) + CBMC(G^2)\right]$$

$$CBMC(C) = \frac{1}{10}\left[1 + F_C \times F_S\right] = \frac{1}{10}\left[1 + \frac{1}{3} \times \frac{1}{2}\left(CBMC(G^{21}) + CBMC(G^{22})\right)\right]$$

$$CBMC(C) = \frac{1}{10}\left[1 + \frac{1}{6}\left(1 + F_C \times F_S\right)\right] = \frac{1}{10}\left[1 + \frac{1}{6}\left(1 + \frac{1}{2} \times \frac{1}{2}\left\{CBMC(G^{221}) + CBMC(G^{222})\right\}\right)\right]$$

$$CBMC(C) = \frac{1}{10}\left[1 + \frac{1}{6}\left(1 + \frac{1}{4}\left\{1 + F_C \times F_S\right\}\right)\right] = \frac{1}{10}\left[1 + \frac{1}{6}\left(1 + \frac{1}{4}\left\{1 + 1 \times 1\right\}\right)\right] = \frac{1}{8}$$

2.4.9 The CCM and ECCM Metrics

Jarallah et al. in [52] proposed two cohsion metrics for assessig the extent to which an inheritance hierarchy follows some four design principles they discussed in their paper. The proposed metrcs are: CCM (Class Connection Metric) and ECCM (Enhanced Class Connection Metric), the difinition of these metrics are given below.

$$CCM(C) = \frac{NC(C)}{NMP(C) \cdot NCC(C)}$$

Where NC(C) is the number of actual connection among the methods of the class, NMP(C) is the number of the maximum possible connections among the methods of the class C and NCC(C) is the number of connected components of the connection graph G_c.

$$ECCM(C) = \frac{NC(C)}{NMP(C).NCC(C)} \times (1 - PenaltyFactor(C))$$

or simply,

$$ECCM(C) = CCM(C) \times (1 - PenaltyFactor(C))$$

Where $PenaltyFactor(C) = \frac{NORM(C)}{NOIM(C)}$

NORM(C) is the number of re-implemented methods and NOIM(C) is the number of inherited methods.

NB:

The connection criterion of CCM and ECCM is slightly different from that of TCC and LCC. For CCM and ECCM, two methods *A* and *B* are connected in the connection graph G_C if they satisfy any or both of the following conditions:

- Methods *A* and *B* access one or more attributes in common.
- Methods *A* and *B* invoke one or more methods in common.

Example 2.3:

Considering the class given in Figure 7, we have the following connected graph.

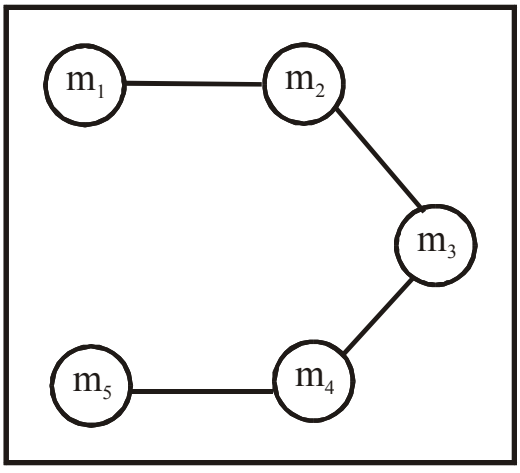


Figure 9: The connected graph of the class in Figure 7

Thus, NC(C) = 5, NMP(C) = 10 and NCC(C) = 1 hence CCM = 2/5

Here the value of ECCM = CCM because in this example no specification is given for the inherited methods.

2.4.10 The OCC and PCC

Aman et al. [4] proposed two cohesion metrics that not only consider the connections among the component of a class but also consider the sizes of connected modules as well as the strength of method connection,. These metrics are: OCC (Optimistic Class Cohesion) and PCC (Pessimistic Class Cohesion).

Definition 2.7: Weak-connection graph

Given a class, let M be the set of methods, and A be the set of attributes, within the class. A weak-connection graph is defined as an undirected graph $G_w(V, E)$, where V = M and

$$E = \{\{u, v\} \in M \times M \mid \exists a \in A \text{ s.t. } (ac(u, a) \wedge ac(v, a))\}$$

Definition 2.8: Strong-connection graph

Given a class, let M be the set of methods, and A be the set of attributes, within the class. Strong-connection graph is defined as a directed graph G_s (V, E), where V = M and

$$E = \{\{u, v\} \in M \times M \mid \exists a \in A s.t. (wr(u, a) \wedge re(v, a))\}$$

Definition 2.9: Optimistic Class Cohesion (OCC)

Given a class, let M be the set of methods, and A be the set of attributes, within C. Consider the weak-connection graph $G_w(V, E)$, where V = M and E is as given in

equation 1. Let n = |M|. For each method $m_i \in$ M (i = 1, . . . , n), let $R_w(m_i)$ be the set of methods which are reachable by m_i on $G_w(V, E)$:

$$R_w(mi)=\{m_j \in M \mid \exists m_{k1},...,m_{kp} \in M \; s.t. \{m_{ks}, m_{ks+1}\} \in E(s=1,..., p-1),$$
$$m_i = m_{k1}, m_j = m_{kp}, i \neq j$$

The Optimistic Class Cohesion (OCC) for a class C is defined as follows:

$$OCC(C) = \begin{cases} \max\limits_{i=1,...,n}\left[\dfrac{|Rw(mi)|}{n-1}\right], (n>1) \\ 0, (n=1) \end{cases}$$

Definition 2.10: Pessimistic Class Cohesion (PCC)

Given a class C, let M be the set of methods, and A be the set of attributes, within C. Consider the strong-connection graph $G_s(V, E)$, where V = M and E is as in equation 2. Let n = |M|. For each method $m_i \in$ M (i = 1, . . . , n), let $R_s(m_i)$ be the set of methods which are reachable by m_i on $G_s(V, E)$:

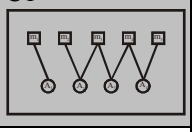
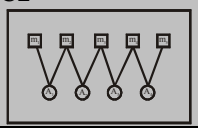
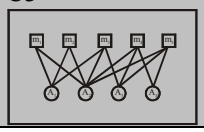
$$R_s(mi)=\{m_j \in M \mid \exists m_{k1},...,m_{kp} \in M \; s.t. \{m_{ks}, m_{ks+1}\} \in E(s=1,..., p-1),$$
$$m_i = m_{k1}, m_j = m_{kp}, i \neq j$$

The Pessimistic Class Cohesion (PCC) for a class C is defined as follows:

$$PCC(C) = \begin{cases} \max\limits_{i=1,...,n}\left[\dfrac{|Rs(mi)|}{n-1}\right], (n>1) \\ 0, (n=1) \end{cases}$$

Table 1 shows three different classes each with five methods and four attributes but with different level of interactions between the components of the class. For each class, the measure for each metric is computed.

Table 1: Cohesion Metrics Examples

#	Metric	C1 	C2 	C3 	Comment
1	LCOM1	6	6	0	Does not differentiate between C1 and C2
2	LCOM2	3	2	0	OK
3	LCOM3	2	1	1	Does not differentiate between C2 and C3
4	LCOM4	2	1	1	Does not differentiate between C2 and C3
5	Co	N/A	0	1	Applicable only when the connected component is one.
6	LCOM5	0.81	0.75	0.44	OK
7	LCC	0.6	1	1	Does not differentiate between C2 and C3
8	TCC	0.3	0.4	1	OK
9	RCI	0.23	0.27	0.43	OK
10	CCM	0.15	0.4	1	OK
11	ECCM	N/A	N/A	N/A	Cannot be computed using these examples
12	CAMC				
13	CBMC	0	0.13	0.6	OK
14	OCC	0.75	1	1	Does not differentiate between C2 and C3
15	PCC	N/A	N/A	N/A	Cannot be computed using these examples

NB:

In Table 1, 'OK' in the comment column signifies that that the metric behaves the way we expect in the above examples. However, even those metrics that appear to follow intuition in the above examples have their own peculiar problems as discussed in chapter 3. For instance LCOM2 may return zero values for classes where the classes have different cohesion values. LCOM5 will give an infinite value if there is only one method in the class. Table 2 gives a brief summary of the existing cohesion metrics.

Table 2: Overview of Cohesion Measure

Metric	Definition	Validation	Cohesion Criteria	Source
LCOM1	The number of pairs of methods that share no attributes.	Validated theoretically	Attribute sharing	[28]
LCOM2	Let P be the pairs of methods without shared attributes, and Q be the pairs of methods with shared attributes. Then $\text{LCOM2} = \begin{cases} \lvert P \rvert - \lvert Q \rvert, if \ \lvert P \rvert > \lvert Q \rvert \\ 0, otherwise \end{cases}$	Validated theoretically and empirically	Attribute sharing	[29]
LCOM3	Consider an undirected graph G where the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods share at least one attribute. LCOM3=|connected components of G|	Not validated	Attribute sharing	[49]
LCOM4	Like LCOM3, where graph G additionally has an edge between vertices representing methods M_i and M_j, if M_i invokes M_j or vice versa.	Not validated	Attribute sharing and methods invocation	[49]
Co	$Co(C) = 2.\frac{\lvert Ec \rvert - (\lvert Vc \rvert - 1)}{(\lvert Vc \rvert - 1).(\lvert Vc \rvert - 2)}$ Where E_c and V_c are the edges and vertices of G from LCOM4.	Not validated	Attribute sharing and methods invocation	[49]
LCOM5	Consider a set of methods $\{M_i\}$ (i=1, … , m) accessing a set of attributes $\{A_j\}$(j=1, …, a). Let $\mu(A_j)$ be the number of methods that reference A_j. Then, $\text{LCOM5} = \frac{\frac{1}{a}\sum_{i=1}^{a}\mu(A_j) - m}{1-m}$	Not validated	Attribute usage	[47]

Metric	Definition	Validation	Cohesion Criteria	Source
TCC	Let NP be the maximum possible number of direct or indirect connection in a class. NP = $\frac{N*(N-1)}{2}$ for N methods. Let NDC be the number of directly connected methods in a class. Then TCC is defined as: TCC = $\frac{NDC}{NP}$	Not validated	Attribute sharing	[11]
LCC	Let NIC be the number of indirect connections in the class. Then LCC is defined as follows $LCC(C) = \frac{NDC(C) + NIC(C)}{NP(C)}$	Not validated	Attribute sharing	[11]
RCI (ratio of cohesive interaction)	$RCI(C) = \frac{\lvert Cl(C)\rvert}{\lvert Max(C)\rvert}$	Validated theoretically and empirically	Type and attribute usage	[17]
CAMC	$CAMC = \frac{\sum_{i=1}^{n} \lvert P_i \rvert}{\lvert T \rvert \times n}$ where n is the number of methods in the class; M_i is the set of parameters of method *I*; T is the union of M_i, for every *i* = 1 to n; and P_i is the intersection of set M_i with T i.e. $P_i = Mi \cap T$	Validated empirically	Type intersection	[6]
CBMC	$CBMC(C) = F_c(G_r(C)) x F_s(G_r(C)) = F_c(G_r(C)) x \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i(C))$ Where $F_c(G_r(C)) = \frac{\lvert M_g(G_r)\rvert}{\lvert M_n(G_r)\rvert}$ is the connectivity factor and $F_s(G_r(C)) = \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i)$ is the structure factor	Validated empirically	Attribute sharing, methods invocation and methods patterns	[26]

Metric	Definition	Validation	Cohesion Criteria	Source
CCM	$CCM(C = \frac{NC(C)}{NMP(C).NCC(C)})$ Where NC(C) = number of actual connection, NMP(C) = maximum possible connections and NCC(C) = number of connected components of the connection graph G_c	Validated theoretically	Attribute sharing and methods invocation	[52]
ECCM	$ECCM(C) = CCM(C) \times (1 - PenaltyFactor(C))$ Where $PenaltyFactor(C) = \frac{NORM(C)}{NOIM(C)}$ NORM(C) is the number of re-implemented methods and NOIM(C) is the number of inherited methods	Validated theoretically	Attribute sharing and methods invocation	[52]
OCC	$OCC(C) = \begin{cases} \max_{i=1,\ldots,n}\left[\frac{\lvert Rw(mi)\rvert}{n-1}\right], (n>1) \\ 0, (n=1) \end{cases}$	Validated theoretically	Attribute sharing and method invocations	[4]
PCC	$PCC(C) = \begin{cases} \max_{i=1,\ldots,n}\left[\frac{\lvert Rs(mi)\rvert}{n-1}\right], (n>1) \\ 0, (n=1) \end{cases}$	Validated theoretically	Attribute sharing and method invocations	[4]

CHAPTER 3

3 CLASSIFICATION FOR OBJECT-ORIENTED COHESION METRICS

3.1 Introduction

In this chapter, we begin in section 3.2 with the discussion of the classification criteria for object-oriented cohesion metrics. Section 3.3 presents a critical analysis of the existing object-oriented cohesion metric in the light of the classification criteria. In section 3.4 we conclude the critical analysis by summarizing our findings.

3.2 Classification for Cohesion Measurements

Software researchers have given so much importance to the area of software metrics with a view to quantify different aspects of software. If software features are accurately measured, the development process can better be understood and hence it can easily be controlled so that better software products are produced. In this section, we identify some classification criteria that can be used to classify, assess, compare, and evaluate the existing cohesion metrics. The classification criteria are of two types: factors and characteristics.

- **Factors:** these criteria identify what the metric considers in its calculation of cohesion value.
- **Characteristics:** these criteria, as the name implies, capture the characteristics of the cohesion metric i.e. the features of the metric.

As shown in Table 3, criterion 3, 7, 9, 10 and 11 can be found in [18]. The table presents a summary of these criteria where *new* (in the *source* column) signifies new criteria that we identified in the course of this research.

Table 3: Classification Criteria

#	Classification Criteria	Description	Source	Remark
1	Underlying Approach	Characteristic	New	
2	Granularity	Characteristic	New	
3	Availability	Characteristic	Briand [18]	Braind et. al. called this usable or partially usable .
4	Soundness/Validity	Characteristic	New	
5	Sensitivity	Characteristic	New	
6	Normalization	Characteristic	New	
7	Validation	Characteristic	Briand [18]	
8	Interpretation	Characteristic	New	
9	Connection type	Factor	Modified	Modified version of Braind's cohesion criteria [18].
10	Special Methods	Factor	Briand [18]	Braind et. al. called this known problems
11	Inheritance	Factor	Briand [18]	

Underlying Approach

To better understand a concept, it is important to understand the underlying principle upon which the work is built; this may be obtained by knowing where the original idea is obtained. Underlying approach gives the reference of the work where the original idea, upon which the approach is built, is obtained.

Granularity

Granularity refers to the level of granularity of the metric, as in which component of the system does the metric measure; method, class or package.

Availability

Availability determines the software engineering development process in which the metric can fully be used. Some metrics can only be used when coding is completed such metrics are available at the implementation level. While some metrics can be

used at the end of the design stage such metrics are available at the design level of the software development process.

Soundness/Validity

Soundness determines the correctness of the metric proposed in the approach, as in how much it really captures the cohesion of the module of a software system. It equally determines if there is an ambiguity in the metric computation. An ambiguity exist if the metric gives the same value for classes that are, intuitively, of different cohesion.

Sensitivity

Sensitivity describes how sensitive the cohesion metric is to changes. How does a change in the module (or class) affect the measurement? Does the change have negative or positive impact on the result of the metric?

Normalization

Normalization determines if the result of the metric is normalized i.e. values returned by the metric is between 0 and 1; classes with zero cohesion value have the least cohesion while classes with cohesion value 1 have perfect cohesion. Or the reverse is the case for inverse metrics like the LCOM metrics.

Validation

Validation specifies whether the metric is validated or not; if it is validated how is it validated-theoretically or empirically. If it is not validated how complex is the

validation process, as in what are the things required for it to be validated and whether or not the researchers have given a way that their metric can be validated.

Interpretation

Interpretation determines the difficulty surrounding the interpretation of the results obtained from the metric. It also describes whether the researchers have given suggestions on how to interpret the values of the metric or not.

Connection Type

Connection type specifies factors the cohesion metric considers in calculating the cohesion of the module i.e. the process the researchers used in capturing the interactions among the different components of a class. Based on our research, we outlined all the possible interactions that may exist among the components of a class in Table 4. However, we have not exhausted all possible types; the types outlined in Table 4 are based on the approaches we have covered in this research. If a new interaction criterion is proposed later, the table simply needs to be updated. The table is a modified version of the one presented in Briand's framework [18].

Special Methods

Methods like constructor, access methods etc have an impact on the cohesion of a class. The *special method* attribute captures whether the impact of such methods is considered in the definition of the cohesion measure.

Inheritance

Inheritance describes whether the approach considers the impact of inheritance in proposing its metric. Inherited methods and attributes have an impact on the cohesion of a class.

Table 4 presents all the possible types of connections used by the existing cohesion metrics (i.e. the possible ways through which the components of a class may interact). This table is a modified version of the work presented by Briand et al. in [18]; here we identified three more connection types: type 3, 8 and 9. In addition, we give names to each of the criterion for easy referencing.

Table 4: Connection Types

#	Element 1	Element 2	Description	Name	Measures
1	Method *m* of class c	Attribute *a* of class c	*m* references *a*	MAR	LCOM5, CBMC, CBAMU
2	Method *m* of class c	Method *m'* of class c	*m* invokes *m'* directly	DMMR	LCOM4, Co, CCM, ECCM, OCC, PCC, CBAMU
3	Method *m* of class c	Method *m'* of class c	*m* relates to *m'* indirectly via other methods that directly invoke each other.	IMMR	CBAMU
4	Method *m* of class c	Method *m'* of class c, $m \neq m'$	*m* and *m'* directly reference an attribute a of class c in common	DAS	LCOM1, LCOM2, LCOM3, LCOM4, Co, TCC, CCM, ECCM, OCC, PCC, LCC
5	Method *m* of class c	Method *m'* of class c, $m \neq m'$	*m* and *m'* indirectly reference an attribute *a* of class *c* in common	IAS	LCC
6	Data-declaration in class c	Data-declaration in class c	Data-data interaction	DDI	RCI
7	Method *m* of class c	Data-declaration in class c	Data-method interaction	DMI	RCI
8	Parameter 1	Parameter 2	The existence of Parameter 1 and Parameter 2 in the same method or otherwise	PPI	CAMC
9	Method *m* of class c	Method *m'* of class c	*m* writes to an attribute *a* of class c and *m'* reads *a*	MIBAT	PCC

Where;

MAR means Method-Attribute Referencing

DMMR means Direct Method-Method Referencing

IMMR means Indirect Method-Method Referencing

DAS means Direct Attribute Sharing

IAS means Indirect Attribute Sharing

DDI means Data-Data Interaction

DMI means Data-Method Interaction

PPI means Parameter-Parameter Interaction (or Intersection).

MIBAT: Methods Interactions Based on Access Types

3.3 Critical analysis of object oriented cohesion metrics

In this section we critically analyze the different approaches we found in the literature, all the object-oriented approaches discussed in chapter 2 (Literature Review) are scrutinized based on the classification criteria.

Evaluating the Degree of Method and Class Cohesion of Eder et al. [33]

Eder et al. define degrees for measuring method cohesion, class cohesion and inheritance cohesion as explained in chapter 2. In this section we discuss the proposed metrics in the light of the attributes discussed in Table 3. See section 2.4.1 for details regarding the cohesion metrics proposed by Eder et al.

Table 5: Eder's et al. Approach

#	Attribute	Approach
1	Underlying Approach	An extension of the concepts of cohesion developed initially for procedure-oriented systems by Yourdon and Constantine [80]
2	Granularity	Measure cohesion at method and class level
3	Availability	Implementation
4	Soundness/Validity	Too subjective; the degrees of method and class cohesion are ambiguous because their meaning cannot be determined from the context. The approach depends on individual to interpret the result
5	Sensitivity	Not sensitive
6	Normalization	Not normalized
7	Validation	Not validated
8	Interpretation	No explanation is given for the metric interpretation
9	Connection Type	Provides ordinal scales for capturing method cohesion, class cohesion and inheritance cohesion as explained in chapter 2. All the three types of cohesion are subjective and too difficult to measure automatically
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Inherited methods and attributes are considered in the case of inheritance cohesion.

Evaluating the LCOM1 and LCOM2 Metrics [28][29]

In their approach, Chidamber and Kemerer proposed the lack of cohesion in methods (LCOM) and later redefine this metric, we call the two metrics LCOM1 and LCOM2. These two metrics are critically analyzed using the classification criteria in Table 6 and Table 7, respectively.

Table 6: LCOM1

#	Attribute	Approach
1	Underlying Approach	Based on the notion of degree of similarity of methods initially proposed by Bunge [25]
2	Granularity	Measures cohesion at class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not correctly capture the cohesion of a class though it gives an idea of how cohesive a class is. The metric is ambiguous because two classes that are, intuitively, of different cohesion may have the same cohesion value.
5	Sensitivity	Reacts positively to changes though not in all situations. That is if the cohesion of a class is altered, the metric reflects this alteration in its computation sometimes.
6	Normalization	Not normalized
7	Validation	Validated theoretically
8	Interpretation	No explanation on how to interpret the result of the metric
9	Connection Type	DAS (Direct Attribute Sharing) see Table 4
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Not considered

Table 7: LCOM2

#	Attribute	Approach
1	Underlying Approach	Based on the notion of degree of similarity of methods initially proposed by Bunge [25]
2	Granularity	Measures cohesion at class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not correctly capture the cohesion of a class. The metric is ambiguous because two classes that are, intuitively, of different cohesion may have the same value for the measure. In the experiment performed by Basili et al [8] it turns out that many classes are set to have cohesion value zero although different cohesions are expected.
5	Sensitivity	Behaves positively to changes in the cohesion of a class though not in all situations
6	Normalization	Not normalized
7	Validation	Validated theoretically and empirically
8	Interpretation	The researchers here asserted that high value of LCOM2 is not desirable because high value of the metric indicates disparateness in the functionality provided by the class. This means that the class is attempting to achieve different objectives. Such classes could be more error prone and more difficult to test and could possibly be disaggregated into two or more classes that are better defined in their behavior. However, though a high value of LCOM2 implies low cohesion, a value of LCOM2 = 0 does not imply the reverse. As a matter of fact, two or more different classes may have the value of LCOM2 = 0. In such a case it is difficult to interpret the result of LCOM2 metric.
9	Connection Type	DAS (Direct Attribute Sharing) see Table 4
10	Special Methods	No consideration was given for special methods
11	Inheritance	Not considered

Evaluating The LCOM3, LCOM4 and Co Metrics [49][50]

Hitz and Montazeri proposed three metrics based on the LCOM metric; their proposed metrics are: LCOM3, LCOM4 and Co. In Table 8, Table 9, and Table 10 these metrics are carefully analyzed based on the classification criteria.

Table 8: LCOM3

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Captures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not correctly capture the cohesion of a class. The metric is not devoid of ambiguity, it always gives a value of 1 if the number of connected component in the undirected G_x graph has one connected component. In addition, the metric is not normalized.
5	Sensitivity	Not sensitive to changes, it does not always behave the way it should when the interactions among the components in a class are tempered with.
6	Normalization	Not normalized
7	Validation	Not validated

8	Interpretation	Difficult to interpret the result
9	Connection Type	DAS (Direct Attribute Sharing) see Table 4
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Not considered

In order to remedy the problems with access method when measuring the cohesion of a class, LCOM4 was proposed. In the definition of this metric, methods invocations are also put into consideration when drawing the undirected graph G_x. This metric is scrutinized in the table that follows.

Table 9: LCOM4

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Captures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not correctly measure the cohesion of a class; it always gives the value 1 when the number of connected component in the undirected graph is 1. The result of the metric is not normalized.
5	Sensitivity	Not very sensitive to changes, it does not always behave the way it should when the interactions among the components in a class are tempered with.
6	Normalization	Not normalized
7	Validation	Not validated
8	Interpretation	Difficult to interpret the result
9	Connection Type	Uses DAS and DMMR of Table 4
10	Special Methods	Considers access method by including an edge between methods in the undirected graph whenever one of the methods invokes the other.
11	Inheritance	Not considered

To further discriminate classes that have the value of LCOM = 1, Hitz and Montazeri proposed a third metric called connectivity (Co). This metric is analyzed in Table 10.

Table 10: The Connectivity Metric

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Captures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not correctly measure the cohesion of a class; it only comes into play when we have a situation whereby there is only one connected component in the reference graph of the class.
5	Sensitivity	Not very sensitive to changes, it does not always behave the way it should when the interactions among the components in a class are tempered with.
6	Normalization	Normalized
7	Validation	Not validated

8	Interpretation	No interpretation is provided for this metric
9	Connection Type	Uses DAS and DMMR of Table 4
10	Special Methods	Considers access method - this metric is an extension of LCOM4. The metric is proposed in order to further discriminate classes with one connected component in the undirected graph by considering the number of edges in the graph.
11	Inheritance	Not considered

Evaluating the TCC and LCC Metrics [11]

Bieman and Kang proposed two cohesion metrics: TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion). TCC and LCC are critically discussed based on the classification criteria in Table 11 and Table 12, respectively.

Table 11: Tight Class Cohesion

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Provides good means of measuring the cohesion of a class but at times it gives result that is contrary to intuition especially when there are many methods in the class. If there are a huge number of methods in the class, the value of the cohesion will be low because of the denominator in the definition of the metric but this is not always the case. It gives an infinite value for classes with no method or classes with only one method. Also, it does not capture the relationships between methods via method invocation. In order words, if two methods do not directly or indirectly share an attribute in common, the methods will be considered as unrelated methods, which is wrong.
5	Sensitivity	Not very sensitive.
6	Normalization	Normalized
7	Validation	Not validated
8	Interpretation	Difficult to interpret the result
9	Connection Type	Uses IAS of Table 4
10	Special Methods	Recommended that constructors be excluded
11	Inheritance	Suggest three alternatives for handling inherited attributes/methods as discussed in chapter 2.

The second metric; Loose Class Cohesion (LCC) considers - in its definition - both direct and indirect connections that may exist among the components of a class.

Table 12: Loose Class Cohesion

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Provides good means of measuring the cohesion of a class but at times it gives result that is contrary to intuition especially when there are many methods in the class. If there are a huge number of methods in the class, the value of the cohesion will be low because of the denominator in the definition of the metric but this is not always the case. Infinity is returned by this metric for classes with only one method and for classes with no method. Also, it does not capture the relationships between methods via method invocation. In order words, if two methods do not directly or indirectly share an attribute in common, the methods will be considered as unrelated methods, which is wrong.
5	Sensitivity	Not very sensitive.
6	Normalization	Normalized
7	Validation	Not validated
8	Interpretation	Difficult to interpret
9	Connection Type	Uses IAS of Table 4
10	Special Methods	Recommended that constructors be excluded in the analysis
11	Inheritance	Suggest three alternatives for handling inherited attributes/methods. See chapter 2 for details

Evaluating the LCOM5 Metric [47]

Henderson-Sellers et al. proposed a new metric by redefining the Lack of Cohesion in Methods (LCOM) metric; we call their metric LCOM5. Below, the metric is discussed based on the classification criteria.

Table 13: LCOM5

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Provides a process of measuring the cohesion of a class but it has some problems. If there is no attribute or if there is only one method in a class, the metric will give an infinite value as the cohesion of the class. This violates the principle of measurement theory and a good metric should not return such in any situation.
5	Sensitivity	Sensitive to changes
6	Normalization	Normalized
7	Validation	Not validated
8	Interpretation	Difficult to interpret
9	Connection Type	Uses MAR of Table 4
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Not considered

Evaluating the RCI Metric [15]

Briand et al. proposed cohesion metric, RCI (Ratio of Cohesive Interaction), based on the visualization of a class as a collection of data declarations and methods. This metric is critically discussed in Table 14.

Table 14: The RCI Metric

#	Attribute	Approach
1	Underlying Approach	Adapted from the early work of Briand et al. for measuring cohesion in object-based systems.
2	Granularity	Measures cohesion at the class level
3	Availability	Fully available at the design level
4	Soundness/Validity	Provides a good means of measuring the cohesion of class at the design level. But because it measure cohesion at the design level, the result of this metric is not always very accurate.
5	Sensitivity	Not very sensitive to changes
6	Normalization	Normalized
7	Validation	Validated theoretically and empirically
8	Interpretation	No interpretation was provided.
9	Connection Type	DDI and DMI of Table 4
10	Special Methods	Not considered
11	Inheritance	In addition to the three options provide by Bieman and Kang, Briand et al. added a fourth alternative for handling inheritance: excluding inherited attributes but include inherited methods. According to them this makes little sense because inherited methods can only access inherited attributes

Though special methods are not considered in the definition of the RCI metric, Briand et al. [18] provides the following suggestions on how to deal with special methods.

1. Suggested to count the invocation of access methods as reference to an attribute (for MAR and DAS in Table 4)
2. Suggest that access methods be excluded (for DAS and DIAS in Table 4) and
3. Suggest that constructors be excluded in the analysis.

Evaluating the CAMC Metric [6]

Bansiya et al. [6], proposed a metric whereby the cohesion of a class is determined by the types of objects that methods take as input parameters. The metric CAMC (Cohesion Among Methods in Class) measures the extent of intersections of individual

method parameter type lists with the parameter type list of all methods in the class. This metric is discussed in Table 15 in light of the classification criteria.

Table 15: The CAMC Metric

#	Attribute	Approach
1	Underlying Approach	Based on the premise that the parameters of a method reasonably define the types of interaction that methods may implement.
2	Granularity	Measures cohesion at the class level
3	Availability	Fully available at the design level
4	Soundness/Validity	Provides a means of measuring the cohesion of class at the design level. This result may or may not capture the actual cohesion of classes because at the design level detailed information is not available which may affect the cohesion of the class. It is discontinuous for classes with no methods.
5	Sensitivity	Not very sensitive
6	Normalization	Normalized
7	Validation	Validated empirically
8	Interpretation	No explanation on how to interpret the result of this metric is provided
9	Connection Type	PPI of Table 4
10	Special Methods	Not considered
11	Inheritance	Not considered

Evaluating the CBMC Metric [26]

Chae proposed the metric CBMC (Cohesion Based on Member Connectivity)

Table 16: The CBMC Metric

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Presents an excellent way to measure the cohesion of a class but have a number of problems. If there are two or more glue methods (i.e. methods that can separate the reference graph) the order by which these methods are removed from the structure tree determines the value of the class cohesion. Thus, different values might be obtained for the same class.
5	Sensitivity	Not very sensitive
6	Normalization	Normalized
7	Validation	Validated empirically
8	Interpretation	No explanation is provided regarding how to interpret the result of the metric
9	Connection Type	MAR of Table 4
10	Special Methods	Introduced the concept of glue methods in order to overcome the problems of special methods. Methods like access methods, constructors etc are made not have any impact on the cohesion of the class by ensuring that their removal in the reference graph does not separate the graph.
11	Inheritance	Not considered

Evaluating the CCM and ECCM Metrics [52]

Jarallah et al. [52] conducted a research on some of the object-oriented design features that can affect the cohesion of a class and attempted to relate how cohesion can be used to assess these design features. Two metrics were proposed: (i) CCM (Class Cohesion Metric) and (ii) ECCM (Enhanced Class Cohesion Metric). These two metrics are critically discussed in Table 16 and Table 17.

Table 17: The CCM Metric

#	Attribute	Approach
1	Underlying Approach	Based four design principles proposed in [52]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Measure the cohesion of a class but not accurately; at times it returns the same cohesion value for two classes that appear to be of different cohesion
5	Sensitivity	Not very sensitive to changes
6	Normalization	Normalized
7	Validation	Validated theoretically
8	Interpretation	No explanation on how to interpret the result of this metric is provided
9	Connection Type	DMMR and DAS of Table 4
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Not considered

The analysis of the second metric is given in the following table.

Table 18: The ECCM Metric

#	Attribute	Approach
1	Underlying Approach	Based four design principles proposed in [52]
2	Granularity	Measures cohesion at the class level
3	Availability	Implementation level
4	Soundness/Validity	Measure the cohesion of a class but not accurately; at times it returns the same cohesion value for two classes that appear to be of different cohesion by intuition
5	Sensitivity	Not very sensitive to changes
6	Normalization	Normalized
7	Validation	Validated theoretically
8	Interpretation	No explanation on how to interpret the result of this metric is provided
9	Connection Type	DMMR and DAS of Table 4
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Suggest (i) avoiding unused inherited methods in a subclass (ii) avoiding re-implementation of inherited methods

Evaluating the OCC and PCC [4]

Aman et al. proposed two metrics: (i) OCC (Optimistic Class Cohesion) and (ii) PCC (Pessimistic Class Cohesion), detailed explanation of how these two metrics work is presented in chapter 2. In Table 19 and Table 20, we critically analyzed these metrics based on the classification criteria.

Table 19: The OCC Metric

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not accurately measure the cohesion of a class. The fact that access methods are treated as normal methods means that certain interactions among methods cannot be captured and thus the overall cohesion of the class will not be accurately captured. Access methods reduce the cohesion of a class.
5	Sensitivity	Not very sensitive
6	Normalization	Normalized
7	Validation	Validated theoretically
8	Interpretation	No interpretation was given
9	Connection Type	DMMR, DAS and MRBAT of Table 4
10	Special Methods	Access methods are treated as normal methods
11	Inheritance	Not considered

Table 20: The PCC Metric

#	Attribute	Approach
1	Underlying Approach	Based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level but fully available at the implementation stage.
4	Soundness/Validity	Does not accurately measure the cohesion of a class. The fact that access methods are treated as normal methods means that certain interactions among methods cannot be captured and thus the overall cohesion of the class will not be accurately captured. Access methods reduce the cohesion of a class.
5	Sensitivity	Not very sensitive
6	Normalization	Normalized
7	Validation	Validated theoretically
8	Interpretation	No interpretation was given
9	Connection Type	DMMR, DAS and MRBAT of Table 4
10	Special Methods	Access methods are treated as normal methods
11	Inheritance	Not considered

3.4 Observations

Based on the critical analysis of state-of-the-art of existing cohesion metrics, the following are our findings:

1. Most of the approaches are based on the work of Chidamber and Kemerer which is based on the notion of degree of similarity of methods initially proposed by Bunge.
2. Most of the approaches are based on attribute usage, method invocations or both.
3. Most of the metrics studied capture cohesion at the class level.
4. None of the metrics studied captures cohesion at the package level.
5. Most of the cohesion metrics studied do not accurately capture the cohesion of a class without violating at least one example.
6. Most of the metrics are not validated and few researchers provide explanation on how to interpret the result of their metrics.
7. Some of the metrics are not normalized.

In an attempt to address some of the short comings of some of the existing cohesion metrics, a new metric has been developed in this research. This metric is presented in chapter 4. The metric is based on LCOM5 and addresses some of its shortcomings.

CHAPTER 4

4 NEW COHESION METRIC

4.1 Introduction

Software metrics can help address the most critical issues in software development and can provide support for planning, predicting, monitoring, controlling, and evaluating the quality of both software products and processes [7]. Quite a number of object-oriented cohesion metrics have been proposed; we identified lapses in the definition of some of the object-oriented cohesion metrics. In this chapter we propose a new metric for measuring cohesion at the design level, which overcomes some of the problems identified with LCOM5.

In addition to having a context and explicit goals, a well defined metric should have the following in order to be complete:

1. A metric should have a measure (expressed as a numerical value)
2. A metric should provide a simple procedure or process for capturing the software attributes it measures.
3. The result of a metric should be normalized for easy understanding and easy comparison.
4. A metric should also provide an interpretation for the measure (the numerical value)

Briand et al. [15] proposed four strategies for proposing high-level design metrics: (i) declaration counts (ii) metrics based on the USES relationships (iii) metrics based on the IS_COMPONENT_OF relationships (iv) interaction-based.

The metric we proposed in this research work is based on the second strategy (i.e. metrics based on the USES relationships). Though not all the USES (Interactions) can be exhaustively captured at the end of the software design stage, the information available at this stage can be used to define a metric that can be used at the design level. We call our proposed metric CBAMU (Cohesion Based on Attribute and Method Usage). In computing this metric, we simply keep track of all the methods that use (access) each of the attributes in a class and all the methods that use (invoke) each of the method in the class. The metric (CBAMU) is also normalized so that cohesion values obtained from the metric lies between 0 and 1.

4.2 Cohesion Based on Attribute and Method Usage

Most of the existing class cohesion metrics attempt to measure the cohesion of a class by taking into account only the interactions among methods and the attributes of a class. This type of cohesion criteria constitutes a restrictive way of capturing the cohesion of a class [5]. The new metric, CBAMU (Cohesion Based on Attribute and Method Usage) is defined based on both attribute usage and method usage (invocation) within a class. The metric does not only considers, in its definition, the direct interaction between methods and attributes but also the interaction between methods which may serve as a means of capturing the indirect relationship among methods as shown in Figure 10.

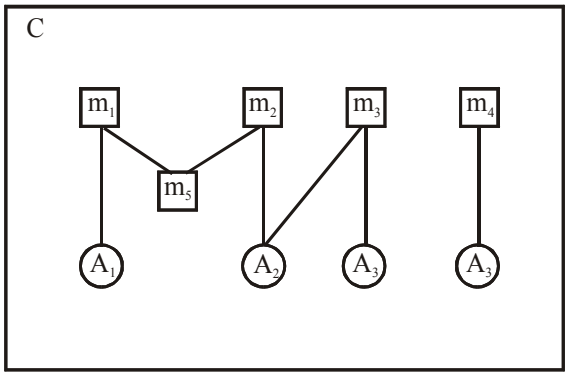


Figure 10: Direct and Indirect Connections

From Figure 10, we can see that m_2 and m_3 are directly related because they both access the same attribute A_2. This is the only interaction that can be captured if the cohesion criterion is only attribute usage. In order to capture the direct interaction between m_1 and m_2 via m_5, and subsequently the indirect connection between m_1 and m_3, we need to consider the method usages in a class.

Definition 4.1

Let C denote a class, A_i the set of attributes and M_j the set of methods in the class. Consider an undirected graph $G_c(V, E)$ where $V = M_j \cup A_i$ (methods are represented in rectangular nodes while attributes are represented in circular nodes) E is the set of all edges in G_c. An edge is drawn from a method to an attribute or another method, in the class, if the method accessed the attribute or invoked the method.

Definition 4.2

Let C denote a class, A_i the set of attributes and M_j the set of methods in the class. Let the total number of attributes in the class be *a* and the total number of methods in the

class be *m*. The attribute usage of the class (AU(C)) and method usage of the class (MU(C)) can be computed using equations 4.1 and 4.2, respectively.

$$AU(C) = \begin{cases} 0, \ if\ a = 0\ (or\ if\ m = 0) \\ \dfrac{1}{am}\sum_{i=1}^{a}\mu(A_i), \ otherwise \end{cases} \quad \text{----------[4.1]}$$

$$MU(C) = \begin{cases} 0, \ if\ m = 0\ (or\ if\ m = 1) \\ \dfrac{1}{m(m-1)}\sum_{j=1}^{m}\mu(M_j), \ otherwise \end{cases} \quad \text{----------[4.2]}$$

The CBAMU(C) is given by the following mathematical expression

$$CBAMU(C) = \frac{1}{2}\cdot\left(AU(C) + MU(C)\right) \text{----------[4.3]}$$

Where;

a = number of attributes in the class

m = number of methods in the class

$\mu(A_i)$ = number of methods that access attribute A_i

$\mu(M_j)$ = number of methods that invoke method M_j

From equations 4.1 and 4.2, we can see that dividing $\sum_{i=1}^{a} \mu(A_i)$ by *am* and $\sum_{j=1}^{m} \mu(M_j)$ by *m(m – 1)* will normalized the result. In the same vain, the summation of AU(C) and MU(C) is divided by 2 in order to keep the value obtained from the CBAMU metric normalized.

4.3 Validation

In recent time, so much attention has been given to the concept of software measurement probably due to the fact that most software projects fail. In view of this, numerous software metrics have been proposed so that features of software can easily be measured. One major criticism of most of these metrics is that they have not been validated, by validating a metric we need to show that it actually measure whatever it claims to measure [46].

The most common approaches for validating software metrics are theoretical validation and empirical validation; these two approaches complement each other. In other words, if a metric is validated theoretically, it needs to be validated empirically before it can be used with confidence [46]. In this section, we theoretically validate the CBAMU metric proposed in section 4.1.

4.3.1 Theoretical Validation

Several researchers have proposed properties that software metrics should posses in order to increase their level of confidence. It is desirable to have a formal set of criteria with which to evaluate proposed metrics.

In 1996, Hitz and Montazeri used the concept of measurement theory to evaluate and validate any given metric. They identified the significance of establishing a "sufficient" empirical relation system after the researcher has identified his attributes of interest. Having established an empirical relation system, a metric M should then map the empirical relation system into an appropriate formal (or numerical) relation system, preserving the semantics of the empirical relation(s) observed. In other words, for every empirical relation $\angle$ and a corresponding formal relation <, the so-called representation condition $X \angle Y \Leftrightarrow M(X) < M(Y)$ must hold [50]. The task of validating a software measure in the assessment sense is equivalent to demonstrating empirically that the representation condition is satisfied for the attribute being measured [40].

So, the empirical relation will be stated as: *The more edges in the interaction graph,G_X, the higher the cohesion of the class X.* Therefore any metric M should preserve the semantic of empirical relation.

In 1998, Briand et al. have proposed a mathematical framework including properties to be satisfied by several types of software metrics [18]. Cohesion measure is one of the measures supported by this framework, others include: size, length, coupling and complexity. The following properties are proposed with respect to cohesion metrics, in other words, any well defined metric should satisfy the following conditions.

Property 1: Non-negativity and Normalization

The cohesion of a class of an object oriented system should belong to a specified interval (i.e. Cohesion (C) ε [0, Max]. Normalization allows meaningful comparisons between the cohesions of different classes, since they all belong to the same interval.

Property 2: Null value and maximum value

The cohesion of a class of an object oriented system is null if there is no interactions among the components of the class (i.e. interaction among the methods and attributes of the class) and it is maximum if the interaction among the components is maximal.

Property 3: Monotonicity

Let C be an object-oriented system, and c $\in$ C be a class in C. Assuming we modified the class c to form a new class c' which is identical to c except that there are fewer interactions in c than in c'. Let C' be the object-oriented system which is identical to C except that class c is replaced by class c'. Then

$$[cohesion(c) \leq Cohesion(c') \mid Cohesion(C) \leq Cohesion(C')]$$

In other words, if a relationship is added to an object-oriented system, cohesion must not decrease.

Property 4: Merging of unconnected classes

Let C be an object-oriented system, and c_1, $c_2 \in$ C be two classes in C. Let c' be the class which is the union of c_1 and c_2. Let C' be the object-oriented system which is

identical to C except that classes c_1 and c_2 are replaced by c'. If no relationship exist between classes c_1 and c_2 in C, then

$$[\max\{Cohesion(c_1), Cohesion(c_2)\} \geq Cohesion(c') \mid Cohesion(C) \geq Cohesion(C')]$$

In other words, the merging of two unconnected classes must not increase cohesion (because the union of two unconnected classes will have little cohesion).

Hermadi et al. [48] augmented two other additional properties to Briand's et. al. framework, that cohesion metrics need to satisfy; these are symmetry and transitive. These properties are defined below.

Symmetry: the cohesion of a class should not be sensitive to the direction of the relation between its components. If there is a relation between m1 and m_2 then the representation of $m_1 \rightarrow m_2$ is equivalent to $m_2 \rightarrow m_1$.

Transitivity: Consider three classes c_1, c_2 and c_3 such that, Cohesion (c_1) < Cohesion(c_2) and Cohesion(c_2) < Cohesion(c_3), then Cohesion(c_1) < Cohesion(c_3).

Theoretical Validation of CBAMU

In this study we will use the following seven properties in addition to the property proposed by Hitz and Montazeri in the theoretical validation of the proposed cohesion metric (CBAMU).

1. Non-negativity
2. Normalization
3. Null value and maximum value
4. Monotonicity
5. Merging of unconnected classes
6. Symmetry
7. Transitivity

Going by the definition of CBAMU, we can see that the more the number of edges in the undirected graph (representing the interactions among the class's component) the more the cohesion of the class. Therefore, CBAMU satisfies Hitz's property (i.e. the representation condition holds).

The CBAMU of a class = 0 if there is no interactions among the components of the class and CBAMU = 1 if all methods are directly or indirectly connected (i.e. if the interactions among the components of the class is maximal). Therefore, the value of CBAMU lies in the interval [0, 1] inclusive. Hence, the metric satisfies the first three of the seven properties.

The monotonicity property says that if relationships are added to the system, then cohesion must not decrease. From the definition of CBAMU, we can see that as relationships are added, the values of $\sum \mu(M)$ and $\sum \mu(A)$ (i.e. the numerators) will increase while the number of attributes and methods remain fixed. Since while the numerator is increasing the denominator remains unchanged, the cohesion must increase. Therefore, CBAMU satisfies the fourth property (monotonicity).

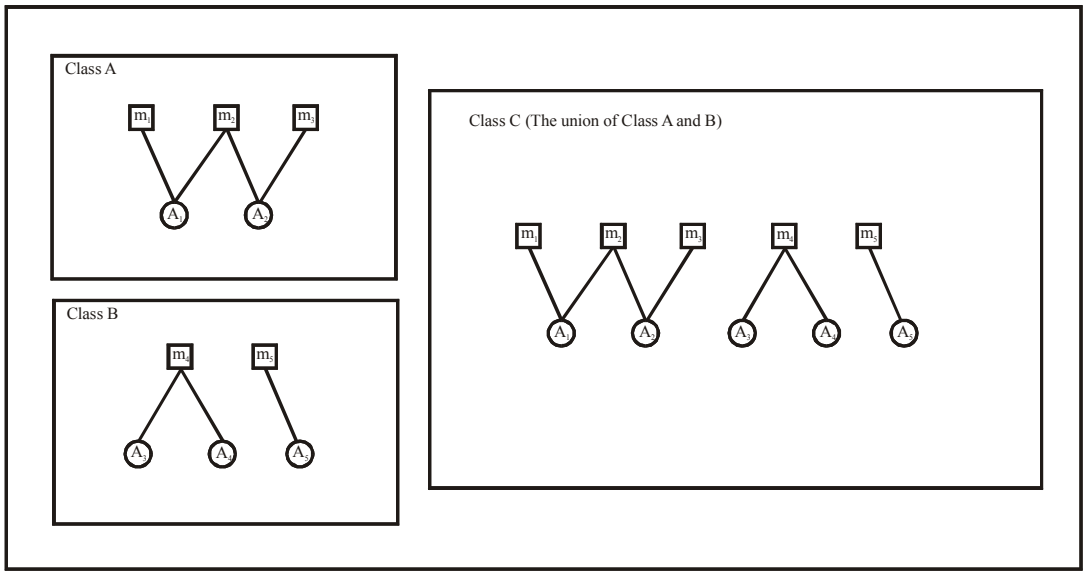


Figure 11: Merging two unconnected classes

Property five says that merging two unconnected classes must not increase cohesion. By merging any two classes that are unconnected, we are increasing the number of connected components as shown in Figure 11. This will lead to a decrease in CBAMU because while $\sum \mu(M)$ and $\sum \mu(A)$ slightly increases the values of *am* and *m(m-1)* drastically increase thereby causing an overall decrease in CBAMU. Thus, CBAMU satisfies the fifth property (merging of unconnected classes).

CBAMU does not satisfy the sixth property (symmetry) because in the case of interactions among methods (method invocations) direction is considered; we are interested in capturing which method invokes which not just the interaction.

Class C is more cohesive than class C' if, in the connection graph, there are less number of connected components in C than in C' or if there are more interactions in C than in C'. CBAMU will always show that a class with less number of connected

components (or with more interactions) has higher cohesion than a class with more connected components. Hence, if we have three classes A, B and C, such that Cohesion (Class A) < Cohesion (Class B) and Cohesion (Class B) < Cohesion (Class C). Then it implies that Cohesion (Class A) < Cohesion (Class C). Therefore, CBAMU satisfies the seventh property (transitivity)..

To conclude this chapter, the proposed metric (CBAMU) is exposed to the same treatment as the remaining cohesion metrics we studied in the cause of this research. Table 21 presents the critical analysis of the CBAMU metric.

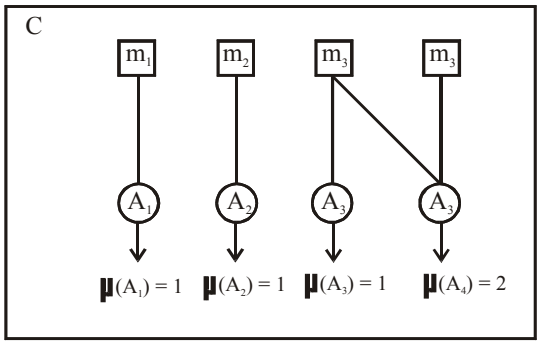
Table 21: The CBAMU Metric

#	Attribute	Approach
1	Underlying Approach	The approach is based on the work of Chidamber and Kemerer which is based on the work of Bunge [25]
2	Granularity	Measures cohesion at the class level
3	Availability	Partially available at the design level
4	Soundness/Validity	The metric may not give accurate results for classes with large number of methods because of the denominator in the definition of the metric. If there is large number of methods in the class, the value of the cohesion will be low which may not always be the case.
5	Sensitivity	The metric is sensitive to changes.
6	Normalization	Normalized
7	Validation	Validated theoretically
8	Interpretation	Difficult to interpret
9	Cohesion Criteria	Uses MAR, DMMR, IMMR of Table 4
10	Special Methods	No consideration was given for special methods in this approach
11	Inheritance	Not considered

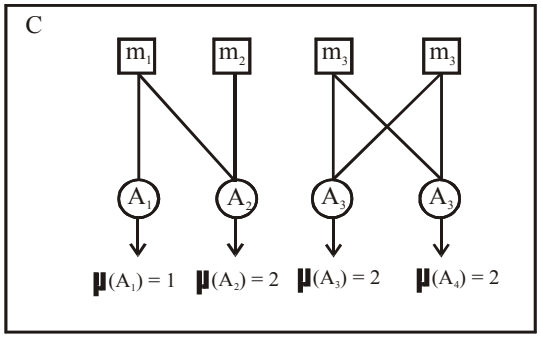
As can clearly be seen from our careful scrutiny of object oriented cohesion metrics in chapter 3, cohesion may be considered as a subjective concept. Hence, we do not claim that our metric accurately captures the cohesion of a class; this is simply our intuition of class cohesion.

4.3.2 LCOM5 vs. CBAMU

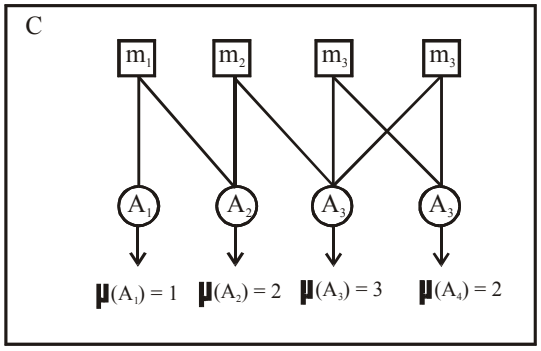
Consider the following three classes, which are intuitively of different cohesion.



LCOM5 = 11/12 = 0.917
CBAMU = 5/96 = 0.052



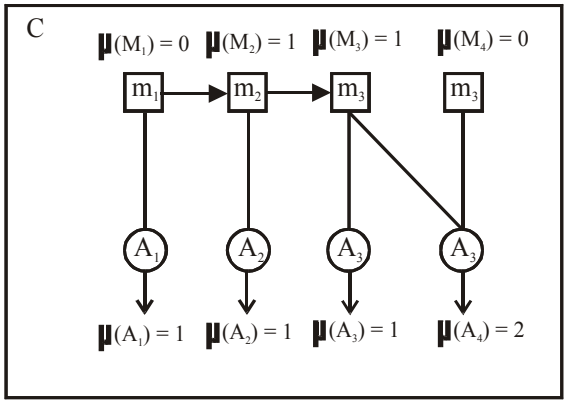
LCOM5 = 3/4 = 0.75
CBAMU = 7/64 = 0.109



LCOM5 = 2/3 = 0.667
CBAMU = 1/4 = 0.25

From the above classes, we can see that both LCOM5 and CBAMU correctly differentiate among the cohesion of the classes. Both metrics show that as the interaction among the component of the class increases, the cohesion value also increases. However, LCOM5 is solely based on attribute referencing and thus will not show an increase in the cohesion of a class where there is method invocation in addition to attribute referencing. This is shown in the following example where there are four methods and four attributes like in the first example the only different is that m1 invokes m2 and m2 invokes m3. Unlike CBAMU, LCOM5 could not differentiate

between the cohesion of this class and the one in the first example. But CBAMU was able to differentiate between the cohesion of these classes.



LCOM5 = 11/12 = 0.917
CBAMU = 23/96 = 0.24

LCOM5 satisfies all the properties used in theoretical validating CBAMU, while CBAMU fail to satisfy one of them (the symmetry property). However, one of the major problems with LCOM5 is that it returns infinity for classes with only one method. Moreover, the symmetry property that CBAMU failed to satisfy is not an agreed upon property by all researchers in the area of software metrics.

4.4 Implementation

In order to easily compute and conduct experiments with object-oriented software metrics, quite a number of software metrics tools have been developed. The main goal of such tools is to increase system quality and to predict relevant system qualities such as fault-proneness, maintainability etc. OOMeter is a tool that can capture object-oriented software metrics from UML models stored in XMI [56]. We extended this tool to support more metrics by implementing cohesion metrics. In this section we give a description of OOMeter.

4.4.1 OOMeter Architecture

OOMeter is a software metrics tool that measures the structural properties of UML models and java code and computes a number of software measures that include coupling, cohesion, and complexity. The tool contains four main components: Java parser & XMI parser, Data repositories for storing source data and metrics output as shown in Figure 12.

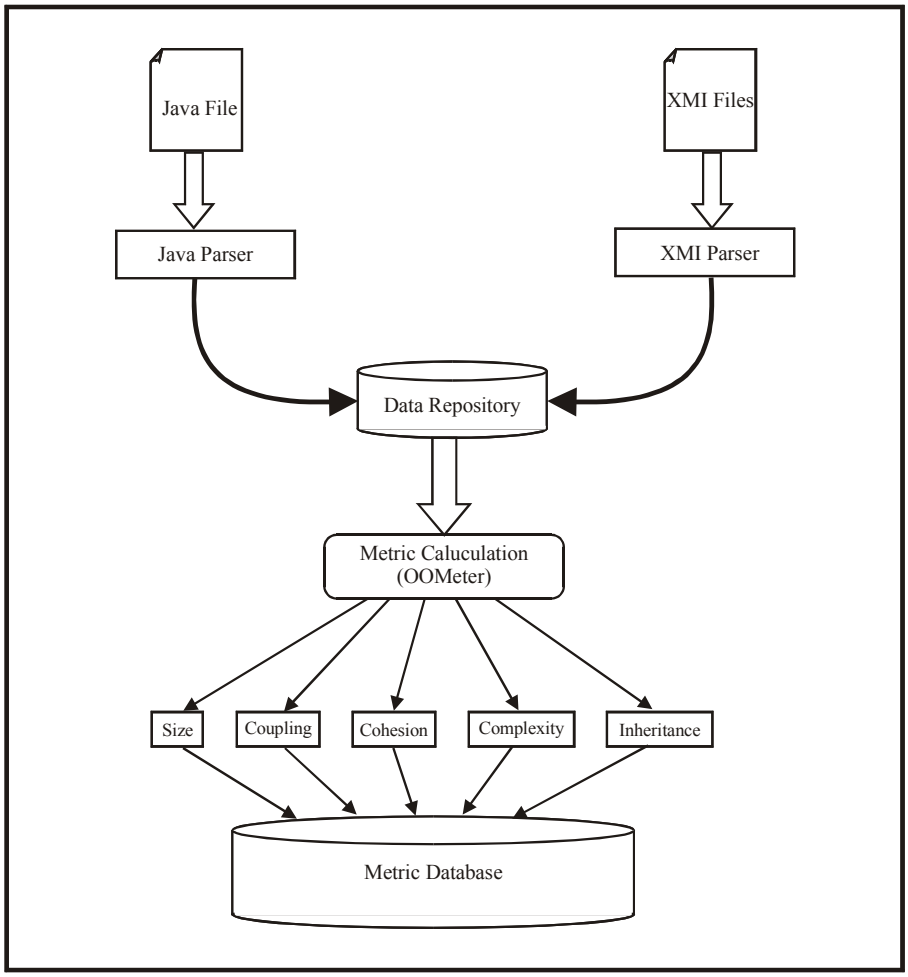


Figure 12: OOMeter Architecture

As shown in Figure 12, object-oriented systems are parsed to the tool in order to collect the data that can be used in computing the various software metrics supported by the tool. The collected data is stored into a Central Data Repository and the results

of the computation of all the metrics are stored in a different database called Metric Database. At the moment the tool supports the parsing of both java source files as well as UML models stored in XMI format. The tool supports the computation of a variety of software measures, which includes size, coupling, cohesion and complexity measures.

4.5 Empirical Validation

To empirically validate any software metric, well documented software projects are required but such projects are hard to find. However, to demonstrate the effectiveness or ineffectiveness of the proposed cohesion metrics, we performed a case study on several open source software systems. Our approach in this study is as follows:

- Let S_n be the set of all systems to be used
- For each system in S_n determine the following:
  - Total number of defects per system (or per class if available)
  - The test coverage for each system
- Normalized the test coverage with respect to the bugs
- Rank the projects based on normalized (modified) bugs per unit size.
- Compute the average class cohesion of the metric and rank the systems based on magnitude.
- Compare the ordering in 4 and 5, if they match then the metric is valid else the metric claim is questionable.

Note: *the above steps can be used as a practical approach for validating any OO cohesion metric.*

4.5.1 Hypothesis

In this part of the research work we will investigate the following hypothesis which is

simply derived from our understanding of the concept of cohesion. However, the hypothesis is not experimentally shown to be true by previous research work. The hypothesis is:

A project with low average class cohesion is likely to have high number of defects than a class with high average class cohesion.

4.5.2 Selected Systems

A total of seven projects were collected from SourceForge.net, which is an open source website that provides a centralized place where open source developers can control and manage open source software development.

The projects used in this study are collected based on their percentile values, which give an idea of how frequent the project is used. The percentile is expressed in a scale of 100; based on page views and download information. The higher the percentile the more used the project is likely to be and the more we expect bugs to be discovered in the project. On the other hand, if a project has low percentile then the bugs reported regarding this project may not be an indicator of its quality. It is worth mentioning here that; it does not always hold true that a user uses a project by merely visiting the projects website or by downloading it. However, it is an indicator that the project is popular and perhaps many of the people that download it may have used it. Details of the projects used are shown in Table 22.

Table 22: Selected Projects

Project	Class	Method	Attribute
Babeldoc 1.0	212	1541	936
Checkstyle 2.4	58	492	228
JGraph 2.0	29	750	340
VR Juggler 1.1DR3	278	2502	1338
Saxon 6.5.2	344	3252	1678
Jext 3.2	553	3233	2435
Saxon 8.0	540	4881	3298

4.5.3 Results and Analysis

On computing the different cohesion metric values, we noticed that some of the metrics have some strange values. One striking thing is that exactly the same cohesion metrics are affected in all the projects. These metrics are: TCC (Tight Class Cohesion), RCI (Ration of Cohesive Interaction), CAMC (Cohesion among Methods of Classes), and LCOM5. The cause of this problem was carefully investigated and we discovered that the affected cohesion metric values of some classes are infinite. The codes of such classes reveal that some of the classes have only one method in them while others don't even have a single method. This calls for us to revisit our implementation and to carefully study the definition of these metrics. At the end of our investigation, we came to understand that the cause of the problems was in the definition of the cohesion metrics.

4.5.3.1 Problems in the Definition of Some Cohesion Metrics

The problems in the definition of LCOM5 have already been discussed at length in chapters 3 and 4. In this sub-section, we would like to discuss the problems of other cohesion metrics which we came to know from the empirical study. At least three other metrics have similar problem as LCOM5 i.e. they give infinity for some classes. TCC and LCC are defined as follows:

$$TCC(C) = \frac{NDC(C)}{NP(C)}$$

$$LCC(C) = \frac{NDC(C) + NIC(C)}{NP(C)}$$

NP is given by NP(C) = N (N – 1) / 2, where N is the number of methods in the class. From this we can see clearly that the problem is in the definition of these metrics. Both

metrics are discontinuous for classes with one method as well as for classes with no method.

Another metric with a similar problem is CAMC (Cohesion Among Methods of Classes). The CAMC metric is defined as follows:

$$CAMC = \frac{\sum_{i=1}^{n} |P_i|}{|T| \times n}$$

Where *n* is the number of methods. The fact that n is the number of methods will make the whole denominator to be zero for classes with zero number of methods and will result in making the metrics to return infinite values for such classes.

A tempting approach to unravel this problem is to consider eliminating all the classes that have this problem. However, this solution will not work because the classes to be eliminated may be the cause of some bugs in the whole project. This will lead us to having an inconsistent result; on one hand we are not considering the classes while on the other hand we are considering their impact on the quality of the system. In the cause of this experiment, what we did was to consider classes with no method to have no cohesion because if there is no method in a class it means that there is no interaction among the component of that class. Classes with only one method are considered to have perfect cohesion; because we expect a class with only one method to have only one functionality. The average values of the cohesion metrics used in this analysis are given in Table 23.

Table 23: Average class cohesion of the projects

Project	Babeldoc	Saxon 6.5.2	Saxon 8.0	Checkstyle	Jext	VR Juggler	Jgraph
LCOM1	0.968	0.921	0.933	0.872	0.947	0.938	0.420
LCOM2	0.969	0.923	0.935	0.869	0.948	0.938	0.418
LCOM3	0.894	0.872	0.880	0.864	0.921	0.868	0.700
LCOM4	0.872	0.845	0.855	0.875	1.000	0.846	0.708
LCOM5	0.267	0.401	0.296	0.298	0.480	0.190	0.144
CBAMU	0.130	0.151	0.165	0.047	0.100	0.092	0.073
CCM	0.129	0.167	0.172	0.154	0.083	0.101	0.069
TCC	0.102	0.156	0.155	0.093	0.149	0.098	0.059
LCC	0.118	0.173	0.178	0.103	0.098	0.110	0.095
CAMC	0.514	0.384	0.358	0.369	0.462	0.477	0.318

4.5.3.2 Projects Ranking

Owing to the fact that no information is provided regarding the test coverage of the systems, we simply ranked the projects based on bugs per unit size. The size metric used for this purpose is discussed below.

Khan in [56] proposed a UML class size metric based on attribute size, method size and inner class. The weights complexities of data types are proportional to their size in java.

Attribute Type	Attribute Size
Int	4
Byte	1
Short	2
Long	8
Float	4
Double	8
Char	2
Boolean	1
String or any other object type	20

The method size is given by the following equation

$$Size(m) = \left(\sum_{i=1}^{np} Size(param_i) + Size(return) \right) \times n(message)$$

The class size is given by

$$Size(c) = \sum_{i=1}^{na} Size(att_i) + \sum_{i=1}^{nm} Size(meth_i) + \sum_{i=1}^{ni} Size(innerClasses)$$

Based on the size metric above, the projects are ranked in order of bug per unit size as shown in Table 24.

Table 24: Project ranking based on bug/design size metric

Project	Total Bugs	Design Size Metric	Bug/Size	Rank
Babeldoc 1.0	4	17116	0.000233699	1
Saxon 6.5.2	36	63048	0.000570994	2
Saxon 8.0	26	29620	0.000877785	3
Checkstyle 2.4	5	3174	0.001575299	4
Jext 3.2	100	23615	0.004234597	5
VR Juggler 1.1DR3	136	20860	0.006519655	6
JGraph 2.0	31	3862	0.008026929	7

The project with rank 1 has the least fault density while the project with rank 7 has the highest fault density. Based on our hypothesis, we expect to have similar ranking from the cohesion metric values of the seven projects. However, on computing the cohesion metrics, none of them show this ranking as shown in Figure 13. We started this experiment with three projects; in this case the results were good. However, on using the whole seven projects none of the metrics follow our intuition.

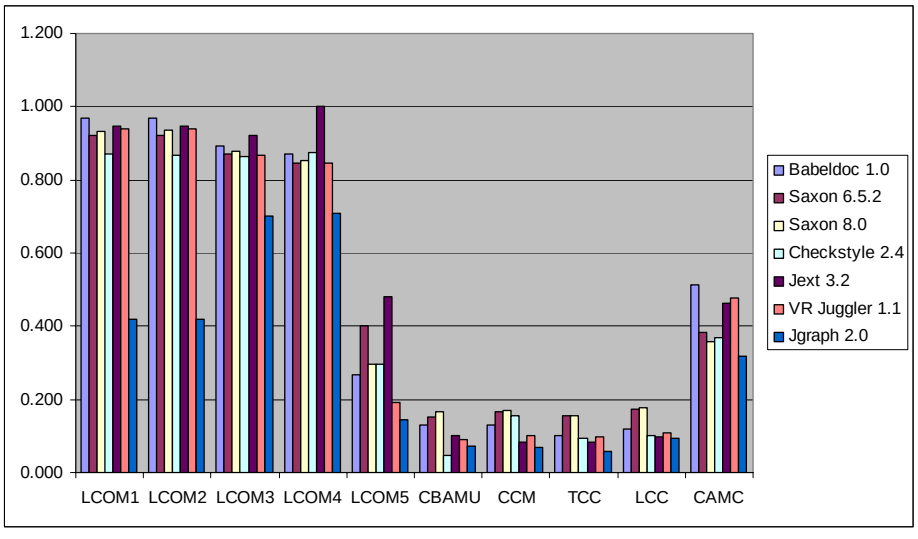


Figure 13: Project ranking based on cohesion results

From the charts presented in Figure 13, we can see that none of the cohesion metrics follow the ordering we have in (i.e. ordering based on quality). The project ordering in the legend of Figure 13 is the same as the ordering based on quality (in Table 24). As clearly seen none of the considered cohesion metrics follow this ordering. Therefore, we can conclude that the existing cohesion metrics are not highly correlated to the number of defects in a software system, therefore, cannot be good predictors of software defects. To support the above assertion, that the existing cohesion metrics are not good predictors of software defect, we built software defect prediction model using an object oriented software project downloaded from NASA Metric Data Program website. Details of these models are given in Chapter 5.

In order to determine which metric has the closest ordering, we generate the spearman's rank correlations. Results show that LCC's ordering is the closest to the ordering based on defect density as shown in Figure 14.

Spearman Rank Order Correlations (Spreadsheet10.sta)
MD pairwise deleted
Marked correlations are significant at p <.05000

Project	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	CBAMU	CCM	TCC	LCC	CAMC
1.000000	0.428571	0.428571	0.535714	0.250000	0.392857	0.642857	0.600000	0.642857	0.750000	0.392857

Figure 14: Spearman rank order correlations based on the design size metric

The projects are also ranked based on LOC, the LOC is computed using Borland Together Control Center. This ranking is shown in Table 25.

Table 25: Project ranking based on Bug/LOC

Project	Total Bugs	LOC	Bug/LOC	Rank
Babeldoc 1.0	4	24588	0.000163	1
Saxon 8.0	26	62482	0.000416	2
Checkstyle 2.4	5	5941	0.000842	3
Saxon 6.5.2	36	34468	0.001044	4
Jext 3.2	100	58848	0.001699	5
JGraph 2.0	31	9096	0.003408	6

On generating the spearman's rank correlations we have similar result, i.e. LCC has the closest correlation to that of defect density as shown in Figure 15.

Spearman Rank Order Correlations (Spreadsheet10.sta)
MD pairwise deleted
Marked correlations are significant at p <.05000

Variable	Project	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	CBAMU	CCM	TCC	LCC	CAMC
Project	1.000000	0.600000	0.600000	0.371429	0.257143	-0.142857	0.428571	0.600000	0.257143	0.714286	0.428571

Figure 15: Spearman rank order correlations based on LOC

4.5.3.3 Linking Results to Classification Criteria

The mean values of the normalized LCOM's as well as the remaining cohesion metrics are shown in Figure 13; we can see some clusters from this plot. To clearly see these clusters, we order the projects based on the individual metric values. When the projects are ordered based on the LCOM1 values, both LCOM1 and LCOM2 have exactly the same pattern as shown in Figure 16. Also based on this ordering, we can see that LCOM1 to LCOM4 have similar pattern as shown in Figure 17.

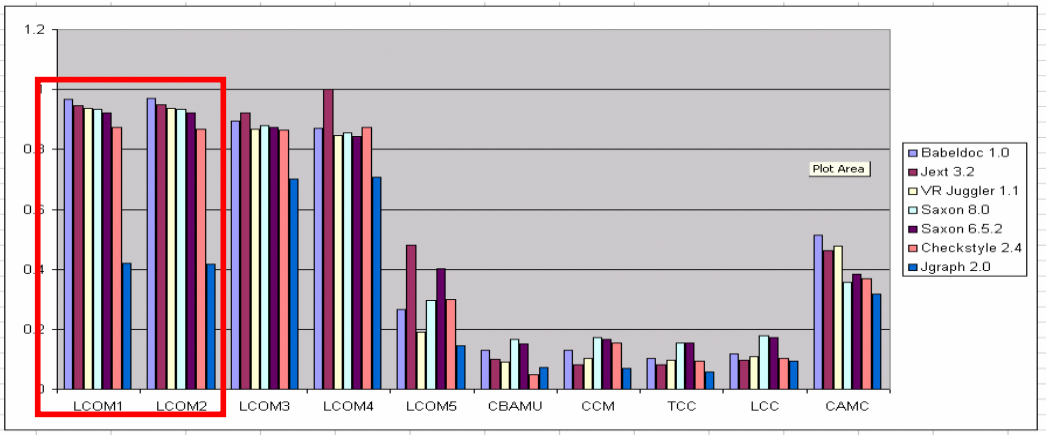


Figure 16: Cluster I (LCOM1 & LCOM2)

We expect LCOM1 and LCOM2 to have similar patterns because they both use the same Connection Type (i.e. DAS in Table 4). Furthermore, LCOM2 is simply an extension of LCOM1.

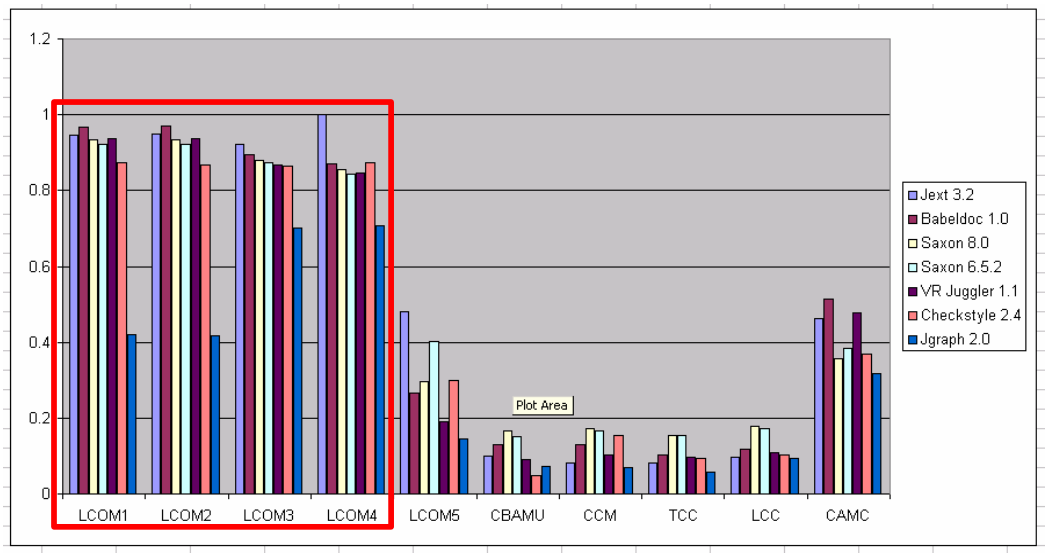


Figure 17: Cluster II (LCOM1-4)

The fact that LCOM1 to LCOM4 have similar patterns, though not exactly the same, is an indication that they all have something in common. All these metrics use the same Connection Type (i.e. DAS in Table 4); though in addition to DAS, LCOM4 uses Connection Type 2 (i.e. DMMR in Table 4).

When the projects are ordered based on TCC mean values, we can see that TCC and LCC have similar patterns. And if the CheckStyle project is pulled out of the analysis, CCM would have exactly the same pattern as LCC; all these metrics use the DAS connection type (though CCM uses the DMMR connection type in addition to DAS and LCC uses IAS in addition to DAS). So we expect these three metrics to have similar pattern as shown in Figure 18.

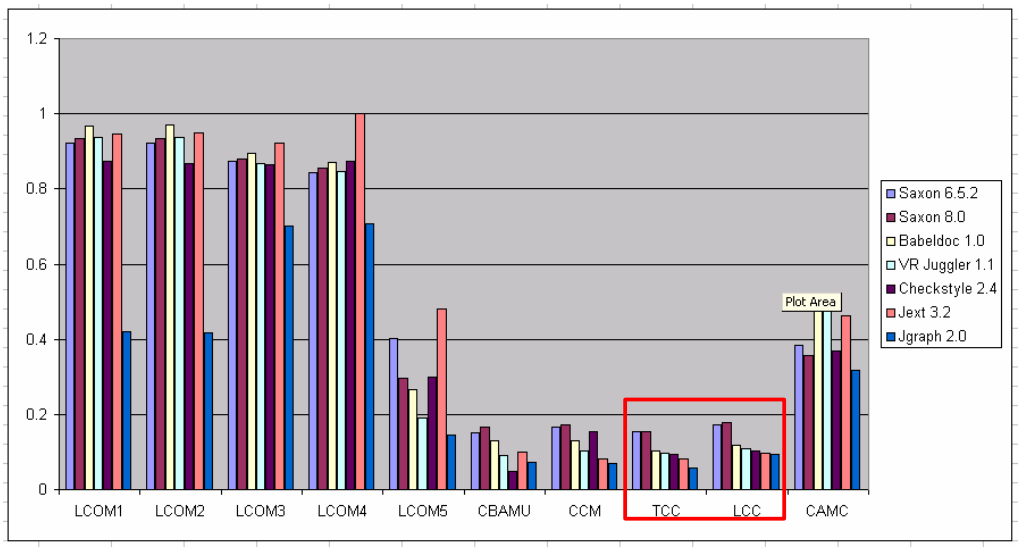


Figure 18: Cluster III (TCC & LCC)

To clearly see the correlations among the cohesion metrics, we generate a correlation matrix of the metrics. The above charts are based on the average cohesion values of the seven projects presented in Table 22. Using this in generating the correlations means that we have only seven cases. In order to have more cases, we use a single project (JDSL), which has 99 classes; this means that we have 99 cases. JDSL is the Data Structures Library in Java. It is a collection of Java interfaces and classes that implement fundamental data structures and algorithms, such as: search trees, hash tables, sorting algorithms, and graph traversals. The correlations are presented in Figure 19; the correlation show similar clusters as the ones shown in the above charts.

Correlations

	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	CBAMU	CCM	TCC	LCC	CAMC
LCOM1	1.0000	0.9923	0.8629	0.4694	0.3529	-0.2963	-0.2846	-0.2749	-0.2576	-0.3058
LCOM2	0.9923	1.0000	0.8851	0.5006	0.3442	-0.2944	-0.2905	-0.2876	-0.2750	-0.2895
LCOM3	0.8629	0.8851	1.0000	0.7850	0.5393	-0.5303	-0.5305	-0.5404	-0.5376	-0.3567
LCOM4	0.4694	0.5006	0.7850	1.0000	0.5923	-0.5964	-0.6639	-0.6181	-0.6235	-0.3531
LCOM5	0.3529	0.3442	0.5393	0.5923	1.0000	-0.5649	-0.5979	-0.5683	-0.5379	-0.5323
CBAMU	-0.2963	-0.2944	-0.5303	-0.5964	-0.5649	1.0000	0.9711	0.9691	0.9513	0.4324
CCM	-0.2846	-0.2905	-0.5305	-0.6639	-0.5979	0.9711	1.0000	0.9677	0.9534	0.4603
TCC	-0.2749	-0.2876	-0.5404	-0.6181	-0.5683	0.9691	0.9677	1.0000	0.9922	0.3824
LCC	-0.2576	-0.2750	-0.5376	-0.6235	-0.5379	0.9513	0.9534	0.9922	1.0000	0.3503
CAMC	-0.3058	-0.2895	-0.3567	-0.3531	-0.5323	0.4324	0.4603	0.3824	0.3503	1.0000

Figure 19: Cohesion metrics correlations

The connection types used by each of the cohesion metrics studied in this research are presented in Table 26. The first ten metrics in this table are empirically investigated.

Table 26: Cohesion metrics Vs connection types

Metric	MAR	DMMR	IMMR	DAS	IAS	DDI	DMI	PPI	MIBAT
LCOM1				X					
LCOM2				X					
LCOM3				X					
LCOM4		X		X					
LCOM5	X								
CBAMU	X	X	X						
CCM		X		X					
TCC				X					
LCC				X	X				
CAMC	X							X	
Co		X		X					
ECCM		X		X					
OCC		X		X					
PCC		X		X					X
RCI	X					X	X		
CBMC	X	X	X						

OBSERVATIONS

- All the LCOM metrics show negative correlations with the remaining metrics because they are inverse metrics as discussed in Chapter 2. Out of all the LCOMs metrics, only LCOM4 show a bit of significant correlation with CCM, TCC, and LCC.
- LCOM1, LCOM2, & LCOM3 are correlated significantly while LCOM4 is slightly correlated with these three metrics. LCOM1, LCOM2, & LCOM3 are correlated because they all use the DAS connection type; LCOM4 is not significantly correlated with these metrics because it uses DMMR in addition to the DAS connection type.
- CCM, TCC, LCC and CBAMU are correlated significantly. The fact that CCM, TCC and LCC use the same connection type (i.e. the DAS connection type) we expect them to be significantly correlated. Moreover, these three metrics are similar in the way they compute the cohesion of a class. CBAMU show significant

correlation with CCM because they share the DMMR connection type. Though, CBAMU does not share any connection type with LCC, the two metrics are similar in that both metrics consider indirect interactions in the computation of the cohesion of a class; while CBAMU uses IMMR the LCC metric uses IAS. More so, this is an indication that there are different orthogonal ways of capturing class cohesion.

- CBAMU and LCOM5 are slightly correlated. They are slightly correlated because they both use the MAR connection type but, in addition to this, CBAMU uses DMMR and IMMR connection types; this justifies why the two metrics are not significantly correlated.
- LCOM4 is slightly correlated to CCM, TCC & LCC because they all share the DAS connection type. It is worth mentioning that LCOM4 and CCM both use the DMMR connection type in addition to the DAS connection type so we expect the correlation between LCOM4 and CCM to be higher than the correlation between LCOM4 and TCC (or LCC). This is exactly what the result shows as we can see in Figure 19.
- From Figure 19 and Table 26, we can see that the most highly used connection type is DAS and, from the patterns shown in the charts, we may conclude that DAS is the most effective connection type.
- Whenever the DMMR connection type is used by a metric, it is always used in addition to another connection type.

Therefore, we can conclude from the above observations that:

1. Some of the metrics are related and the following clusters can clearly be seen

a. LCOM1, LCOM2 and LCOM3
b. LCOM3 and LCOM4 (the correlation between these metrics is weaker than the one in (a)).
c. CBAMU, CCM, TCC and LCC
d. LCOM4, CCM, TCC and LCC (here LCOM4 is not strongly correlated to the remaining three metrics).

2. The fact that metrics form different clusters (i.e. clusters a, b & c) show different results is an indication of inconsistencies among cohesion metrics.
3. Furthermore, we may conclude that there are four mechanisms that may be used to in order to measure the cohesion of a module. These are:
    a. Direct Method to Method interaction via method invocation
    b. Indirect Method to Method interaction via attributing sharing or indirect relationship based on method invocation.
    c. Direct Method to Attribute interaction
    d. Indirect Method to Attribute interaction

These mechanisms are presented in Table 27 along side the cohesion metrics.

Table 27: Mechanisms for measuring OO cohesion metrics

Metric	Interaction Type		Interaction Mode		Method Interaction	
	M → A	M → M	Direct	Indirect	M/Invocation	A /Sharing
LCOM1		X		X		X
LCOM2		X		X		X
LCOM3		X		X		X
LCOM4		X	X	X	X	X
LCOM5	X		X			
CBAMU	X	X	X	X	X	
CCM		X	X		X	X
TCC		X	X			X
LCC		X	X	X		X
CAMC	X		X			
Co		X	X		X	X
ECCM		X	X		X	X
OCC		X	X	X		X
PCC		X	X	X		X
RCI	X		X	X		
CBMC	X	X	X	X	X	

From Table 27 we can draw the following conclusions:

1. There are two ways via which the interactions among methods can be captured: (1) Method Invocation (2) Attribute sharing.
2. The most effective Interaction Type is **M** → **M**. This may be considered as the best way to capture the cohesion of a class because methods play a better role (than attributes) in determining what the functionality of a class is.
3. It is interesting to note that with the mechanisms presented in Table 27, the cohesion metrics correlations can easily be explained. For instance:
    a. LCOM1, LCOM2 and LCOM3 show high correlation because they share the same interaction type/mode. LCOM4 does not show high correlation with these three metrics because it uses both the direct and indirect *interaction mode* while the remaining three metrics use only the indirection *interaction mode*.
    b. CCM, TCC, LCC and CBAMU are significantly correlated because they all use the stronger *interaction type* (**M** → **M**) and the stronger *interaction mode* (Direct).
    c. CBAMU and LCOM5 are not significantly correlated because although they use the same *interaction type* and *interaction mode*, CBAMU use the stronger *interaction type* while LCOM5 does not.
    d. From Table 27, we can see that CAMC and LCOM5 use the same type and mode of interaction. However, the coefficient of correlation between these metrics is 0.5323; the lack of significant correlation may be due to the fact that CAMC is a design level metric while LCOM5 is a code level metric.

The intersections among the connection types based on the metrics that use them are presented in Figure 20. In order words, the sets in the figure represent the metrics.

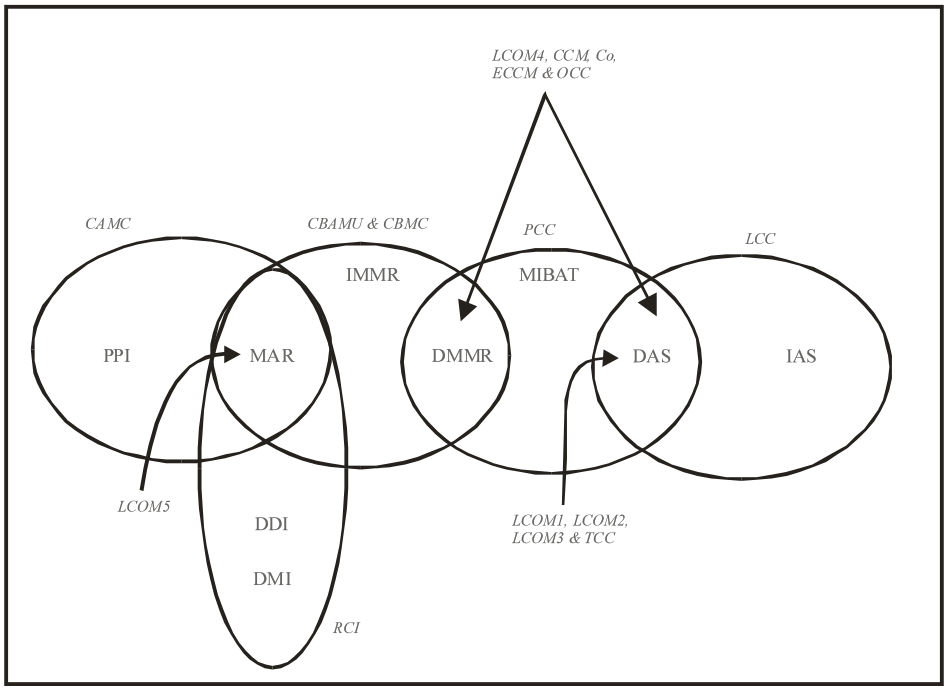


Figure 20: Intersections among the Connection Types

From the figure above, we expect CBAMU to be more correlated to LCOM5 than with LCC but this is not the case. It has already been discussed why LCOM5 and CBAMU show partial correlation. The fact that LCC is correlated to CBAMU but share no connection type in common is indication that there are different ways through which the aspects of class cohesion can be measured and these ways are orthogonal. These ways are: (1) Measuring class cohesion by capturing direct attribute to method relationship or direct/indirect method to method relationship (2) Measuring class cohesion by capturing the indirect method to method relationship via attribute sharing. These two ways are termed as *"Interaction Mode"* in Table 27

Since we don't have 100% correlation between metrics of type (1) and metrics of type (2), it means that either none or at least one of the orthogonal ways for measuring class cohesion does not capture all the dimensions (aspects) of cohesion. To effectively measure class cohesion, we need to have metrics that can capture all the dimensions (aspects) of cohesion. The main challenge here is how to determine all the dimensions of class cohesion. This may be achieved by determining which of the orthogonal set significantly correlates to an external quality attribute (e.g. fault).

CHAPTER 5

5 NUMBER OF DEFECT PREDICTION MODELS

5.1 Introduction

This chapter presents defect prediction models, models that will help us interpret the values obtained from software design metrics. The chapter is organized as follows: Section 5.2 discusses the different approaches for building software prediction models. Section 5.3 presents the experimental goals and the hypothesis, and Section 5.4 presents related work. The description of study and analysis of results are discussed in sections 5.5 and 5.6, respectively.

5.2 Approaches for Building Software Prediction Model

The different approaches through which software prediction models are built can be classified into four different classes: machine learning, probabilistic approaches, statistical approaches and mixed methods; as shown in Figure 21.

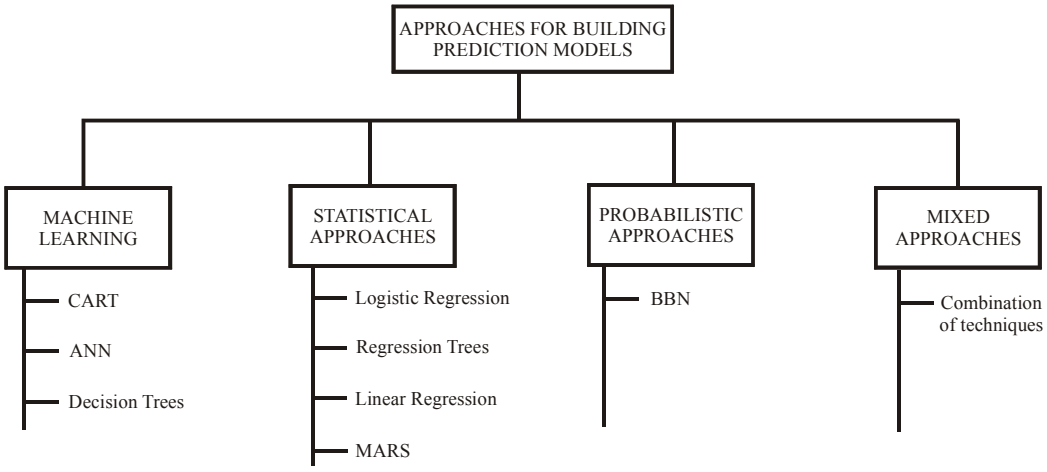


Figure 21: Approaches for Building Prediction Models

Quite a number of researchers have investigated the use of machine learning techniques for building software prediction models. Porter and Selby studied the use of decision trees in building prediction models [68][71] and Khoshgoftaar et al. applied artificial neural networks in building prediction models [57][58]. Statistical approaches have been investigated by Abreu et al. who applied linear regression [1], by El Emam et al. who applied logistic regression [34], and by Briand et al. who applied Logistic regression and MARS (Multivariate Adaptive Regression Splines) [21]. Probabilistic approaches have been exploited by Fenton et al. who highly criticized the complete reliance on historical data of software projects when building prediction models, he proposed the use of Bayesian Belief Networks [39]. Other researchers combine different techniques; for instance Morasca and Ruhe worked by combining rough set analysis and logistic regression [64]. In this work, prediction models are built using Regression analysis and Artificial Neural Network and the performance of the models are compared based on coefficient of determination.

When building prediction models, the choice of the prediction technique may affect the result. The choice of the prediction techniques used in this research is due to the following reasons:

- Multiple Linear Regression (MLR) is used because the data set used is not large enough to investigate non-linear interactions. For accurate non-linear prediction models, large data set is required [22]. In addition, MLR is better than some techniques like CART-LS (Classification and Regression Trees-Least Square) and S-PLUS regression trees [59].
- However, the performance of the MLR model was not very good so we built a nonlinear model using regression analysis, which gave better results.

- The Artificial Neural Network (ANN) technique is claimed to be simple and accurate [58]. A third model is build using this technique and its performance is compared with the performance of the regression models.

5.3 Experimental Goal

The goal of this experiment is to determine the predictive power of cohesion metrics amidst other object oriented software metrics with respect to the number of defects in a class. In other words, our external software quality attribute of interest is the number of defects in a class. The number of defects in a system will give an idea of how reliable the system is; a system with high number of defects will not be reliable because the presence of the defects may lead to undesirable behavior of the system. On the other hand, a system with few defects is expected to be more reliable than a system with high number of defects. Our aim here is to achieve the following:

1. To determine which type of measure (coupling, cohesion, complexity etc) is significantly correlated to the number of defects of a class.
2. To compare the performance of the models built using Artificial Neural Network and Regression Analysis using the same data.

5.3.1 Hypothesis

In this study, we want to test a number of hypotheses; basically we want to find out which measure (coupling, complexity, cohesion or inheritance) has significant impact on the number of defects in a class. The following hypotheses will be tested in this study:

1. A class with high coupling measure is likely to have high number of defects. In other words, coupling measure is significantly correlated to the number of defects in a class.

2. A class with low cohesion is more likely to have high number of defects than a class with high cohesion
3. A class with high measure of complexity is more likely to have higher number of defects than a class of lower complexity.
4. A class with high inheritance measure (e.g. a class with many descendents or a class situated deeper in the inheritance hierarchy) is more likely to have high number of defects.

5.4 Related Work

One of the earliest defect prediction studies was in 1971 by Akiyama [1], which was based on a system developed at Fujitsu, Japan. The study showed that linear models of some simple metrics provide reasonable estimate for the total number of defects D (the dependent variable) which is actually defined as the sum of the defects found during testing and the defects found during two months after release. One of Akiyama's correlation involving lines of code is shown in the following equation.

$$D = 4.86 + 0.018L$$

Another early study was in 1974 by Ferdinand, [41], who argued that the expected number of defects increases with the number *n* of code segments; a code segment is a sequence of executable statements which, once entered, must all be executed. Specifically the theory asserts that for smaller numbers of segments, the number of defects is proportional to a power of *n*; for larger numbers of segments, the number of defects increases as a constant to the power *n*.

In 1975, Halstead [44], proposed a number of size metrics, which have been interpreted as "complexity" metrics, and used these as predictors of program defects.

Most notably, Halstead asserted that the number of defects D in a program P is predicted by the following equation.

$$D = V/3000$$

Where V is the volume metric, which like all the Halstead metrics is defined in terms of the number of unique operators and unique operands in P. The divisor 3000, represents the mean number of mental discriminations between decision made by the programmer. Each such decision possibly results in error and thereby a residual defect.

The availability of artificial neural network programming tools has attracted the attention of software engineers. Software researchers now use artificial neural network in several software related applications. Karunanithi et al. [55] explored the applicability of neural network models for dynamic software reliability growth prediction, and demonstrated that neural network models exhibit better predictive quality than some analytic models. Khoshgoftaar et al. proposed a neural network approach for predicting the number of faults in program modules [57]. In 1994, Khoshgoftaar et al. introduced a neural network approach for detecting high-risk modules, and compared the results of their approach with that of discriminant analytic approach. They concluded that neural network gave better performance [58].

5.5 Description of Study

In this subsection, we present a detailed description of the Number of defects prediction model build using Artificial Neural Network.

5.5.1 System

The system used for this experiment is a C++ project (KC1) downloaded from NASA IV & V MDP (Metric Data Program) repository. Owing to the fact that the source code of the project is not made available on the repository, we solely rely on the information provided on the site as such we only used the class level metrics provided in the documentation of the project.

The NASA IV&V Metrics Data Program project is being developed by Galaxy Global Corporation, Inc. The primary objective of the Metrics Data Program is to collect, validate, organize, store and deliver software metrics data. The project detail is as follows:

- Number of classes : 145 classes
- Classes with defects : 60 classes
- Number of modules : 2107 modules
- Modules with defects : 293 modules

5.5.2 Dependent Variable

We want to evaluate the predictive power of the class level metrics provided in the NASA KC1 project with respect to the number of defects in a class. More precisely, we want to determine the number of defects in a class by considering the values of the class level metrics. Hence, the dependant variable is number of defects in a class.

The defects in the project are not associated to classes, they are linked to Modules (a term applied to the lowest level functional unit which metrics can be applied e.g. functions, modules, subroutines). However, since we can determine the number of

modules in a class and defects are related to modules, we can easily determine the number of defects in each class.

5.5.3 Independent variables

The independent variables are the class level metrics captured in the KC1 NASA project. We limit our analysis to these metrics because the source code of this project is not made available in the repository, thus more metrics cannot be calculated. A total of ten different class-level software metrics were computed for the classes in this project. They are:

1. PERCENT_PUB_DAT (PPD)
2. ACCESS_TO_PUB_DATA (ATPD)
3. COUPLING_BETWEEN_OBJECTS (CBO)
4. DEPTH
5. LACK_OF_COHESION_OF_METHODS (LCOM)
6. NUM_OF_CHILDREN (NOC)
7. DEP_ON_CHILD (DOC)
8. FAN_IN
9. RESPONSE_FOR_CLASS (RFC)
10. WEIGHTED_METHODS_PER_CLASS (WMPC)

The above metrics are all considered in building the prediction model. We start by classifying them into complexity, coupling, cohesion and inheritance measures as shown in Table 28.

Table 28: Metrics Classification

#	Complexity	Coupling	Cohesion	Inheritance
1	FAN_IN	CBO	LCOM	DEPTH
2	WMPC			NOC
3	PPD			DOC
4	ATPD			
5	RFC			

5.6 Analysis of Results

In this section we present the analysis of the results obtained from the experiments, we start by discussing the descriptive statistics of the respective parameters we consider to building the prediction model.

5.6.1 Descriptive Statistics

Table 29 presents the descriptive statistics for the 80% of the project, which is used as training data for building the prediction models. We used 80% of the data for training because the size of the project is not large enough; to properly train the model we need to have large data set. Rows, "Max", "P75", "Median", "P25", "Min", "Mean", and "Variance" state for each metric the maximum value, 75 % percentile, median, 25 % percentile, minimum, mean and variance respectively. For this project, the inheritance metrics (DEPTH, DOC, and NOC) have low mean and variance values; this is an indication that the use of inheritance is sparse.

Table 29: Descriptive statistics I

	Max	P75	Median	P25	Min	Mean	Variance
PPD	100	0	0	0	0	14.4	1058
ATPD	0	0	0	0	0	0	0
CBO	24	14	8	3	0	8.32	40.7
DEPTH	7	2	2	1	1	2	1.5833
LCOM	100	96	84	58	0	68.72	1361
NOC	5	0	0	0	0	0.21	0.49
DOC	1	0	0	0	0	0.01	0.01
FAN_IN	3	1	1	0	0	0.6345	0.4835
RFC	222	44	28	10	0	34.4	1311
WMPC	100	22	12	8	0	17.421	304.47
DEFECT	101	4	0	0	0	4.613793	117.9054

Of these metrics, ATPD (Access to public data) was not considered for the analysis because it returns zero for all the classes in the training set. DOC (Dependent on Child) was also not considered for further analysis because almost all its entries are zeros. Table 30 shows the descriptive statistics of the remaining measures.

Table 30: Descriptive statistics II

	Max	P75	Median	P25	Min	Mean	Variance
PPD	100	0	0	0	0	14.4	1058
CBO	24	14	8	3	0	8.32	40.7
DEPTH	7	2	2	1	1	2	1.5833
LCOM	100	96	84	58	0	68.72	1361
NOC	5	0	0	0	0	0.21	0.49
FAN_IN	3	1	1	0	0	0.6345	0.4835
RFC	222	44	28	10	0	34.4	1311
WMPC	100	22	12	8	0	17.421	304.47
DEFECTS	101	4	0	0	0	4.613793	117.9054

So we are left with a total of eight class-level metrics, which will serve as the independent variables or rather the input parameters when building the prediction model using Artificial Neural Network. From Table 30, we make the following observations:

- Both DEPTH and NOC have low variance; this is an indication of low variation of values in the sample space. Hence, both measures will not be very good candidates for prediction. However, they may help in building prediction models.
- More than 75% of the classes don't have any child which is an indication that most of the classes are leaf classes. Furthermore, less than 25% of the classes have a depth of more than 2. Hence, the overall use of inheritance in this project is low.
- More than 75% of the data is not public (from the distribution of the PPD metric); this is an indication that coupling measure is not very high as can be seen from CBO whose mean value is relatively low.

- The low variance of FAN_IN is an indication of low dependence which supports the above argument that the coupling across modules is not very high.

5.6.2 Artificial Neural Network (ANN) Prediction Model

A prediction model, the following details, was built using ANN.

- Neural Model : Multilayer perceptron
- Hidden Layers : 3 layers
- Transfer function : TanhAxon
- Number of epochs : 5000 epochs

An MLP is a network of simple neurons called perceptrons. The basic concept of a single perceptron was introduced by Rosenblatt in 1958. The perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then possibly putting the output through some nonlinear activation function. Mathematically this can be written as [51].

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi\left(W^T X + b\right)$$

Where w denotes the vector of weights, x is the vector of inputs, b is the bias and φ is the activation function. A signal-flow graph of this operation is shown in Figure 22.

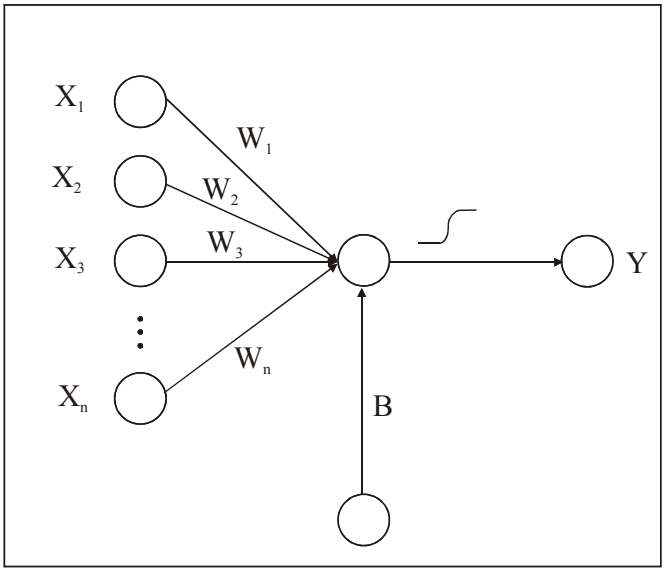


Figure 22: Perceptron

A single perceptron is not very useful because of its limited mapping ability. No matter what activation function is used, the perceptron is only able to represent an oriented ridge-like function. The perceptrons can, however, be used as building blocks of a larger, much more practical structure. A typical multilayer perceptron (MLP) network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-by-layer. The signal-flow of such a network with one hidden layer is shown in Figure 23 [51].

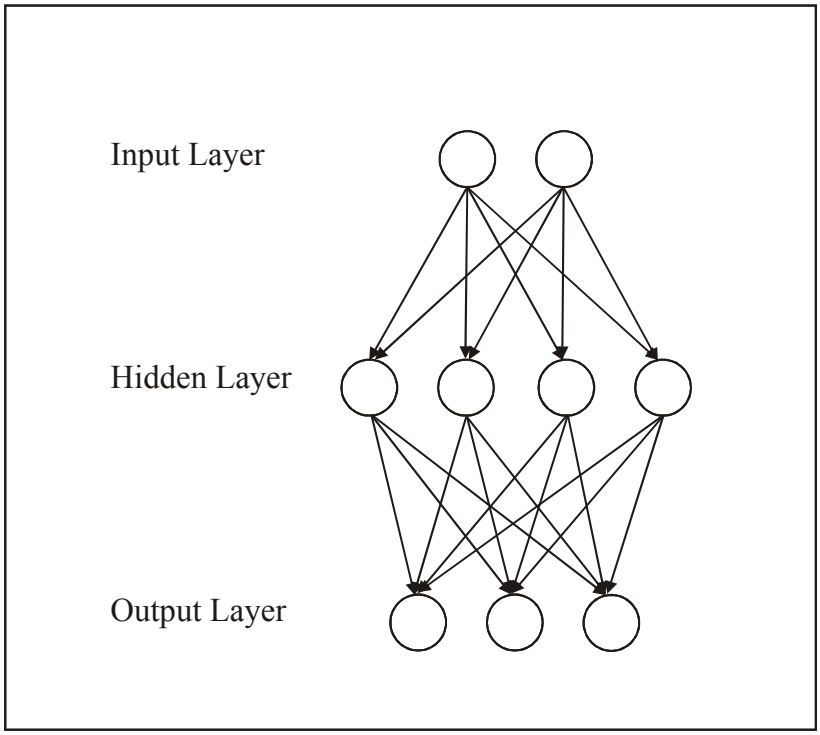


Figure 23: Signal-flow with hidden layer

Nowadays, and especially in multilayer networks, the activation function is often chosen to be the logistic sigmoid $1/\left(1+e^{-x}\right)$ or the hyperbolic tangent $\tanh(x)$. They are related by $\frac{\tanh(x)+1}{2}=\frac{1}{1+e^{-2x}}$. These functions are used because they are mathematically convenient and are close to linear near origin while saturating rather quickly when getting away from the origin. This allows MLP networks to model well both strongly and mildly nonlinear mappings [51].

Figure 24 shows the Artificial Neural Network that was generated after series of trial and error in order to optimize the performance of the network.

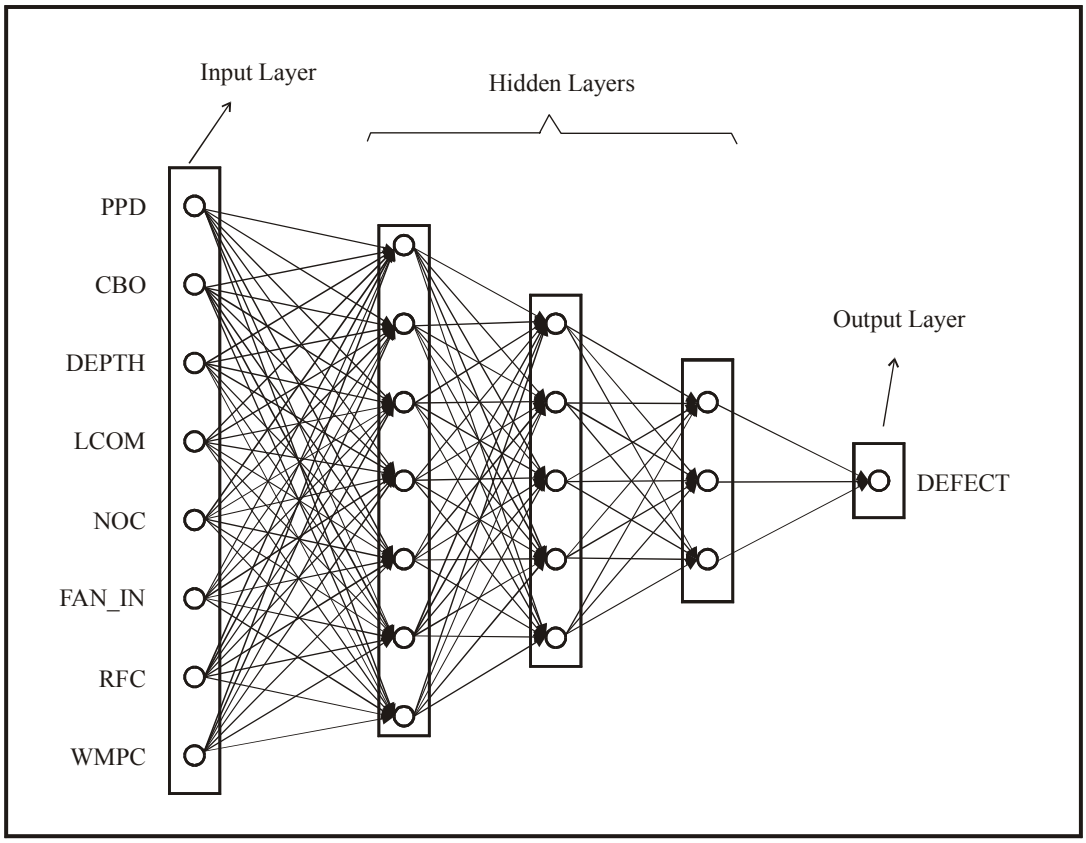


Figure 24: The ANN Defects Prediction Model 1

5.6.3 Regression Models

In this subsection we described the two prediction models that were built using regression technique. These models are: Multiple Linear Regression model and nonlinear regression model. We begin by determining the correlation among the parameters.

5.6.3.1 Correlations

As explained at the beginning of this chapter, one of the goals of this research work is to determine which measures are significantly correlated to the number of defects in a class. Table 31 shows the correlation that exist among the different variables and most importantly the correlation of each of the independent variable on the dependant variable (number of defects).

Table 31: Measures Correlations

Variable	Correlations (NASAPROJECT.sta) Marked correlations are significant at p < .05000 N=116 (Casewise deletion of missing data)								
	PPD	CBO	DEPTH	LCOM	NOC	FAN_IN	RFC	WMPC	DEFECTS
PPD	1.00	0.12	-0.06	0.25	0.11	0.00	0.06	0.12	0.14
CBO	0.12	1.00	0.50	0.35	-0.01	0.49	0.47	0.27	0.34
DEPTH	-0.06	0.50	1.00	0.37	-0.01	0.72	0.63	0.15	-0.00
LCOM	0.25	0.35	0.37	1.00	-0.03	0.34	0.39	0.31	0.14
NOC	0.11	-0.01	-0.01	-0.03	1.00	-0.07	-0.05	0.00	-0.10
FAN_IN	0.00	0.49	0.72	0.34	-0.07	1.00	0.52	0.10	0.07
RFC	0.06	0.47	0.63	0.39	-0.05	0.52	1.00	0.69	0.28
WMPC	0.12	0.27	0.15	0.31	0.00	0.10	0.69	1.00	0.47
DEFECTS	0.14	0.34	-0.00	0.14	-0.10	0.07	0.28	0.47	1.00

From Table 31 we can see that apart from DEPTH (which does not have any correlation with the dependant variable), the other inheritance measure (i.e. NOC) has a negative correlation with the dependant variable. One interpretation for this may be that classes with high inheritance measure (because such classes are more complex to deal with) have been carefully designed and implemented possibly by more experienced programmers and therefore less defects are discovered in such classes. WMPC, CBO, and RFC have high positive correlation with the number of defects. While PPD, LCOM and FAN_IN have low positive correlation with the number of defects.

5.6.3.2 Stepwise Correlation

To determine those measures that are significantly correlated to the number of defects in other words to determine those parameters that can be considered in building a prediction model at a confidence limit of 0.05, we run a stepwise regression. The result of the last step (step 8, after the last input parameter) is presented below.

```
Step 8   Variable FAN_IN Entered    R-square = 0.33045914   C(p) =  9.00000000

               DF         Sum of Squares      Mean Square          F   Prob>F

Regression      8          5242.75412769     655.34426596       6.60   0.0001
Error         107         10622.30621714      99.27388988
Total         115         15865.06034483

               Parameter        Standard          Type II
Variable        Estimate           Error   Sum of Squares          F   Prob>F

INTERCEP      -0.60840605      2.51544224       5.80755578       0.06   0.8093
PPD            0.02173577      0.03062470      50.00824071       0.50   0.4794
CBO            0.62856421      0.18572793    1137.05088067      11.45   0.0010
DEPTH         -1.92505065      1.34690377     202.78985571       2.04   0.1558
LCOM          -0.01836089      0.03022309      36.63908023       0.37   0.5448
NOC           -1.74195553      1.26765582     187.45907042       1.89   0.1723
FAN_IN         1.18694648      2.00371776      34.83567407       0.35   0.5549
RFC           -0.03797026      0.05558022      46.33205791       0.47   0.4960
WMPC           0.31406710      0.08360890    1400.79614054      14.11   0.0003

Bounds on condition number:     4.349991,     141.6002
------------------------------------------------------------------------------
```

From the result above, we can see that only two parameters show statistical significance; these are CBO and WMPC. Hence, using these two parameters, the linear regression model is:

	Regression Summary for Dependent Variable: DEFECT (TRAINING.sta) R= .51935031 R²= .26972474 Adjusted R²= .25679951 F(2,113)=20.868 p<.00000 Std.Error of estimate: 10.126					
N=116	Beta	Std.Err. of Beta	B	Std.Err. of B	t(113)	p-level
Intercept			-3.58368	1.795957	-1.99542	0.048404
CBO	0.223306	0.083575	0.42436	0.158820	2.67194	0.008656
WMPC	0.411797	0.083575	0.26172	0.053117	4.92730	0.000003

The model can be expressed by the following formula:

$$DEFECT = 0.42\ (CBO) + 0.26\ (WMPC) - 3.58$$

The performance of this model is nothing to write home about. On plotting the scatter diagrams of the parameters, we have noticed that there is an outlier in the plots as shown in the following Figure 25. An outlier is a data point which is located in an empty part of the sample space; the inclusion or exclusion of outliers can have a large influence on the result of the prediction model.

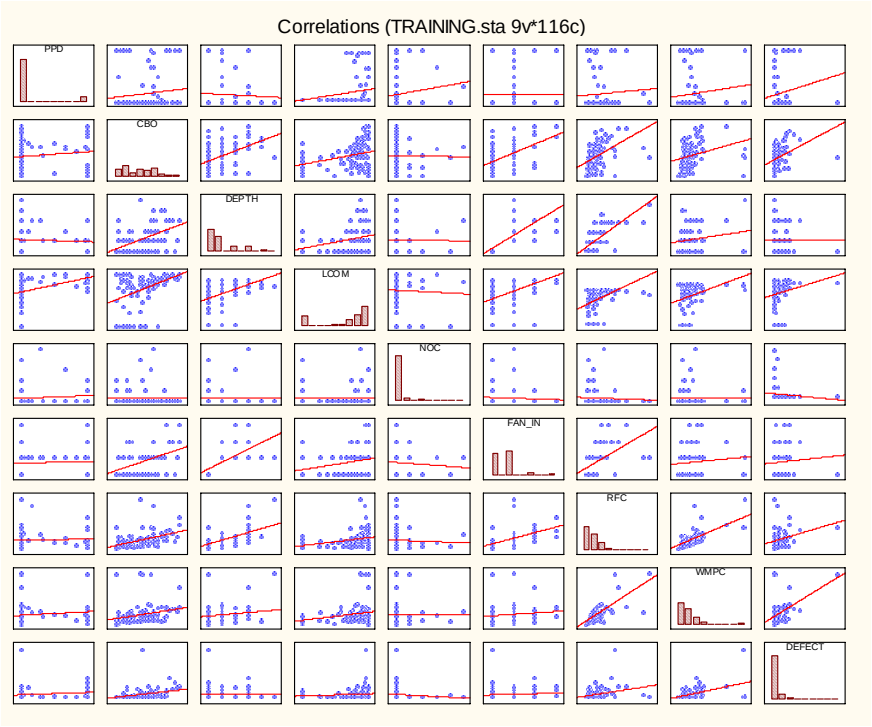


Figure 25: Scatter Diagrams

The same experiment is repeated without the outlier; slightly better results were obtained from the new model. The model without the outlier is given below:

	Regression Summary for Dependent Variable: DEFECT (TRAIN.sta) R= .52610654 R²= .27678809 Adjusted R²= .26398788 F(2,113)=21.624 p<.00000 Std.Error of estimate: 6.4971					
N=116	Beta	Std.Err. of Beta	B	Std.Err. of B	t(113)	p-level
Intercept			-2.14548	1.156143	-1.85572	0.066099
CBO	0.467585	0.083712	0.57295	0.102575	5.58567	0.000000
WMPC	0.140006	0.083712	0.06273	0.037507	1.67248	0.097197

Mathematically, the model can be expressed as:

DEFECT = 0.57 (CBO) + 0.06 (WMPC) - 2.15

5.6.4 Prediction Model Evaluation (Goodness of fit)

In order to compare the performance of the models built using Regression analysis and the one built using Artificial Neural Network, we built two more prediction models. The first is a nonlinear regression model and the second an ANN model both using CBO and WMPC, which the same input parameters used for the linear regression model. The statistics of the nonlinear model is shown in Table 32.

Table 32: Nonlinear Regression Model

Model	pr1+pr2*X1^1+pr3*X2^1+pr4*X1^2+pr5*X2^2+pr6*X1^3+pr7*X2^3
Equation	1.097 + 1.231*X1^1 - 0.518 * X2^1-0.152 *X1^2 + 2.878E-02 *X2^2 + 5.958E-03*X1^3-2.401E-04*X2^3
R^2	0.428

Details of the new model built using ANN are:

- Neural Model : Multilayer perceptron
- Hidden Layers : 2 layers
- Transfer function : TanhAxon
- Number of epochs : 5000 epochs

The model is shown in Figure 26.

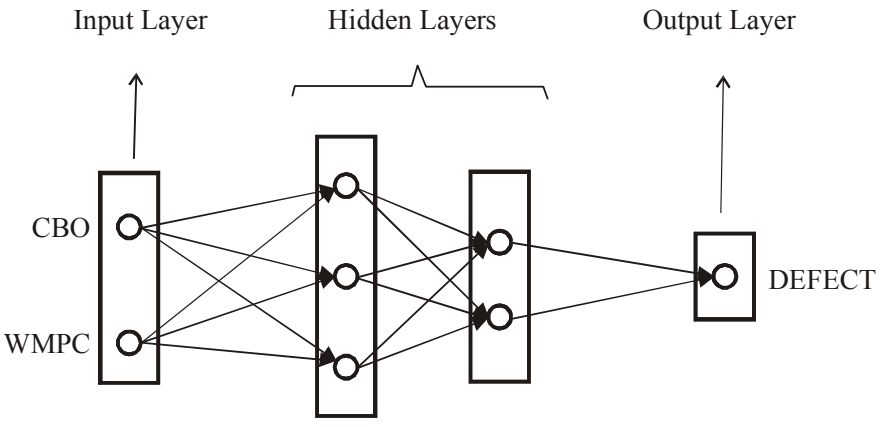


Figure 26: The ANN Defects Prediction Model 2

To evaluate the model's goodness of fit, we apply the prediction models to the remaining 20% of the same NASA project that we used in building the prediction models. The coefficient of determination of the three different models is shown in Table 33. The Table shows that the model built using ANN has better performance than models built using regression analysis.

Table 33: Comparing the Performance of the Prediction Models

Approach	**R^2**
ANN	0.588
Nonlinear	0.482
Linear	0.280

From the results of the above experiment we can draw the following conclusions:

1. Only coupling and complexity metrics are significantly correlated to the number of defects in a class.
2. Cohesion is not correlated to the number of defects in a class, one reason for this might be that the cohesion metric used (LCOM) does not accurately capture the cohesion of a class.
3. Inheritance measures are not correlated to the number of defects in a class. As explained at the beginning of this chapter the reason might be because such classes were handled by experts as such few defects were reported.

CHAPTER 6

6 CONCLUSION

In this chapter we present a summary of the contribution of this thesis, outline the limitation of the work and provide suggestions on how it can be improved in the future.

6.1 Summary and Contributions of the Thesis

In this research we conducted a literature survey of object-oriented cohesion metrics. Most researchers based their definition of cohesion on the assumption that the more the interaction among the components of a class, the higher the cohesion of the class; this interaction is captured by looking at method-method and method-attribute accesses. As a matter of fact, Most of the approaches are based on attribute usage or method invocations and the cohesion metrics studied capture cohesion at the class level. We identified some problems in the definition of some of the existing object-oriented cohesion metrics.

We also proposed eleven classification criteria, which we used in critically analyzing all the cohesion metrics we found in the literature. The classification criteria are of two types: Factors and characteristics; factors identify the things that may affect the cohesiveness of a module. The more factors a metric considers in its definition the more effective it is likely to be in computing the cohesion of modules. Characteristics specify the characteristics of the cohesion metric i.e. the features of the metric. See chapter 3, for details of the classification criteria. At the end of our analysis we found that there are some inaccuracies in the definition of some of the cohesion metrics. For instance, LCOM1 does not accurately capture the cohesion of a class, LCOM2 returns

zero for classes that are intuitively of different cohesion. LCOM4 always gives the value 1 when the number of the connected component is one irrespective of the access density. LCC and TCC give result that is contrary to intuition especially when there is huge number of methods in the class. In fact, of all the cohesion metrics that are critically analyzed, none seem to accurately capture the cohesion of a class may be the concept is yet to be fully understood in the object oriented paradigm. As discussed in Chapter 4, there are inconsistencies in the existing cohesion metrics.

We determine the relationships that exist among the connection types used by different object-oriented cohesion metrics. At the end of our analysis, we came to the conclusion that there are different ways through which the aspects of class cohesion can be measured and these ways are orthogonal. These ways are: (1) Measuring class cohesion by capturing direct attribute to method relationship or direct/indirect method to method relationship (2) Measuring class cohesion by capturing the indirect method to method relationship via attribute sharing.

We proposed a cohesion metric (CBAMU); we do not claim that our metric is the best nor do we claim that the metric accurately captures the cohesion of a class but this is simply our intuition of class cohesion. The metric uses both method and attribute usage in its definition see Chapter 4 for details.

Number of defects prediction models were built using Artificial Neural Network and regression analysis; results show that model built using ANN performs better.

6.2 Limitations and Future Work

In this section, we discuss the limitations of this work and give indications of how it can be improved in the future.

1. In Chapter 5, we built defect prediction models using ANN and regression analysis. However, the models built are solely dependent on the NASA KC1 project data, Future work requires that similar experiments be rigorously conducted using more well documented projects.
2. In addition, the models may not be very accurate because uncertainty factors were not considered in the cause of building the models. Uncertainties in prediction may arise from the input parameters due to, but not limited to, the following:
    a. Researchers cannot exhaustively incorporate all the possible factors that may affect the accuracy of prediction models because some of the factors may not be known at the point of building the models.
    b. Other, researchers may deliberately ignore some factors even though they know that such factors will affect the accuracy of the models due to the fact that such factors may complicate the model; probably to enable them meet datelines. Thus, some factors are deliberately left out of the analysis.

    NB:

    In order to account for uncertainty in an output function of any sort we need to consider the uncertainties in the input variables as well as any uncertainty surrounding the way the model is built. Further work is needed to address the issues of uncertainty in building similar models.

3. In section 4.5.1, we hypothesized that: "*A project with low average class cohesion is likely to have high number of defects than a class with high average class cohesion"*. Conducting rigorous experiments in order to show that this assertion truly holds may be pursued as future work.
4. The cohesion studied in this work measure cohesion at class level, measuring cohesion at a higher level of abstraction is a promising area for research.
5. We identified two orthogonal ways through which class cohesion can be measured: (1) Method-Method accesses (2) Method-Method accesses. Determining which of these ways is strongly correlated to an external software quality attribute may be pursued as future work.

REFERENCES

[1] Abreu R., Goulao M., Esteves R., "Toward the Design Quality Evaluation of Object-Oriented Software Systems", 5th International Conference on Software Quality, Austin, Texas, USA, October 1996.

[2] Akiyama F., "An Example of Software System Debugging", Information Processing, vol. 71, pp. 353-379, 1971.

[3] Alkadi G., Alkadi I., "Application of a Revised DIT Metric to Redesign an OO Design", Eth Zurich, Chair of Software Engineering ©JOT, Vol. 2, No. 3, May-June 2003.

[4] Aman H., Yamasaki K., Yamada H. Noda M., "A Proposal of Class Cohesion Metrics Using Sizes of Cohesive Parts", Knowledge-based Software Engineering, T. welzer et al.(Eds.), pp102-107, IOS Press, Sept. 2002.

[5] Badri L. and Badri M.,"A Proposal of a New Class Cohesion Criterion: An Empirical Study", Journal of Object Technology, Vol. 3, No. 4, April 2004.

[6] Bansiya J., Etzkorn L., Davis C., and Li W., "A Class Cohesion Metric for Object-Oriented Designs", Jornal of Object-Oriented Programming (January), pages 47-52, 1999.

[7] Basili V. and Rombach H., "The TAME Project: Towards Improvement-Oriented Software Environments", IEEE Trans. Software Eng., 14(6), June, 1988.

[8] Basili V. R., Briand L. C., and Melo W. L., "A Validation of object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, 22(10), 751-761 (1996).

[9] Beyer D., Lewerentz C., and Simon F., "Impact of Inheritance on Metrics for Size, Coupling and Cohesion in Object Oriented Systems", in Dumke, Abran (Eds):"New Approaches in Software Measurement", Lecture Notes on Computer Science, pp. 1-17, Springer-Verlag, 2001.

[10] Bieman J. M., Kang B., "Measuring Design-Level Cohesion", IEEE Trans. On Software Engineering, Vol. 24, No. 2, February 1998.

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

[12] Bieman J., Ott L., "Measuring Functional Cohesion", IEEE Transactions on Software Engineering, vol. 20 no 8, August 1994.

[13] Booch G., Rumbaugh J., Jacobson I., "The Unified Modeling Language User Guide", Addison Wasley Longman, Inc., USA, 1998.

[14] Briand L., Morasca S., Basili V., "Assessing Software Maintainability at the end of High-Level Design", IEEE Conference on Software Maintenance (ICSM), 1993, Montreal, Canada

[15] Briand L., Morasca S., Basili V., “Defining and Validating High-Level Design Metrics”, Computer Science Technical Report CS-TR 3301, University of Maryland at College Park, 1994.

[16] Briand L., Morasca S., Basili V., “Property-Based Software Engineering Measurement”, IEEE Transactions on Software Engineering, vol 22 no 1, January 1996.

[17] Briand L., Morrasca S., and Basili V., “Defining and Validating Measures for Object-Based High-Level Design”, IEEE Transactions on Software Engineering, 25(5), (1999), pp. 125-132.

[18] Briand L., Daly J., Wust J., “A Unified Framework for Cohesion Measurement in Object-Oriented Systems”, Technical Report ISERN-97-05. Kaiserlautern: Germany: Fraunhofer Institute for Experimental Software Engineering, 1997.

[19] Briand L., Bunse C., and Daly J., “A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs”, IEEE Transactions on Software Engineering, 27(6), (2001), pp. 513-530.

[20] Briand L. C., Wust J., Daly J. W., and Porter V., “Exploring the Relationships Between Design Measures and Software Quality in Object-Oriented Systems”, ISERN 98-07 – Version 2.

[21] Briand L., Melo W., Wüst J., “Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects”, IEEE Transactions on Software Engineering, 28 (7), 706-720, 2002.

[22] Briand L., Wüst J., “Empirical Studies of Quality Models in Object-Oriented Systems”, Advances in Computers Vol. 59, 97-166, 2002.

[23] Briand L., Wüst J., Lounis H., “Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs”, Empirical Software Engineering: An International Journal, Vol 6, No 1, 11-58, 2001.

[24] Briand L., Wüst J., “The Impact of Design Properties on Development Cost in Object-Oriented Systems”, IEEE Transactions on Software Engineering, 27 (11), 963-986, 2001.

[25] Bunge M., “Treatise on Basic Philosophy: Ontology I: The Furniture of the World”, Boston, Riedel, 1977.

[26] Chae H. S., Kwon Y. R., Bae D., “A cohesion Measure for Object-Oriented Classes”, Software-Practice & Experience, V.30 n.12, p.1405-1431, Oct. 2000.

[27] Cherniavsky, J.C. and Smith, C.H., "On Weyuker's Axioms for Sotware Complexity Measures", IEEE Transactions on Software Engineering, vol. 17, pp. 636 – 638, 1991.

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

[29] Chidamber S., Kemerer C., "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol 20 no6, June 1994.

[30] Counsell S., Mendes E., Swift S., "*Comprehension of Object-Oriented Software Cohesion: the Empiri*cal Quagmire", Proceedings of the 10th International Workshop on Program Comprehension(IWPC'02) 1092-8138/02 © 2002 IEEE.

[31] Demuth B., Hussmann H., Obermaier S., "Experiments With XMI Based Transformations of Software Models", In Workshop on Transformations in UML. 2001.

[32] Denaro G., Pezze M., "An Empirical Evaluation of Fault-Proneness Models", ICSE '02, May 19-25, 2002, Orlando, Florida, USA.

[33] Eder J., Kappel G., Schrefl M., "Coupling and Cohesion in Object-Oriented Sysems", Technical Report of University Klagenfurt, Institute of Computer Science, 1993.

[34] El Emam K., Melo W., Machado J., "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", Journal Of Systems and Software 56, 63-75, 2001.

[35] Elish M. O., "Measuring Inheritance Coupling in Object-Oriented Systems", Master Thesis, Department of Information and Computer Science, KFUPM, Dec. 1999.

[36] Fenton, N.E., "Software Metrics, A rigorous approach", New York: Chapman & Hall, 1991

[37] Fenton E. N., Pfleeger S. L., "Software Metrics – A Rigorous & Practical Approach", PWS Publishing company, Boston, 1997.

[38] Fenton N., Neil M., "Software Metrics: Roadmap", In A. Finkelstein, editor, The Future of Software Engineering. ACM Press, New York, 2000.

[39] Fenton N. E. and Neil M., "A Critique of Software Defect Prediction Models, IEEE Transactions on Software Engineering,SE-25(5), pp. 675-689, Oct. 1999.

[40] Fenton N. E., "Software Measurement: A necessary scientific basis", IEEE Trans. Software Eng., vol. 20, no. 3, March 1994, pp. 199-206.

[41] Ferdinand A. E., "A Theory of System Complexity", Int'l J. General Systems, vol. 1, pp. 19 – 33, 1974.

[42] Glasberg D., El Emam K., Melo W., Madhavji N.,"Validating Object-Oriented Design Metrics on a Commercial Java Application", TR ERB – 1080, NRC, Sep. 2000.

[43] Grose T. J., Doney G. C., Brodsky S. A., *"Mastering XMI",* Wiley & Sons, Inc., New York, USA, 2002.

[44] Halstead M. H., "Elements of Software Science", Elsevier North-Holland, 1975.

[45] Hamilton G., Cattell R., Fisher M., "JDBC Database Access with Java: A Tutorial and Annotated Reference", Addison Wesley Longman, Inc. USA, January 1998.

[46] Harrison T., Counsell S. "Theoretical Validation and Empirical Evaluation of Object-oriented Design Metrics", Declarative Systems and Software Engineering Group. 1998.

[47] Henderson-Sellers B., Constantine L. L., Graham I. M., "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)", Object Oriented Systems 3, (1996) 143-158.

[48] Hermadi I., El-Badawi K., Al-Ghamdi J., "Theoretical Validation of Cohesion Metrics in Object Oriented Systems", International Arab Conference on Information Technology 2002 (ACIT2002), 16-19 Dec 2002, University of Qatar, Doha, Qatar.

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

[50] Hitz M., Montazeri B., "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", IEEE Transaction on Software Engineering, vol 22 no 4, April 1996.

[51] http://www.cis.hut.fi/ahonkela/dippa/node41.html, "Multilayer Perceptrons"

[52] Jarallah A., Wasiq M., and Ahmed M. A., "Principle and Metrics for Cohesion-Based Object-Oriented Component Assessment". Confidential Draft Copy, 2001.

[53] Kabaili H., Keller R., Lustman F.,*"Class Cohesion as Predictor of Changeability: An Empirical Study"*, supported by SPOOL project organized by CSER(Consortium for Software Engineering Research) which is funded by Bell Canada, NSERC(National Sciences and Research Concil of Canada), and NRC (National Research Council of Canada) 2001.

[54] Kang B., Bieman J., "Using Design Coesion to Visulaize, Quantify, and Restructure Software", In Eighth International Conference onSoftware Engineering and Knowledge Engineering (SEKE'96), pages 222-229, Skokie, IL, June 1996. Knowledge Systems Institute.

[55] Karunanithi N., Whitley D., and Malaiya Y. K., "Using Neural Networks in Reliability Prediction", IEEE Software, vol. 9, no. 4, pp. 53-59, July 1992.

[56] Khan S., "Design Level Coupling Metrics for UML Models", Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia, 2001.

[57] Khoshgoftaar T. M., Pandya A. S., and More H. B., "A Neural Network Approach for Predicting Software Development faults," in Proc. Third Int. Symp. Software Reliability Eng., Research Triangle Park, NC, Oct. 1992, pp. 83-89.

[58] Khoshgoftaar T. M., Lanning D. L., and Pandya A. S., "A Comparative Study of Pattern Recognition Techniques for Quality Evaluation of Telecommunications Software" IEEE Journal on Selected Areas in Communications, 12(2):279--291, February 1994.

[59] Khoshgoftaar T. M., Seliya N., "An Empirical Study of Commonly Used Modeling Techniques for Software Fault Prediction", FAU Technical Report TR-CSE-01-32, 2001.

[60] Larman C., "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Desing and the Unified Process", Prentice-Hall, Inc. US, 2002.

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

[62] Mens T., Lanza M., *"A Graph-Based Metamodel for Object-Oriented Software Metrics"*, Electronic Notes in Theoretical Computer Science 72 No. 2(2002).

[63] Mens T., Demeyer S., *"Evolution Metrics"*, IWPSE 2001, Vienna, Austria 2001 ACM.

[64] Morasca S., Ruhe G., "A Hybrid Approach to Analyze Empirical Software Engineering Data and its Application to Predict Module Fault-Proneness in Maintenance", The Journal of Systems and Software 53(3): 225-237, Sept. 2000.

[65] Ott L., Bieman J., Kang B., Mehra B., "Developing Measures of Class Cohesion for Object-Oriented Software", Proceedings of the Annual Oregon Workshop on Software Metrics (AOWSM95), 1995.

[66] Ott L. M., Bieman J. M., "Program Slices as an Abstraction for Cohesion Measurement", Journal of Information and Software Technology, 40(11-12):691-699, November 1998.

[67] OMG, UML Specification v. 1.3 Draft, http://www.omg.org

[68] Porter A. A., Selby R. W., "Empirically Guided Software Development Using Metric-Based Classification Trees", IEEE Software, 7(2): 46-54, Mar. 1990.

[69] Purao S., Vaishnavi V. "Product Metrics for Object-Oriented Systems" ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 191-221.

[70] Schmuller J., "Sams Teach Yourself UML in 24 Hours", USA, Sams Publishing, 1999.

[71] Selby R.W., Porter A. A., "Learning From Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis", IEEE Transactions on Software Engineering, 14(12): 1743-1757, Dec. 1988.

[72] Sheldon F. T., Jerath K., and Chung H., *"Metrics for Maintainability of Class Inheritance Hierarchies"*, article accepted for publication in the Journal of Software Maitenance and Evolution: Research and Practice Copyright © 2002.

[73] Shumway M. F., "Measuring Class Cohesion in Java", Computer Science Department, Colorado State University, June 11, 1997.

[74] Simon F., Loffler S., Lewerentz C., *"Distance Based Cohesion Measuring"* Accepted for FESMA99, Amsterdam 4 -8 October.

[75] Simon F., Loffler S., Lewerentz C., *"Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object-Oriented Systems"* , R. Dumke and A. Abran (Eds.): IWSM 2000, LNCS 2006, pp. 1-17, Berlin Heidelberg 2001

[76] Wasiq M., "Measuring Class Cohesion in Object-oriented Systems: MS Thesis", Information and Computer Science Department, King Fahd University of Petroleum and Minerals, 2001.

[77] Weyuker, E., "Evaluating Softwaer Complexity Measures", Third International Conference on Applications of Software Measurement, 1992, La Jolla, California.

[78] W3C, Extensible Markup Language (XML), http://www.w3.org

[79] Youness S., Boutquin P. et al., "SQL Unleashed" SAMS Publishing, USA, 2002.

[80] Yourdon E. and Constantine L., "Structured Design", Prentice Hall, 1979.

[81] Zhao J., Xu B., “Measuring Aspect Cohesion”, Proc. International Conference on Fundamental Approaches to Software Engineering (FASE'2004), LNCS 2984, pp.54-68, Springer-Verlag, Barcelona, Spain, March 29-31, 2004.

[82] Zuse, H., “Properties of Software Measures”, Software Quality Journal, vol. 1, pp. 225 -260, 1992.