CHAPTER 4

A NEW APPROACH TO DESIGN REUSABILITY METRIC BASED UPON OBJECT-ORIENTED METRICS

4.1 Introduction
4.2 Related Work
4.3 Overview of CK Metrics
4.4 Object Oriented Structure
4.5 Proposed Metrics
  4.5.1 Inheritance Metric (DIT+NOC)
  4.5.2 Interaction Metric (CBO+LCOM)
  4.5.3 Structural Metric (WMC+RFC)
4.6 Description of Empirical Study
  4.6.1 Data Set
  4.6.2 Data Collection Procedure & Method
4.7 Object-Oriented Design
4.8 Analysis Results
  4.8.1 Evaluation Results for CK Metrics
  4.8.2 Evaluation Results for Proposed Metrics
4.9 Summary

The major findings of the Chapter have been published in

4.1 INTRODUCTION

Object oriented systems continue to share a major portion of software development and customer base for these systems is on the rise. This is because there are many advantages in taking the object oriented concept. The weakness though is that most object oriented systems tend to be quite complex. Hence, the quality of such systems takes priority and lots of time, money and effort is spent in ensuring it (Kayarvizhy and Kanmani, 2011). One such method that predicts quality of a software system is by evaluating impact on reusability of class of the software through the use of metrics. The introduction and subsequent use of metrics as a means to evaluate the software quality has had deep and useful impact on the overall system.

The subject of software reuse has received much attention over the last two decades, especially with the widespread use of object technology. Software houses that embrace object technology are starting their own reuse programs in which the reuse of components is monitored and analyzed. Many developers are experiencing the benefits of reuse in project completion time and cost (Terry and Dikel, 1996). Many other developers believe software reuse provides the key to enormous savings and benefits in software development; the U.S. Department of Defense alone could save $300 million annually by increasing its level of reuse by as little as 1% (Anthes, 1993).

Reuse of existing software components increases quality and productivity in software development and maintenance. Software reuse reduces the amount of software that needs to be produced from scratch and hence reduces the testing time for new software. Industrial observers suggest that a reuse strategy could save up to 20% of development costs (Henderson-Sellers, 1996). With reuse, software development becomes a capital investment. C++ features are used to support reusability in object-oriented programming. Many researchers have done research on reusability metrics (Barnard, 1998), (Aggarwal et al. 2005), (Aggarwal et al. 2006).

In this chapter, an attempt is made to use software metrics as a quality predictor for software characteristics of the underlying system. The study consists of
calculating and analyzing object-oriented metrics on three programs developed using C++. The following section represents a review of related work. Section 4.3 discusses the brief description of the six class based CK Metrics. Section 4.4 describes object-oriented structure. Following that Section 4.5 proposed a set of three new metrics Inheritance Metric, Interaction Metric and Structural Metric. Section 4.6 discusses our empirical study and provides more details of the programs considered. Section 4.7 discusses the object-oriented design of the programs considered. Section 4.8 describes the analysis of the results and the findings. Section 4.9 summarizes the study undertaken and presents conclusions.

4.2 RELATED WORK

A lot of object-oriented metrics have been proposed in the literature (Xenos et al. 2000). Chidamber and Kemerer proposed six measures which are the most widely used design measures for object-oriented systems, focusing on classes and class hierarchies. After the inception of the CK metric suite, many have introduced new class level design metrics. For example, metrics proposed by Abreu, Li and Henry metrics, MOOD metrics, Lorenz and Kidd metrics etc (Abreu and Carapuca, 1994), (Li and Henry, 1993), (Abreu, 1995), (Lorenz and Kidd, 1994). CK metrics are the most popular among them. One of the first suites of OO design measure was proposed by Chidamber and Kemerer (1994) will be discussed here. The authors of this suite of metrics claim that these measures can aid users in understanding design complexity, in detecting design flaws and in predicting certain project outcomes and external software qualities such as reusability, software defects, testing, and maintenance effort. Use of the CK set of metrics and other complementary measures are gradually growing in industry (Subramanyam and Krishnan, 2003). The CK metrics suite is one of the object-oriented design complexity measurement systems which support the measurement of the external quality parameter (Chidamber and Kemerer, 1994).

4.3 OVERVIEW OF CK METRICS

Brief description of the six CK metrics suite for OO Design (Shatnawi, 2010a), (Erika, 2008) is the deepest research in OO metrics investigation:
I. **Weighted Methods per Class (WMC)**

It is defined as the sum of the complexities of all methods of a class.

- The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class.
- The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class.
- Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.

II. **Depth of Inheritance Tree (DIT)**

It is defined as the maximum length from the node to the root of the tree.

- The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior.
- Deeper trees constitute greater design complexity, since more methods and classes are involved.
- The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

III. **Number of Children (NOC)**

It is defined as the number of immediate subclasses.

- The greater the number of children, the greater the reuse, since inheritance is a form of reuse.
- The greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of subclassing.
• The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

IV. Coupling between Object Classes (CBO)

It is defined as the count of the classes to which this class is coupled. Coupling is defined as: Two classes are coupled when methods declared in one class use methods or instance variables of the other class (Chidamber and Kemerer, 1994).

• Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

• In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.

• A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

V. Response for a Class (RFC)

It is defined as number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class.

• If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding on the part of the tester.

• The larger the number of methods that can be invoked from a class, the greater the complexity of the class.

• A worst case value for possible responses will assist in appropriate allocation of testing time.
VI. Lack of Cohesion in Methods (LCOM)

It is defined as the number of different methods within a class that reference a given instance variable.

- A highly cohesive module should stand alone; high cohesion indicates good class subdivision.
- High cohesion implies simplicity and high reusability.
- Cohesiveness of methods within a class is desirable, since it promotes encapsulation. As a drawback, a highly cohesive class has high coupling between the methods of class, which in turn indicates high testing effort for that class.
- Lack of cohesion implies classes should probably be split into two or more subclasses.
- Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

Object-oriented methodologies require significant effort early in project life cycle to identify objects and classes, attributes and operations, relationships between objects and classes, encapsulation, inheritance, and polymorphism require designers to carefully structure the design and consider the interaction between objects. Accordingly, much effort will be saved rather than rewriting the code and helps producing high quality software. In the current work, CK suite is utilized for several reasons: CK suite covers all aspects of object oriented (reusability, encapsulation and polymorphism). CK suite also proves to be useful in predicting class fault proneness (Benlarbi, 2000).

4.4 OBJECT ORIENTED STRUCTURE

The new object oriented development methods have their own terminology to reflect the new structural concepts. An object oriented system starts by defining a class that contains related or similar attributes and operations (some operations are methods). The classes are used as the basis for objects. A child class inherits all of the attributes and operations from its parent class, in addition to having its own attributes
A class is a template from which objects can be created. This set of objects shares a common structure and a common behavior manifested by the set of methods. A method is an operation upon an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method. Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Effective object oriented designs maximize cohesion because cohesion promotes encapsulation. Coupling is a measure of the strength of association established by a connection from one entity to another. Classes (objects) are coupled when a message is passed between objects; when methods declared in one class use methods or attributes of another class. Inheritance is the hierarchical relationship among classes that enables programmers to reuse previously defined objects including variables and operators.
Let us consider an OO application example with 5 classes with root class TCP/IP to illustrate the OO CK metrics as shown in figure 4.2.

Figure 4.2 Object Oriented Design for TCP/IP Classification Model
It is an object oriented application example with 5 classes; the root or main class is TCP/IP Model and four child classes are Host to network layer, Network layer, Transport layer and Application layer.

Table 4.1 Values of CK Metrics for TCP/IP Classification Model

<table>
<thead>
<tr>
<th>Classes</th>
<th>WMC</th>
<th>DIT</th>
<th>NOC</th>
<th>CBO</th>
<th>RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>Host to network</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Network</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Transport</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Application</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

4.5 PROPOSED METRICS

A set of new metrics are proposed to measure reusability of an OO codes (Goel and Bhatia, 2012).

4.5.1 Inheritance Metric (DIT+NOC)

It is a combination of inheritance path from the class to the root class and the number of immediate subclasses of a class.

- Deeper a particular class is in the hierarchy, the greater the potential for reuse of inherited methods (Liang and Colemon, 2001). It states that reusability of a class increases with increase in DIT of a class. So DIT has positive impact on reusability of a class.
- A moderate value for NOC indicates scope for reuse (Liang and Colemon, 2001). Up to particular threshold value NOC has positive impact on reusability of a class.
Therefore the reusability of a class increases with the increase in combination of DIT and NOC of a class. So DIT + NOC have positive impact on reusability of a class.

Let \( \text{Metric1} = \text{Inheritance Metric} = \text{DIT (Depth of Inheritance Tree)} + \text{NOC (Number of Children)} \)

### 4.5.2 Interaction Metric (CBO+LCOM)

It is a combination of the number of classes to which a class is coupled and the number of different methods within a class that reference a given instance variable.

- Excessive coupling indicates weakness of class encapsulation and may inhibit reuse (Liang and Colemon, 2001). It indicates that coupling has negative impact on reusability of a class.
- High LCOM increases complexity, thereby increasing likelihood of errors during the development process. The class should probably split into two or more smaller classes. It indicates that high LCOM has negative impact on reusability of a class.
- Therefore the reusability of a class decreases with the increase in combination of CBO and LCOM of a class. So CBO + LCOM have negative impact on reusability of a class.

Let \( \text{Metric2} = \text{Interaction Metric} = \text{CBO (Coupling between Objects)} + \text{LCOM (Lack of Cohesion in Methods)} \)

### 4.5.3 Structural Metric (WMC+RFC)

It is a combination of the sum of the complexities of all methods of a class and the set of methods that can potentially be executed in response to a message received by an object of that class.
The large no. of methods in a class, the greater the potential impact on children. Classes with large no. of methods are likely to be more application specific, limiting the possibility of reuse. So WMC has negative impact on reusability of a class.

The larger the no. of methods that can be invoked from a class through message, the greater the complexity of the class. So RFC has negative impact on reusability of a class.

Therefore the reusability of a class decreases with the increase in combination of WMC and RFC of a class. So WMC+RFC have negative impact on reusability of a class.

Let Metric3 = Structural Metric=WMC (Weighted Methods per Class) + RFC (Response for a Class).

The proposed set of metrics Metric1, Metric2 & Metric3 is applied to C++ program in Figure 4.3 to measure the impact of reusability of a class.
A New Approach to Design Reusability Metric Based Upon OO Metrics

#include<iostream.h>
#include<conio.h>
class GF {
    int a;
    public:
        GF(int x) {
            a=x;
        }
        int geta() {
            return a;
        }
};
class F: public GF {
    int b;
    public:
        F(int x, int y):GF(y) {
            b=x;
        }
        int getb() {
            return b;
        }
};
class S: public F {
    int c;
    public:
        S(int x, int y, int z):F(y,z) {
            c=x;
        }
        void show() {
            cout << geta() << " " << getb() << " " ;
            cout << c << "\n" ;
        }
};
main() {
    clrscr();
    S ob(1,2,3);
ob.show();
cout << ob.geta() << " " << ob.getb() << "\n" ;
getch();
return 1;
}

Figure 4.3 C++ Program
Table 4.2 Values of Proposed Metrics

<table>
<thead>
<tr>
<th>Metric Class</th>
<th>DIT</th>
<th>NOC</th>
<th>Metric1= DIT+NOC</th>
<th>CBO</th>
<th>LCOM</th>
<th>Metric2= CBO+LCOM</th>
<th>WMC</th>
<th>RFC</th>
<th>Metric3= WMC+RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF (Grand Father)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>F (Father)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>S (Son)</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

The investigation of values of Metric1, Metric2 and Metric3 from Table 4.2 based upon C++ program from figure 4.3 has negative impact on reusability of class.

The organizations implement systematic software reuse programs in an effort to improve productivity and designing. Reusability increases with increase of DIT and NOC, reusability decreases with increase of CBO and LCOM, reusability decreases with increase of WMC and RFC.

In this chapter, we have proposed an approach to measure the reusability of object oriented program based upon CK metrics. Since reusability is an attribute of software design, we can analyze software design by measuring software reusability. Hence, this approach is important to measure reusability of class diagram.

4.6 DESCRIPTION OF EMPIRICAL STUDY

4.6.1 Data Set

Three programs developed in C++ have been considered for this study. The first program uses “Multilevel Inheritance,” which means a class is derived from another derived class. The second program uses “Multiple Inheritance,” which means one class inherits the features of multiple classes. The third program uses “Hierarchical Inheritance,” which means the traits of one class may be inherited by more than one class. All the programs were monitored for their use of object-oriented features in the design phase to avoid a procedural oriented approach. All programs
contained 3 classes each. The lines of code were also comparable among the three programs.

4.6.2 Data Collection Procedure and Method

The metrics were calculated using a semi automated approach. A tool was specifically developed for this purpose. The details of the classes, attributes, methods and their usage were provided to the tool which then calculated the various metrics based on this input.

Object-oriented methodologies require significant effort early in the project life cycle to identify objects and classes, attributes and operations, relationships between objects and classes, encapsulation, inheritance, and polymorphism require designers to carefully structure the design and consider the interaction between objects. Accordingly, much effort will be saved rather than rewriting the code which helps when producing high quality software.

4.7 OBJECT-ORIENTED DESIGN

Let us consider an object-oriented design of the programs to illustrate the class, attributes, methods and objects. An object-oriented system starts by defining a class that contains related or similar attributes and methods. The classes are used as basis for objects.

A class is a template from which objects can be created. This set of objects shares a common structure and a common behavior manifested by the set of methods. A method is an operation upon an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method.

Let us consider three hypothetical object-oriented designs for analysis written in the C++ language as illustrated in figures 4.4, 4.5, & 4.6 for which source code is available with authors.
(a) Figure 4.4 Illustrates the OO design for Multilevel Inheritance.
(b) Figure 4.5 Illustrates the OO design for Multiple Inheritance.
(c) Figure 4.6 Illustrates the OO design for Hierarchical Inheritance

![Diagram]

Figure 4.4 Object-Oriented Design for Multilevel Inheritance
Figure 4.5 Object-Oriented Design for Multiple Inheritance
4.8 ANALYSIS RESULTS

The CK metrics provide insight on how the classes are dependent on each other. Inheritance metrics provide details of the various inheritance attributes. A high degree of inheritance is detrimental to the health of the system while too little inheritance might miss out on the advantages of object-oriented concepts. While there
is no hard rule on the amount of inheritance, logical conclusions can be derived by measuring them (Kayarvizhy and Kanmani, 2011).

Strong coupling complicates a system, since a module is harder to understand, change, or correct if it is interrelated with other modules. The more independent a class, the easier it is to reuse it in another application. The larger the number of couples is, the higher the sensitivity to changes in other parts of the design will be leading to difficulty in maintainability (Kayarvizhy and Kanmani, 2011).

Cohesion metrics are actually inverse in nature – meaning they measure the lack of cohesion. High cohesion indicates good class subdivision (Kayarvizhy and Kanmani, 2011).

### 4.8.1 Evaluation Results for CK Metrics

The following table provides the values for the various metrics i.e. inheritance metrics, coupling metrics and cohesion metrics for the three programs considered for the study.

**Table 4.3 CK Metrics Class-wise**

<table>
<thead>
<tr>
<th>Class</th>
<th>Metric</th>
<th>Program1</th>
<th>Program2</th>
<th>Program3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DIT</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>NOC</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>CBO</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>LCOM</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>WMC</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>RFC</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>DIT</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NOC</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CBO</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>LCOM</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>WMC</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
A New Approach to Design Reusability Metric Based Upon OO Metrics

Where DIT= Depth of Inheritance Tree, NOC=Number of Children, CBO=Coupling between Objects, LCOM=Lack of Cohesion in Methods, WMC=Weighted Methods per Class, RFC=Response for a Class.

![Figure 4.7 CK Metric Graph](image-url)
Table 4.4 CK Metrics using Average of Classes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Program1</th>
<th>Program2</th>
<th>Program3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIT</td>
<td>1</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>NOC</td>
<td>1</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>CBO</td>
<td>1.3</td>
<td>1.3</td>
<td>1</td>
</tr>
<tr>
<td>LCOM</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WMC</td>
<td>3.7</td>
<td>3.7</td>
<td>3.7</td>
</tr>
<tr>
<td>RFC</td>
<td>6.3</td>
<td>5.3</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Figure 4.8 CK Metric Graph using Average of Classes

4.8.2 Evaluation Results for Proposed Metrics

The following table provides the values for the proposed metrics i.e. Inheritance metric (Metric1), Interaction Metric (Metric2) and Structural Metric (Metric3) for the three programs considered for the study.
Table 4.5 Proposed Metrics

<table>
<thead>
<tr>
<th>Class</th>
<th>Metric</th>
<th>Program1</th>
<th>Program2</th>
<th>Program3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Metric1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Metric2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Metric3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>Metric1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Metric2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Metric3</td>
<td>10</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>Metric1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Metric2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Metric3</td>
<td>16</td>
<td>17</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 4.9 Proposed Metric Graph
Table 4.6 Proposed Metrics using average of classes

<table>
<thead>
<tr>
<th>Metric values using Average of classes</th>
<th>Metric</th>
<th>Program1</th>
<th>Program2</th>
<th>Program3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric1</td>
<td>2</td>
<td>1.3</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>Metric2</td>
<td>2.3</td>
<td>2.3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Metric3</td>
<td>10</td>
<td>9</td>
<td>8.7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.10 Proposed Metric Graph using Average of Classes

The metric values listed above for all the programs indicate that inheritance has been used moderately in all the programs. From Tables 4.3, 4.4, 4.5, 4.6 and Figures 4.7, 4.8, 4.9, 4.10, it is clear that DIT, NOC, Metric1 values are higher for Program1 than Program2 and Program3. But CBO, LCOM, Metric2 values are almost same for Program1, Program2 and Program3. Similarly WMC, RFC, Metric3 values are also almost same for Program1, Program2 and Program3. So as according to
Inheritance Metric from section 4.5.1, Program1 means multilevel inheritance is more reusable than multiple inheritance and hierarchical inheritance.

4.9 SUMMARY

In this study, object-oriented metrics have been measured for three C++ programs under the categories of inheritance, coupling and cohesion. The metrics have been analyzed and used to understand the various characteristics of the object-oriented systems. The conclusion that can be drawn from this study is that all the programs show good use of object-oriented features and result in reusable classes. It has also been found that out of the three features, Multilevel Inheritance has more impact on reusability. This study hence not only helps to get some understanding of the object-oriented systems but also proves that the metrics are good at evaluating the object-oriented system.