IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

A Quality Based Novel Subclass Coupling Factor Metric for Evaluating Software

Dr. S.A. Sahaya Arul Mary[1], M.Kavitha[2]

Dean Academics, Jayaram College of Engineering & Technology, Trichy[1]

Research Scholar, Dept. of Computer Science, Manonmaniam Sundaranar University, Tirunelveli[2]

Abstract: Software metrics are developed for the purpose of evaluating the software in all stages of the development process to ensure the inhabitant of quality in all phases. Though the evolution of software programming has been developing over the years, the categorization of software metrics are centered only on procedure oriented software programming and object oriented software programming. In object oriented modular programming, coupling refers to a justifying factor for measuring the quality of program code. As for the code concern, a module with low coupling and high cohesion is an ideal objective of the programmers. The analysis over the classification of types of coupling in object oriented programming is still diminutive and has been an active research in software metrics. The objective of this paper is to propose a coupling metric that could possibly identify the subclass coupling one of the types of coupling, that exist in the software modules there by highlighting the modules that can be focused for further improvement.

Keywords: software metrics, cohesion, coupling, coupling factor, weyuker's properties.

I. INTRODUCTION

Software metrics in software engineering is the process of measuring the quality of software. Software metrics are usually incorporated with testing phase of software development life cycle. Software testing not only verifies the requirements, design, and functionalities of code but also to ensure the qualitative writing of program. The verification of quality of code depends on how well the modularization of the software program is constructed. The two important factors that can assess the effective modularization of the program code are coupling and cohesion. Coupling is the measure of the degree of relationship between modules. The measurement of coupling over the structured development context was first defined by Stevens et al. during the year 1974 [1]. Coupling measures the interdependencies between one or more objects. For example, objects A and B are said to be coupled if a method of object B accesses or calls a method or variable in object A. A classic design of the object-oriented programming necessitates the modules to be designed with low coupling [11]. As low coupling has a direct impact with the quality of good program code, it may be obligatory for the software to be assessed with the identification of types of coupling in object oriented programming. The types of coupling called, subclass coupling and temporal coupling [12] are the two streams of object oriented coupling where the prior describes the relationship between a parent and its children and the posterior bundles two actions into one module as they just happen to occur at the same time.

The primary goal of this paper is to propose novel software metric that identifies the complexity of the program code by computing the ratio of subclass coupling in modules. The higher value of the metric designates the higher the complexity of the modularization.

The remaining section of the paper is organized as follows: section 2 contains the review of literature; section 3 comprises the methodology of the proposed work, section 4 encompasses the illustration and finally section 5 comprehends the conclusion.

II. REVIEW OF LITERATURE

Yadav et al.[1] applied Cohesion and Coupling metrics on programs of inheritance and interface and evaluated the traditional software metrics values. The cohesion and Coupling metrics identified the complexity between inheritance and interface programming. The authors wanted to show the concepts that was good to use and beneficial for software developer. The authors also focused on an empirical evaluation of object oriented metrics in C#. The resulting values were analyzed to provide significant insight about the object oriented characteristics of reusability programs.

Aloysius et al. [2] presented a cognitive complexity metric namely cognitive weighted coupling between objects for measuring the types of coupling involved in object-oriented systems. The authors concentrated on five types of coupling that may exist between classes such as control coupling, global data coupling, internal data coupling, data coupling and lexical content coupling were considered in computing their proposed CWCBO. CWCBO had proven that, complexity of the class was getting affected based on the cognitive weights of the various types of coupling.

Poshyvanyk et al.[3] introduced set of coupling measures for OO systems – named conceptual coupling, based on the semantic information obtained from the source code, encoded in identifiers and comments. The authors have conducted a case study on open source software systems to compare the new measures with existing structural coupling measures. The authors proved that their case study was able to capture the conceptual coupling in new dimensions of coupling, which were not captured by existing coupling measures.

Misra et al.[4] proposed a metric that was based on the important feature of the OO systems: Inheritance. The metric calculates the complexity at method level considering internal structure of methods, and also considers inheritance to calculate the complexity of class hierarchies. The proposed metric was validated both theoretically and empirically. For theoretical validation, principles of measurement theory are applied since the measurement theory had been proposed and extensively used in the literature as a means to evaluate the software engineering metrics. The authors applied their metric on a real project for empirical validation and compared it with Chidamber and Kemerer (CK) metrics suite [5]. The theoretical, practical and empirical validations and the comparative study proved the robustness of the measure.

Aggarwal et al.[6] addressed the need for software metrics and introduced a new set of design metrics for object-oriented code. The authors developed two OO metrics for measuring the amount of robustness included in the code. The metrics were analytically evaluated against Weyuker's proposed set of nine axioms. These set of metrics were calculated and analyzed for standard projects and accordingly ways in which project managers could utilize these metrics were suggested by the authors.

III.BACKGROUND

A. Coupling Factor (CF)

The metric Coupling Factor (CF) was proposed by MOOD for assessing the ratio of coupling involved between classes [10]. The ratio of coupling is evaluated using a fraction, where the numerator represents the total number of non-inheritance couplings in the module and the denominator signifies the maximum number of coupling that is possible for the corresponding module. The maximum number of couplings includes both inheritance and non-inheritance related coupling.

$$CF = \frac{\sum_{i=1}^{n}\left[\sum_{j=1}^{n} is\_client(C_i, C_j)\right]}{(n^2 - n)}$$

where n is the total number of classes in the module. The value of *is_client* (C_i, Cj) is 1 if the class C_j calls a method or attribute of C_i or otherwise 0.

B. Motivation

The empirical study of various researches suggests creating modules with stronger coupling is more difficult to understand, to locate the origin of errors and to perform addition and modification of programs in the existing modules. Moreover, excessive coupling between objects is disadvantageous to modular design as more testing is required to achieve reliable results. Hence, to conclude, a module with low coupling is desirable. Though CF metric explicates the coupling between non-inheritance classes, and only exhibits the occurrence of direct coupling between the classes. Indirect coupling also plays a vital role in assessing the complexity of any module. A coupling metric should be capable of handling both direct and indirect coupling [8], hence the motivation of the proposed work is to modify the existing CF metric so as to evaluate both direct and indirect coupling in the module.

IV. METHODOLOGY

A. Subclass Coupling Factor

The proposed Subclass Coupling Factor metric (SCF) measures the direct and indirect subclasses of the individual class so as to calibrate the complexity of the whole module. SCF metric adopts the concepts of CF metric as its base and performs the union operation with the intersected sets as the results. Hence, the complexity of the whole module can be weighed to assess the quality of software modules.

$$C_L = C_{i=1}^{n}[C_{j=1}^{n} C_i(C_j)(is\_subclass(C_i, C_j))$$

C_L is the class labels of sets C_i .. to C_n where n is the total number of classes in a module, C_i ,C_j represents the i^{th} and j^{th} class respectively and C_i represents the sets of all subclasses of i^{th} class. The subclass elements are added onto the set C_i if and only if C_j is the subclass for C_i .

$$CO(C_{CL}) = C_{i=1}^{n}[C_i \cup all\ direct\ and\ indirect\ subclasses(C_i)]$$

CO is the complexity of each class.

$$SCF = \frac{\sum_{i=1}^{n} all\ elements\ in\ C_i}{\sum_{i=1}^{n}(n-i)}$$

B. Illustration

An illustration is conducted to evaluate the performance of the proposed SCF metric with an inheritance based java program. The program is chosen in such a way that the classes in the module are coupled directly and indirectly between one another. This section illustrates the step by step calibration of SCF in the following program.

Program 1:

```
class Shape
{
        public void getShapeValue()
        {
        System.out.println("getShape value Method");
        }
        public void setShapeValue()
        {
        System.out.println("setShape value Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
class Circle extends Shape
{
        public void getCircleValue()
```

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

```
        {
        System.out.println("getCircleValue Method");
        }
        public void setCircleValue()
        {
        System.out.println("setCircleValue Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
class RectAngle extends Shape
{
        public void getRectValue()
        {
        System.out.println("getRectValue Method");
        }
        public void setRectValue()
        {
        System.out.println("setRectValue Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
class SemiCircle extends Circle
{
        public void getSemiValue()
        {
        System.out.println("getSemiValue Method");
        }
        public void setSemiValue()
        {
        System.out.println("setSemiValue Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
class QCircle extends Circle
{
        public void getQValue()
        {
        System.out.println("getQValue Method");
        }
        public void setQValue()
        {
        System.out.println("setQValue Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
class TriAngle extends RectAngle
{
        public void getTriValue()
        {
        System.out.println("getTriValue Method");
        }
        public void setTriValue()
        {
        System.out.println("setTriValue Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
class Square extends RectAngle
{
        public void getSqureValue()
        {
        System.out.println("getSqureValue Method");
        }
        public void setSqureValue()
        {
        System.out.println("setSqureValue Method");
        }
        public void calculateValue()
        {
        System.out.println("Calulate Method");
        }
        public void displayValue()
        {
        System.out.println("Display Method");
        }
}
public class TotalShape
{
public static void main(String arg[])
{
Shape sh=new Shape();
sh.getShapeValue()
sh.setShapeValue()
sh.calculateValue();
sh.displayValue();
Circle cir=new Circle();
cri.getCircleValue()
cri.setCircleValue()
cri.calculateValue();
cri.displayValue();

RectAngle  rect=new RectAngle ();
rect.getRectValue()
rect.setRectValue()
rect.calculateValue();
rect.displayValue();

SemiCircle semi=new SemiCircle();
semi.getSemiValue()
semi.setSemiValue()
semi.calculateValue();
semi.displayValue();

QCircle qc=new QCircle();
qc.getQValue()
qc.setQValue()
qc.calculateValue();
qc.displayValue();

TriAngle tri=new TriAngle ();
tri.getTriValue()
tri.setTriValue()
tri.calculateValue();
tri.displayValue();
Squre sq=new Squre()
sq.getSqureValue();
sq.setSqureValue();
sq.calculateValue();
sq.displayValue();
}
}
```

C. Subclass Coupling Factor Analysis

Total Number of classes in the program: 8
C_{Shape}= {Circle, Rectangle}
C_{Circle}= {SemiCircle, QCircle}
$C_{Rectangle}$= {TriAngle,Square}
$C_{SemiCircle}$= {NULL}
$C_{QCircle}$= {NULL}
$C_{TriAngle}$= {NULL}
C_{Sqaure}= {NULL}
As SCF calculates the direct and indirect coupling of subclasses in the module exclude the main class for calculating the coupling complexity. Hence, TotalShape class in the program is excluded as it contains the main function in the program. Now calculate the complexity of each class by traversing through all subsequent subclasses of the parent class. Hence the complexity of each class is
C_{Shape}= {Circle, Rectangle, SemiCircle, QCircle, TriAngle,Square}
C_{Circle}= {SemiCircle, QCircle}
$C_{Rectangle}$= {TriAngle,Square}
$C_{SemiCircle}$= {NULL}
$C_{QCircle}$= {NULL}
$C_{TriAngle}$= {NULL}
C_{Sqaure}= {NULL}

Class A_{Shape} has four elements such as Circle, Rectangle, SemiCircle, QCircle with which Circle and Rectangle classes are the direct subclasses of Class Shape and SemiCircle, QCircle are indirect subclasses of Class Shape. Finally, the SCF for the program is calculated as the sum of all elements of each class in the module divided by the possible number coupling with the module.
SCF= (6+2+2)/8(8-1) =10/28=0.357
Coupling Factor:
CF= (1+1+1+1+1+1/8(8-1) =6/28=0.214

TABLE I: SUBCLASS COUPLING COMPLEXITY METRIC VALUE FOR PROGRAM 1

Program#	**SCF**	**CF**
1	0.357	0.214

The pictorial representation of the comparison of coupling factor and subclass coupling factor metrics is represented in Fig.1. The metric value of SCF is higher than CF as the complexity is higher. The metric enhances the complexity with increased value.

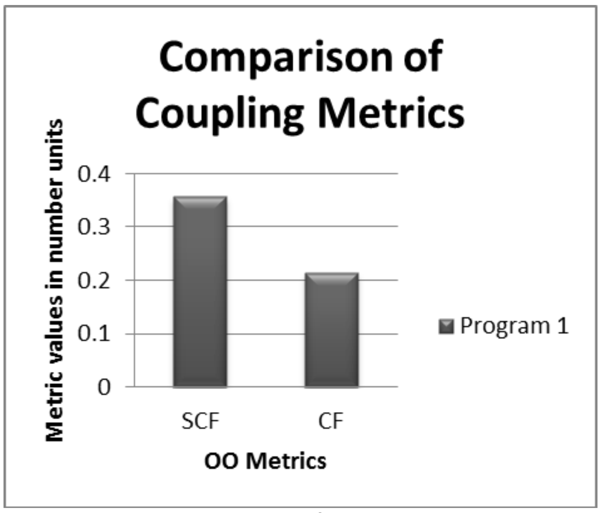

Fig. 1. Comparison of Coupling Metrics

V. SCF EVALUATION

Many inventions have suggested that software metric should satisfy certain properties for to evaluate their real time usability in development environment. Basili and Reiter[7] suggested that software metrics should be sensitive to external observable differences in development process, and should correspond to intuitive notions about the characteristic differences between the software artifacts being measured. Weyuker has also proposed an authorized list of properties for software metrics that could be evaluated on the existing software metrics [9]. The notions of the Weyuker's properties include permutation, interaction, monotonicity, non-researchers have recommended various properties uniqueness and so on. The challenge in this section is to evaluate the proposed SCF coupling metric against the nine properties of Weyuker's to prove its usefulness.

Though, several Weyuker's properties are considered to be most significant to classify the complexity of a measure. Weyuker's properties state that [1].

Property1
Non-coarseness:
$(\exists R)(\exists S)(\mu(R) \neq \mu(S))$
Not all class can have the same complexity. If there are 'n' numbers of modules in the software, SCF does not rank all 'n' modules as equally complex.

Property 2
Granularity: Let 'r' be a non-negative number and there could be only finite number of modules have the complexity r. If the number of modules in large scale system is finite, the complexity value of SCF is also finite. Hence this property is satisfied.

Property 3
$\mu(R) = \mu(S)$
Non-uniqueness: This property implies that there may be number of modules have the same complexity. SCF abides this property, if the hierarchy of class in the modules are similar, the complexity of the modules are also similar.

Property 4
$(\exists R)(\exists S)(R \equiv S)\ and\ (\mu(R) \neq \mu(S))$
Design details are important:
The property affirms that though if two classes have the same functionality, they may differ in terms of details of implementation. If the design implementation of two modules is different, SCF produces different complexity values for each module.

Property 5
Monotonicity:
For all modules R and S such that $\mu(R) \leq \mu(R + S\ and\ \mu S \leq \mu R + S$.
Let the concatenation of two modules R and S be R+S. Hence, this property states that complexity value of the combined class may be larger than the complexity of the individual classes R or S. SCF abides this property if there is a possibility of inheritance between the modules R and S while concatenation.

Property 6
Non-equivalence of interaction:
$$(\exists R)(\exists S)(\exists T) such\ that\ \mu(R) = \mu(S) does\ not\ imply\ that\ \mu(R + T) = \mu(S + T))$$
This property states that if a new module is added to the two existing modules R and S which has the same module complexity, if a new module T is added with both modules, the module complexities of the two new combined modules may be different or the interaction between R and T may be different than the interaction between S and T resulting in different complexity values for R + T and S + T. SCF for sure yields different complexity values for both modules R and S since T is dependent on the fitness of inheritance with the existing modules R and S.

Property 7
Permutation: There are program bodies I and J such that J is formed by permuting the order of the statements of I and (|I| = |J|). This property is not taken into the consideration of object oriented metrics.

Property 8
Renaming:
If R is a renaming of S then $\mu(R) = \mu(S)$
If module R is renamed as S then |R| = |S|. This property requires that renaming a module should not affect the complexity of the module. SCF does not have any impact over the change of name of module, hence SCF satisfies property 8.

Property 9
Interaction increases complexity:
$(\exists R)(\exists S)(\mu((R) + \mu(S)) < \mu(R + S))$
The property says that the class complexity measure of a new class combined from two classes may be greater than the sum of two individual class complexity measures. This property is not satisfied with SCF as the complexity of combined modules could be possibly equal to the individual complexity but not greater. Summary of the SCF validation is described in Table 2.

TABLE II
EVALUATION OF SCF AGAINST WEYUKER'S PROPERTIES

S.No	Property	Result
1	Non-Coarseness	Satisfied
2	Granularity	Satisfied
3	Non-uniqueness	Satisfied
4	Design details matter	Satisfied
5	Monotonicity	Satisfied
6	Non-equivalence of interaction	Satisfied
7	Interaction among statements	Not Applicable for object oriented Programming
8	No change on renaming	Satisfied
9	Interaction increases complexity	Not Satisfied

VI. CONCLUSION

Subclass coupling is an important factor for assessing the quality of software programming. A module that contains more subclass coupling increases the complexity of the software which should further be focused for simplification. So far, in the existing literature there is no specific metric available for computing the subclass coupling. As an effort in this paper, we have proposed a novel Subclass Coupling Factor (SCF) metric which evaluates the complexity of a module in terms of subclass coupling. The complexity values of SCF should range from 0 to 1, where 0 represents low subclass coupling and 1 represents high subclass coupling. An high subclass coupling is an alarm for the programmers as it implicitly depicts the high complexity in program design. Low subclass coupling is desirable as it reduces the code complexity.

Moreover, a metric is considered as valid if it satisfies at least seven of the weyuker's properties. The proposed SCF metric also satisfies seven properties of weyuker's as it is depicted in Table. 2 and proven as a valid object oriented metric for evaluating the subclasses coupling involved in any module.

REFERENCES

[1] Yadav, Maya, Jasvinder Pal Singh, and Pradeep Baniya. Complexity Identification of Inheritance and Interface based on Cohesion and Coupling Metrics to Increase Reusability. International Journal of Computer Applications 64.8 (2013).
[2] Aloysius, A., and L. Arockiam. Coupling Complexity Metric: A Cognitive Approach. International Journal of Information Technology and Computer Science (IJITCS) 4.9 (2012): 29.
[3] Poshyvanyk, Denys, and Andrian Marcus. The conceptual coupling metrics for object-oriented systems. Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on. IEEE, 2006.
[4] Misra, Sanjay, Ibrahim Akman, and Murat Koyuncu. An inheritance complexity metric for object-oriented code: A cognitive approach. Sadhana 36.3 (2011): 317-337.
[5] Shepperd, M. J., S. Chidamber, and Chris F. Kemerer. A metrics suite for object oriented design. Software Engineering, IEEE Transactions on 21.3 (1995): 263-265.
[6] Aggarwal, K. K., et al. Software Design Metrics for Object-Oriented Software. Journal of Object Technology 6.1 (2007): 121-138.
[7] Basili, Victor R., and Robert W. Reiter Jr. Evaluating automatable measures of software development. Proceedings on Workshop on Quantitative Software Models. 1979.
[8] Sharma, Aman Kumar, Arvind Kalia, and Hardeep Singh. Metrics identification for measuring object oriented software quality. International Journal of Soft Computing and Engineering 2.5 (2012): 255-258.
[9] Gandhi, Parul, and Pradeep Kumar Bhatia. Analytical Analysis of Generic Reusability: Weyuker’s Properties.International Journal of Computer Science 9.2 (2012).
[10] Chou, Chen-huei. Metrics in Evaluating Software Defects. DEF 18 (2013): 86-4.
[11] Gui, Gui, and Paul D. Scott. Measuring software component reusability by coupling and cohesion metrics. Journal of computers 4.9 (2009): 797-805.
[12] https://en.wikipedia.org/wiki/Coupling_(computer_programming)