

An Approach to Measure Software Reusability of OO Design

Pradeep Kumar Bhatia*, Rajbeer Mann**

Guru Jambheshwar University of Science and Technology, Hisar

*pk_bhatia2002@yahoo.com, **rajbeer.mann@gmail.com

Abstract - Software reusability has considerable effect on software quality. Software quality increases as reuse of software components increases. But software quality cannot be improved unless it can be measured. In this paper we have derived a new approach to measure the software reusability of OO class diagram, an attribute of software quality. We have also empirically studied the approach in OO programming language C++.

Key-words: - software reuse, reusability, metrics, CK metrics, Cyclomatic complexity

I. INTRODUCTION

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 monitored and analyzed.

Many developers are experiencing the benefits of reuse in project completion time and cost [1]. 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% [2].

However, we have yet to identify a reliable way to quantify the “reusable” software. Such a measure may help us not only to learn how to build reusable components but also to identify reusable components among the wealth of existing programs. Existing programs contains years of knowledge and experience gained from working in the application domain and meeting the organization’s software needs. If we could extract this information efficiently, we could gain a valuable resource upon which to build future applications.

As yet little research has been carried out in presenting an actual metric – only guidelines have been given. This paper takes a bold step and calculates the value of reusability of object-oriented code. It will be shown that the metrics that are important to calculate reusability are related to inheritance and coupling.

A. TRADITIONAL METRICS

McCabe Cyclomatic Complexity (CC): Cyclomatic complexity is a measure of a module control flow complexity based on graph theory [3]. Cyclomatic complexity of a module uses control structures to create a control flow matrix, which in turn is used to generate a connected graph. The graph represents the control paths through the module. The complexity of the graph is the complexity of the module [4], [3]. Fundamentally, the CC of a module is roughly equivalent

to the number of decision points and is a measure of the minimum number of test cases that would be required to cover all execution paths. A high cyclomatic complexity indicates that the code may be of low quality and difficult to test and maintain.

Source Lines of Code (SLOC): The SLOC metric measures the number of physical lines of active code, that is, no blank or commented lines code [5]. Counting the SLOC is one of the earliest and easiest approaches to measuring complexity. It is also the most criticized approach [6]. In general the higher the SLOC in a module the less understandable and maintainable the module is.

Comment Percentage (CP): The CP metric is defined as the number of commented lines of code divided by the number of non-blank lines of code. Usually 20% indicates adequate commenting for C++ [7]. A high CP value facilitates in maintaining a system.

B. CK METRICS

Weighted Methods per Class (WMC): WMC measures the complexity of an individual class. Two different approaches are used to calculate the WMC metric. The first uses the sum of the complexity of each method contained in the class. The second approach assigns a complexity of 1 for each method in the class and then sums the result. This is equivalent to using the number of methods per class as a measure for WMC [8]. The number of methods and complexity of methods involved is a direct predictor of how much time and effort is required to develop and maintain the class.

Depth of Inheritance Tree of a Class (DIT): DIT is defined as the length of the longest path of inheritance ending at the current module [8]. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the root of the tree [8]. The deeper the inheritance tree for a class, the harder it might be to predict its behavior due to the interaction between the inherited features and new features. However, the deeper a particular class is in the hierarchy, the greater the potential for reuse of inherited methods.

Number of Children (NOC): NOC represents the number of immediate subclasses subordinated to a class in the class hierarchy [8]. A moderate value for NOC indicates scope for reuse and high values may indicate an inappropriate abstraction in the design. Classes with a large number of children have to provide more generic service to all the children in various contexts and must be more flexible, a

constraint that can introduce more complexity into the parent class.

Coupling between Objects (CBO): CBO is defined as the count of the number of other classes to which it is coupled [8]. A class is coupled to another class if it uses the member method and/or instance variables of the other class. Excessive coupling indicates weakness of class encapsulation and may inhibit reuse. High coupling also indicates that more faults may be introduced due to inter-class activities.

Response for a Class (RFC): RFC gives the number of methods that can potentially be executed in response to a message received by an object of that class [8]. 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 required on the part of the tester.

Lack of Cohesion in Methods (LOCM): LOCM counts the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. The larger the number of similar methods in a class the more cohesive the class is [8]. Cohesiveness of methods within a class is desirable, since it promotes encapsulation and lack of cohesion implies classes should probably be split into two or more subclasses.

II. PROPOSED APPROACH

Our approach is to derive formula to measure reusability of a class diagram based on following principles:

- Deeper a particular class is in the hierarchy, the greater the potential for reuse of inherited methods. [9] 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 [9]. Up to particular threshold value NOC has positive impact on reusability of a class.
- Excessive coupling indicates weakness of class encapsulation and may inhibit reuse [9]. It indicates that coupling has negative impact on reusability of a class.

$$\text{Reusability of a class} = a*(DIT) + b*(NOC) - c*(CBO) \quad \dots (1)$$

Where a, b, c are empirical constants

For convenience, we take a = b = 1 and c = 0.5

Thus reusability of a class diagram is same as class having maximum reusability.

$$\text{Reusability of class diagram} = \max(\text{reusability}(\text{class}_i))$$

Where i = 1 to n (2)

III. EMPIRICAL STUDY

To empirically validate the formula we have taken five object oriented programs and calculated the values of CK metrics for each program.

Figure 1. shows an OO design and Table 1 shows the values of CK metrics obtained from OO design depicted in figure 1.

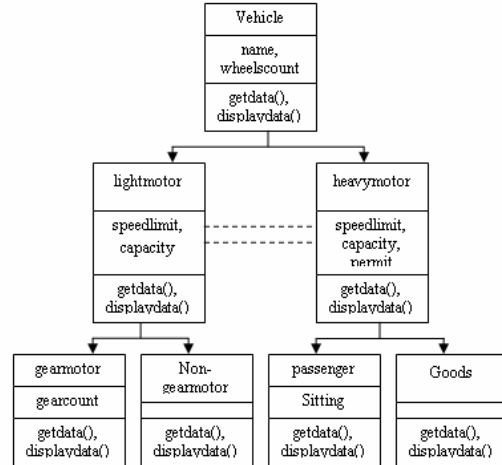


Figure 1: object oriented design for vehicle classification program

Table 1. values of CK metrics for vehicle classification.

Classes	WMC	DIT	NOC	CBO	RFC
Vehicle	2	0	2	0	8
Lightmotor	2	1	2	2	6
Heavymotor	2	1	2	2	6
Gearmotor	2	2	0	0	2
Non-gearmotor	2	2	0	0	2
Passenger	2	2	0	0	2
goods	2	2	0	0	2

Figure 2 shows graph of reusability vs classes of vehicle classification

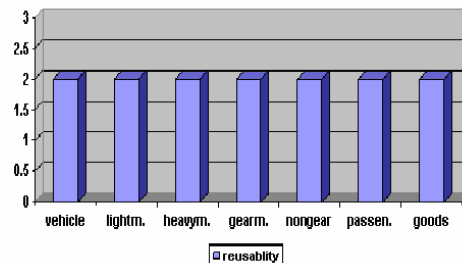


Figure 2. Reusability of various classes.

Table 2. depicts the reusability of all classes and reusability of class diagram vehicle classification

Class name	Class reusability	Reusability of class diagram
Vehicle	2	2
Lightmotor	2	
Heavymotor	2	
Gearmotor	2	
Nongearm.	2	
Passenger	2	
Goods	2	

Figure 3. shows an Object Oriented design for student education program and Table 3 shows the values of CK metrics of class diagram depicted in figure 3.

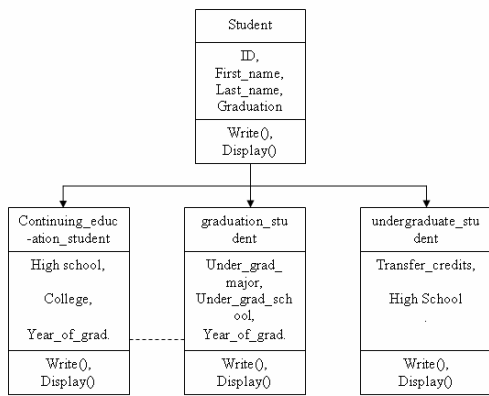


Figure 3. Object Oriented Design for Student Education program

Table 3. Values of CK metrics for Student Education.

Class	WMC	DIT	NOC	CBO	RFC
Student	2	0	3	0	8
Conti.edu	2	1	0	1	2
Graduation	2	1	0	1	2
Under-grad.	2	1	0	0	2

Figure 4 shows a graph of reusability vs classes of Student Education

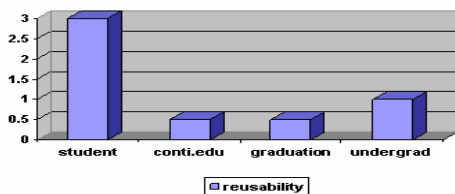


Figure 4. reusability of various classes of student program. Table 4. depicts the reusability of all classes and reusability of class diagram student education.

Class name	Class	Reusability
------------	-------	-------------

	reusability	of class diagram
Student	3	3
Conti.edu	0.5	
Graduation	0.5	
Undergrad.	1	

Figure 5. shows the Object Oriented Design for Student Result and Table 5. shows the values of CK metrics class diagram depicted in figure5.

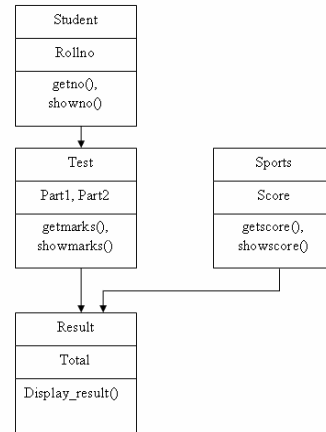


Figure 5. Object Oriented Design for Student Result program

Table 5. Values of CK metrics for Student Result class diagram

Classes	WMC	DIT	NOC	CBO	RFC
Student	2	0	1	0	3
Test	2	1	1	0	3
Sports	2	0	1	0	3
Result	1	2	0	0	1

Figure 6 shows a graph of reusability vs classes of Student Result

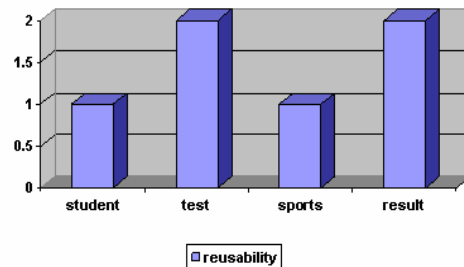


Figure 6. reusability of various classes of student result program.

Table 6. depicts the reusability of all classes and reusability of class diagram student result

Class name	Class reusability	Reusability of class diagram
Student	1	2
Test	2	
Sports	1	
Result	2	

Figure 7. shows Object Oriented design for Shapes Drawing program and Table 7 shows the values of CK metrics for class diagram shown in figure 7.

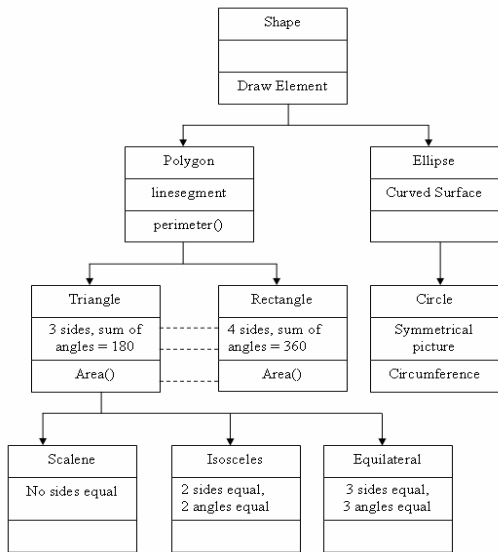


Figure 7: Object Oriented design for Shape Drawing program

Table 7. Values of CK metrics for Shape Drawing

Classes	WMC	DIT	NOC	CBO	RFC
Shape	1	0	2	0	8
Polygon	1	1	2	0	9
Ellipse	0	1	1	0	0
Triangle	1	2	3	3	4
Rectangle	1	2	0	3	4
Circle	1	2	0	0	1
Scalene	0	3	0	0	0
Isosceles	0	3	0	0	0
Equilateral	0	3	0	0	0

Figure 8 shows a graph of reusability vs classes of shape drawing class diagram

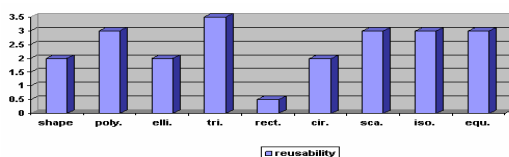


Figure 8. reusability of various classes of shape drawing class diagram.

Table 8 depicts the reusability of all classes and reusability of class diagram shape drawing.

Class name	Class reusability	Reusability of class diagram
Shape	2	3.5
Polygon	3	
Ellipse	2	
Triangle	3.5	
Rectangle	0.5	
Circle	2	
Scalene	3	
Isosceles	3	
Equilateral	3	

Figure 9 shows Object Oriented design for Trade Classification program and Table 9 shows values of CK metrics for class diagram depicted figure 9.

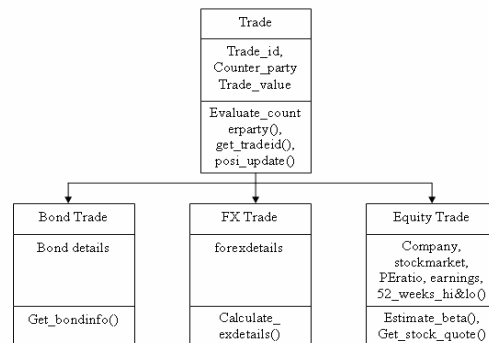


Figure 9. Object Oriented Design for Trade classification program

Table 9. Values of CK metrics for Trade classification program.

Classes	WMC	DIT	NOC	CBO	RFC
Trade	3	0	3	1	4
Bond Trade	1	1	0	0	1
FX Trade	1	1	0	0	1
Equity Trade	2	1	0	2	3

Figure 10 shows a graph of reusability vs classes of trade classification class diagram

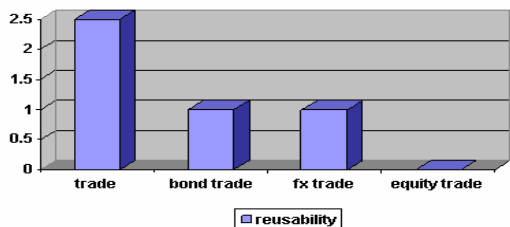


Figure 10. Reusability of various classes of trade program
 Table 10 depicts the reusability of all classes and reusability of class diagram vehicle classification.

Class name	Class reusability	Reusability of class diagram
Trade	2.5	2.5
Bond Trade	1	
FX Trade	1	
Equity Trade	0	

The reusability of all Class diagrams is compared in figure 11.

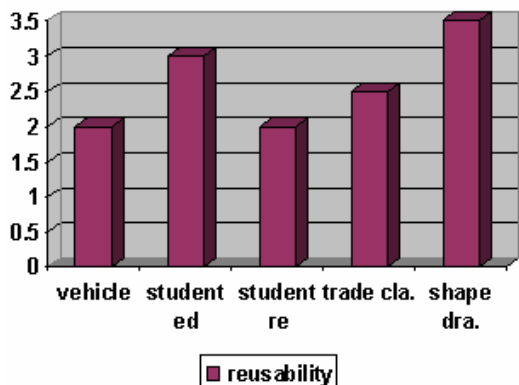


Figure 11. reusability of various programs

CONCLUSION

Reusability increases with increase in DIT and NOC but it decreases with increase in CBO. In this paper, we have proposed an approach to measure the reusability of object oriented program based upon CK metrics (DIT, NOC & CBO). Since reusability is an attribute of software quality, we can quantify software quality by measuring software reusability. Hence, this approach is important to measure reusability of class diagram.

REFERENCES

- 1.C. Terry and D. Dikel. Reuse library standards aid users in setting up organizational reuse programs, Embedded Systems Programming Product News, 1996.
- 2.Anthes, Gary II. , “Software Reuse Plans Bring Paybacks,” *Computerworld*, Vol. 27, KO. 49, pp.73, 76.

- 3.McCabe and Associates, *Using McCabe QA 7.0*, 1999, 9861 Broken Land Parkway 4th Floor Columbia, MD 21046.
- 4.McCabe, T. J., “A Complexity Measure”, *IEEE Transactions on Software Engineering*, SE-2(4), pages 308-320, December 1976.
- 5.Lorenz, Mark & Kidd Jeff, *Object-Oriented Software Metrics*, Prentice Hall, 1994.
- 6.Tegarden, D., Sheetz, S., Monarchi, D., “Effectiveness of Traditional Software Metrics for Object-Oriented Systems”, *Proceedings: 25th Hawaii International Conference on System Sciences*, January, 1992, pp. 359-368.
- 7.Rosenberg, L., and Hyatt, L., “Software Quality Metrics for Object-Oriented System Environments”, Software Assurance Technology Center, Technical Report SATC-TR-95-1001, NASA Goddard Space Flight Center, Greenbelt, Maryland 20771.
- 8.Chidamber, S. R. & Kemerer, C. F., “A Metrics Suite for Object Oriented Design”, *IEEE Transactions on Software Engineering*, Vol. 20, #6, June 1994.
- 9.Liang,V., and Colemon, C., “Principal Components of Orthogonal Object Oriented Metrics”, Software Assurance Technology Center, White Paper SATC-323-08-14, NASA Goddard Space Flight Center, Greenbelt, Maryland 20771.
- 10.Jefferey S. Poulin “Measuring Software Reusability”, *IEEE Transactions on Software Engineering*, 0-8186-6632, pages 126-138, March 1994.
11. Rosenberg, L., “Metrics for Object-Oriented Environment”, EFAITP/AIE Third Annual Software Metrics Conference, 1997.
- Tang, M., Kao, M., and Chen, M., “An Empirical Study on Object- Oriented Metrics”- *IEEE Transactions on Software Engineering*, 0-7695-0403-5, 1999.