ISSN: 2348 - 2117

International Journal of Engineering Technology and Computer Research (IJETCR)
Available Online at www.ijetcr.org
Volume 3; Issue 3; May-June; Page No. 52-56

A Critical Evaluation of LI Metric using Weyuker's Properties

Rashmi Gupta

Faculty of Computer Science Department, Amity University Haryana

goyal.rashmi18@gmail.com

Abstract:
In the modern age of Computerization Object Oriented Paradigm is becoming more and more prominent. This has stimulated the need of high quality software, as the traditional metrics cannot be used in the object-oriented systems. Although CK suit of metric is the widely accepted metrics but when analyzed, according to their validation criteria on which these are based, these metrics can't satisfy certain axioms. This paper gives the evaluation of LI suit of metrics and suggests the refinements to CK metrics so that these metrics should reflect accurate and precise results for OO based systems.

Keywords: LI metric, Weyuker's axioms.

1. Introduction

The OO paradigm for the software development differs from traditional procedural counterpart so the traditional metrics can't be applied on OO software. Opponents of the use of traditional metrics within the OO paradigm argue that such metrics were originally designed to go along procedural methodologies and languages, and therefore fail to capture concepts as inheritance and polymorphism which are unique to the OO paradigm [7]. Moreover Traditional paradigm requires more effort during the coding and maintenance phases as compared to OO paradigm, which put more emphasis on the earlier stages of software development, especially on design phase [5]. Krishnan et al. empirically show that higher up-front investment in design helps in controlling costs as well as in improving quality [6]. Software metrics are measures of the attributes of software products and processes. Because of the importance of knowing quality of the design phase there is the need of metrics that measures the goodness of design and it provides the designer with improved insight that leads to a higher level of quality[3].

Li discovered some metrics as he discovered problems with Chidamber and Kemerer metrics during the course of defining the unit definition model for the metrics. An alternative metrics suite was proposed by Li [7]. Six metrics, Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling through Abstract Data Type (CTA), and Coupling through Message Passing (CTM) were proposed in order to overcome some limitations found in Chidamber and Kemerer metrics.

Even today, there is lack of availability of metrics that measures all aspects of OO systems. Hence additional metrics are required to measure currently unexplored or partially explored dimensions of OO systems. Research on metrics for OO software development is limited and empirical evidence linking the OO methodology and project outcomes is scarce [2].

The organization of the rest of the paper is as follows: In the next section, we discuss Weyuker's nine axioms and LI Suit of Metrics with the flaws and the inconsistencies in the suit that arose when validated by Weyuker's nine axioms. Section3 we also discussed about the refinements to the discrepancies in LI suit of metrics and propose two new metrics .In the In Section 4, we present the conclusion and finally the reference.

2: LITERATURE REVIEW

2.1. LI Suit of Metrics

One of the suites of OO design measures was proposed by Li and Henry. This suite of metrics claims that these measures can aid users in understanding object oriented design complexity and in predicting external software qualities such as software defects, testing, and maintenance effort. Even though this metric suite is widely, empirical validations of these metrics in real world software development settings are limited. Various flaws and inconsistencies have been observed in the suite of six class-based metrics as shown under.

2.2. Validation Criteria for LI Metric Suite

Weyuker establishes a standard for software measures in this seminal article. She states the conditions for a

Corresponding author: Rashmi Gupta

measure as follows:
"All the measures we consider depend only on the syntactic features of the program." [12]
The nine properties of measures are:

Property 1: Non-coarseness
Given a class A and a metric μ another class B can always be found such that: μ (A) ≠ μ (B). This implies that not every class can have the same value for a metric; otherwise it has lost its significance as a measurement.

Property 2: Granularity
According to this property there could be a finite number of cases having the same metric value. Since the universe of discourse deals with at most a finite set of applications, each of which has a finite number of classes, this property will be met by any metric measured at the class level.

Property 3: Design details are important
Given two class designs, A and B, which provide the same functionality, does not imply that μ (A) = μ (B). The specifics of the class must influence the metric value. The intuition behind Property 3 is that even though two class designs perform the same function, the details of the design matter in determining the metric value for the class.

Property 4: Monotonicity
For all classes A and B, the following must hold: μ(A) ≤ μ(A+B) and μ(B) ≤ μ(A+B) where A + B implies combination of A and B. This implies that the metric value for the combination of two classes can never be less than the metric for either of the component classes.

Property 5: Non-equivalence of interaction
∃ A, ∃ B, ∃ C, such that μ(A) = μ(B) does not imply that μ(A+C) = μ(B+C).
This suggests that interaction between A and C can be different than interaction between B and C resulting in different complexity values for A+C and B+C.

Property 6: Non-uniqueness (notion of equivalence)
There can exist distinct classes A and B such that μ (A) = μ (B). This implies that two classes can have the same metric value, i.e., the two classes can be equally complex.

Property7: Permutation of elements
Permutation of elements within the item being measured can change the metric value. The intent is to ensure that metric values change due to permutation of program statements.

Property 8: Renaming
It requires that when the name of the measured entity changes, the metric should remain unchanged.

Property 9: Interaction increases complexity
∃A and ∃B such that: μ(A) + μ(B) < μ(A+B)
The principle behind this property is that when two classes are combined, the interaction between classes can increase the complexity metric value.

2.3 Inconsistencies in LI metric suit and their possible solutions

The LI metrics were validated against Weyuker's nine axioms and most of the metrics was found to comply to either of the property 7 about "Permutation of elements" or property 9 about "Interaction increases complexity" of the above Weyuker list of axioms. In support of non-compliance to property 7 Li and Henry suggested that "Permutation of elements" property is meaningful in traditional program design, where the ordering of if- then-else blocks could alter the program logic (and consequent complexity). Failure to meet the property 9 suggest that it is probably not applicable to object –oriented systems where interaction might in fact decrease complexity by rendering classes closer to the abstractions they are supposed to portray. There are six metrics proposed:

2.3.1 Number of Ancestor Classes (NAC)

- NAC measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy.

2.3.2. Number of Local Methods (NLM)

- The Number of Local Methods (NLM) is defined as the number of local methods defined in a class which are accessible outside the class

2.3.3. Class Method Complexity (CMC)

- The Class Method Complexity (CMC) metric is defined as the summation of the internal structure complexity of all local methods.

2.3.4. Number of Descendent Classes (NDC)

- The definition of NDC is the total number of the descendent classes (subclass) of a class.

2.3.5. Coupling through Abstract Data Type (CTA)

- The Coupling through Abstract data Type (CTA) is defined as the total number of classes that are used as abstract data types in the data-attribute declaration of a class.

2.3.6. Coupling through Message Passing (CTM)

Coupling through Message Passing (CTM) is defined as the number of different messages sent out from a class to

other classes excluding the messages sent to the objects created as local objects in the local methods of the class.

3. Evaluation ofLi metric by Weyuker's properties:

Let us take an example for the evaluation of LI metrics:

```
Abstract class Shape
{
        Abstract double area (double a, double b);

        Void getName (String x)

        {

                System.out.println(x);

        }

}

Class Triangle extends Shape
{
        Double area ()
        {
                Return get Width () * get Height ()
        / 2;
        }
}
Class Rectangle extends Shape
{
        Double area ()
        {
                Return gets Width () * get Height ();
        }
}
Class ABC

{

   Public static void main (String arg [])

   {

        Triangle ob1 = new Triangle ();

System.out.println ("Area="+ ob1.area (3.5, 4.5));

        Rectangle ob12 = new Rectangle ();

        System.out.println ("Area="+ ob2.area
        (6.5, 8.5));

        ob1.getName ("Triangle");

        ob2.getName ("Rectangle");

   }

}
```

Table 1: Evaluation of the Li and Henry metrics against Weyuker's nine axioms

LI Metriccs	Weyuker's nine axioms								
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
NAC	Y	Y	Y	Y	Y	Y	Y	Y	N
NLM	Y	Y	Y	Y	N	N	N	Y	Y
CMC	Y	Y	Y	Y	Y	N	N	Y	Y
NDC	Y	Y	Y	Y	Y	Y	Y	Y	N
CTA	Y	Y	Y	Y	Y	Y	N	Y	N
CTM	Y	Y	Y	Y	Y	Y	Y	Y	Y

3.1. A New PROPOSED METRICS

In this section, a set of new metrics are proposed to measure the reusability of an object oriented codes.

3.1.1. Metric1 (NAC + NDC)

- The large number of ancestor classes for a particular class in the hierarchy, the greater the potential reuse of inherited methods
- In NDC metric data or methods are being inherited. This would increase the accuracy of complexity measure caused

by inheritance relations amongst classes

3.1.2. Metric2 (CTA + CTM)

- Two classes are coupled when one class uses the other class as an abstract data type. This metric gives the scope of how many other classes' a class needs in order to provide its own service to others.
- Coupling through references like parameter passing return type etc. have not been accounted for. Thus this metric cannot be independently used for measuring coupling.

3.2. Comparison of new proposed metrics with LI metrics with Respect to Weyuker's Properties:

Table 2: Comparison of new proposed metrics with LI metrics against Weyuker's nine axioms

Properties	NAC	NLM	CMC	NDC	CTA	CTM	Metrics1	Metric2
1	Y	Y	Y	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y	Y	Y	Y
3	Y	Y	Y	Y	Y	Y	Y	Y
4	Y	Y	Y	Y	Y	Y	Y	Y
5	Y	N	Y	Y	Y	Y	Y	Y
6	Y	N	N	Y	Y	Y	Y	Y
7	Y	N	N	Y	N	Y	Y	Y
8	Y	Y	Y	Y	Y	Y	Y	Y
9	N	Y	Y	N	N	Y	N	Y

4. Conclusion

Two new object-oriented metrics Metric1 and Metric2 have been proposed in this paper. These proposed metrics are evaluated with Weyuker's properties and also compared with other well known object-oriented LI metrics. It has been found that the proposed metrics Metric1 and Metric2 better represent the complexity of a class and program due to inheritance.

So as the two proposed versions of Li metric are able to alleviate the problems. In the Metric1 the ninth property still not is satisfied. Hence the second version is able to solve both the problems that the original LI metric was not able to solve so it can become the true candidate of the rectified version of LI metric as far as the cohesiveness of a class is concerned.

As a future work, these metrics can be used for fault prediction in a quantitative way. It is also planned to develop a tool to calculate the proposed metrics.

5. References

1. Basili, V., Briand, L., and Melo, W. "A Validation of Object Oriented Design Metrics as Quality Indicators", *IEEE Trans. Software Engineering*. vol. 22, 1996.
2. Chidamber, S.R., and Kemerer, C.F. "A Metric Suite for Object Oriented Design," *IEEE Trans. Software Engineering*, Vol. 20, 1994.
3. Chidamber, S.R., and Kemerer, C. F. "Towards a Metrics Suite for Object Oriented Design," *Proc. Conf. Object Oriented Programming Systems, Languages, and Applications (OOPSLA'91),* vol. 26, no. 11, 1991.
4. Henderson-Seller, B., and Constantine, L. L. "Coupling and Cohesion (towards a valid metrics suite for object oriented analysis and Design", *Object Oriented Systems, 3*, 1996.
5. Henderson-Sellers, B. "Some metrics for object oriented software engineering," In J. Plotter, M. Tokoro, and B. Meyer, editors, *TOOLS 6: Technology of Object-Oriented Languages and Systems*, Englewood Ciffs, N. J., 1991, Prentice Hall TOOLS Conference Series.
6. Krishnan, M. S., Kriebel, C. H., Kekre, S., and Mukhopadhyay, T. "An Empirical Analysis of Productivity and Quality in Software Products,"

Management Science, vol. 46, 2000.

7. Li, W., Henry, S., Kafura, D., and Schulman, R. "Measuring Object- Oriented Design, *"Journal of Object Oriented Programming,* July-August 1995.

8. Lorenz, M., and Kidd, J. *Object – Oriented Software Metrics*, Prentice Hall, Englewood Cliffs, N. J., 1994.

9. Mayer, T., and Hall, T. "Critical Analysis of Current OO Design Metrics", *Software Quality Journal*, 8, 1999.

10. Pressman, R. "A Practitioner's Approach to Software Engineering," Mc-grawhill Publications, 2001.

11. Sheldon, F. T., Jerath, K., and Chung, H. "Metrics for Maintainability of Class Inheritance Hierarchies", *Journal of Software Maintenance*, issue 3, May 2002.

12. Weyuker, E. "Evaluating software complexity measures," *IEEE Trans. Software Eng*., 14, 1988.