International Journal of Information Technology and Knowledge Management
July-December 2009, Volume 2, No. 2, pp. 449-454

DYNAMIC METRICS AT DESIGN LEVEL

Payal Khurana & Puneet Jai Kaur

As object oriented analysis and design techniques become widely used, the demand on assessing the quality of object-oriented designs substantially increases. Recently, there has been much research effort to develop and empirically validate metrics for OO design quality. Complexity, Coupling and Cohesion have received a considerable interest in the field. Despite the rich body of research and practice in developing design quality metrics, there has been less emphasis on dynamic metrics for object-oriented designs. The complex dynamic behavior of many real-time applications motivates a shift in interest from traditional static metric to dynamic metrics.

In this research, Improved Dynamic cohesion metric is proposed. Existing cohesion measures only consider the pattern of interactions but do not differentiate write interaction from read interaction. Thus, do not reflect properties of class. This research measures improved version of cohesion measures considering read and write interaction as well as dynamic environment. This cohesion measures will be evaluated against some open source code of Java and to prove that write interaction are so commonly used in classes that they have much influence on cohesion measures and also cohesion measured dynamically is better than measured statically.

Keywords: Dynamic Measurement, Cohesion, Interaction Patterns, Read/Write Interactions, AOP.

1. Introduction

Recent years have seen the increasing use of object oriented paradigm in software development. The use of object-oriented software development technique introduces new elements to software complexity both in software development process and in the final product.

Software metrics measure different aspects of software complexity and therefore play an important role in analyzing and improving software quality [10, 13]. They provide useful information on external quality aspects of software such as maintainability, reusability, and reliability, and provide a means of estimating the effort needed for testing.

Traditional metrics for measuring software such as Lines of Code (LoC) have been found to be inadequate for analysis of object-Oriented software [9]. In recent years many researchers and practitioners have proposed a number of static code metrics for object-oriented software, e.g the suite of metrics proposed by Chidamber and Kemerer [16, 17]. These code metrics quantify different aspects of complexity of the source code. However the ability of such static metrics to accurately predict the dynamic behavior of an application is as yet unproven.

Static metrics alone may be insufficient in evaluating the dynamic behavior of an application at run time, as its behavior will be influenced by the operational environment as well as complexity of object-oriented software.

* Department of Information Technology, Panjab University, Chandigarh, INDIA.
E-mail: payal_f12@yahoo.co.in, puneetkaur79@yahoo.co.in

Cohesion refers to the relatedness of the elements in a module. A highly cohesive module is one whose elements have tight relationship among them in order to provide a single functionality of the module. On the contrary, a low cohesive module has some elements that little relation with others, which indicates that the module may contain several unrelated functionalities. It is widely accepted that higher the cohesion of a module is, the easier the module is to develop, maintain and reuse.

In the object oriented paradigm, various cohesion measures for [4, 11, 14, 15, 16, 17, 18] classes have been proposed. In the beginning of research on cohesion measures for classes, researchers just considered syntactic relationship between class members such as interaction between class members such as interactions between instance variables and methods. However more recent researches have been tried to identify inherent characteristics of classes which can affect the cohesiveness of classes and incorporate them into cohesion metrics.

Chae *et.al* [4] introduced the notion of special methods. They noted that special methods have no influence on class cohesion because those methods are designed to show a specific behavior, interacting inherently with only some of instance variables for the specific purposes. They attempted to enhance the existing cohesion measures by including the implicit and hidden interactions between class members due to data dependency [5].

Gyun woo *et.al*. [1] proposed an approach to enhancing the existing cohesion measures by considering type of interaction between method and instance variable.

The interaction between method and instance variable can be classified into two categories read interactions and write interactions. Write interaction is considered stronger than read interaction because the write interaction can affect other methods that read the instance variable written. These interactions are given weights. More weight is given to read interaction. Using these weights cohesion measures are revised.

Still, these revised cohesion measures are static cohesion measures. In this research paper these revised measures are made dynamic for the some student Java programs and compared with static ones.

Java is general-purpose object-oriented programming language [12] developed by Sun Microsystems. As it is being increasingly used as a development language for new software products, it is important to have a means of evaluating the quality of such products.

2. RELATED WORK

Briand *et. al.* [8] carried out an extensive survey of current available cohesion literature in object-oriented systems and concluded that all the current metrics measured cohesion at class level (static analysis). No measures of the object level cohesion had been proposed (Dynamic analysis). They suggested that the reason for this obstacle of determining the degree of cohesion within individual objects. They proposed that a way of evaluating these would be to find some method of instrumenting the source code to log all occurrences of object instantiations, deletions, method invocations, and direct reference to attributes while the system is executing even though no methods of measuring cohesion at runtime for object-oriented system has been proposed, a number of researchers were found to be investigating applying at other stages of the software life-cycle [6].

Gupta conducted a study and Rao comparing a program execution based approach of measuring the levels of module cohesion present in legacy software, with a static based method [3]. The results from this study showed that static based approach significantly overestimated the level of cohesion present in the software tested, indicating that a dynamic measurement would prove useful.

Power *et. al.* [2] measured new dynamic class level cohesion metrics suitable for the run-time evaluation of a program. These dynamic cohesion metrics are then applied to assess the quality of java programs from the java Grande Forum Benchmark Suite and the SPECjvm98 Benchmarks. They to see if the results bear any relation to those obtained from static analysis also did an investigation.

3. DYNAMIC COHESION METRICS

Basic Definitions: Here are some basic definitions of coherency weights to incorporate the impact of write interactions into cohesion measures. Basically weights are assigned to every edge of the member interaction graph of a class then weights of method pairs are derived from these edge weights.

$V(c)$- Total number of variables in a class *C*.

$M(c)$ - Total number of methods in a class *C*.

$M_R(v)$ - Set of methods of class *C* that directly read *v*.

$M_W(v)$ - Set of methods of class *C* that directly write *v*.

$M_U(v)$ - Set of methods of class *C* that directly read or write *v*.

$$M_R(v)\ U\ M_W(v) = M_U(v)$$

$$M_R(v) = \{m\ ª\ M(c) |\ v\ ª\ V_R(m)\}$$

$$M_W(v) = \{m\ ª\ M(c) |\ v\ ª\ V_W(m)\}$$

$$M_U(v) = \{m\ ª\ M(c) |\ v\ ª\ V_U(m)\}$$

1. Absolute Coherency Weight:

Absolute coherency weight between method $m(m\ ª\ M(c))$ and variable $v\ ª\ V(c)$ is given as

$$W_{MVabs}(m,v) = \begin{cases} 1 + M_R(v) - \{m\} & \text{if } m \in M_W(v) \\ 1 & \text{if } m \in M_R(v)\ \&\ m \notin m M_W(v) \\ 0 & \text{if } m \notin M_U(v) \end{cases}$$

For a class *C* It is maximum possible coherency weight of $W_{MV\text{abs}}$ provided that the set of variables & set of methods are not changed.

$$W_{\max} = \begin{cases} M(c) & \text{when we consider the impact of write interactions} \\ 1 & \text{otherwise} \end{cases}$$

3. Coherency Weight between *m* and *v*, $W_{MV}(m, v)$

For a method $m ª M(c)$ and an instance variable $v ª V(c)$ Coherency weight is given as

$$W_{MV}(m, v) = W_{MV\text{abs}}(m, v) / W_{\max}(c)$$

4. Absolute Coherency Weight between Method Pairs *mi* and *mj* $W_{MM\text{abs}}$

$$W_{MMabs}(m_i, m_j) = \begin{cases} 0 & \text{if } V_U(m_i) \cap V_U(m_j) = \Phi \\ max(W_{com}(m_i, m_j)) & \text{otherwise} \end{cases}$$

Where $W_{com}(M) = (W_{MV\text{abs}}(m, v) |\ m ª M, v ª \cap V_U(m)$ where $m ª M(c))$

5. Coherency Weight between Method Pairs m_i and m_j, W_{MM}

$$W_{MM}(m_i, m_j) = W_{MM\text{abs}}(m_i, m_j) / W_{\max}(C)$$

3.1 Dynamic Revised Cohesion Measures:

a) $LCOM1_w(c)$:

Since *LCOM*1 counts the number of unrelated pairs of methods, so *LCOM*1 can be computed by subtracting the number of related pairs of methods from the whole number of distinct method pairs. In order to improve *LCOM*1, Coherency weight of a method pair is considered to determine its relatedness. As a result less value is subtracted for less related pairs from total number of related pairs.. Revised version of *LCOM*1 is

$$LCOM1_w(C) = |Mp(C)| - \Sigma W_{MM}(m_i, m_j)$$
$$(m_i, m_j) \in Mp(C)$$

b) $LCOM2_w(c)$

In the original *LCOM*2, number of related pairs of methods are subtracted from the number of unrelated method pairs. Revised version of *LCOM*2 is.

$$LCOM2_w(c) = \begin{cases} |Mp(c)| - 2X\Sigma W_{MM}(m_i, m_j) & \\ \qquad (m_i, m_j) \in Mp(c) & \text{if } |P| > |Q| \\ 0 & \text{otherwise} \end{cases}$$

where $|P|$ is number of unrelated method pairs and $|Q|$ is number of related method pairs. For the case $|P| > |Q|$.

c) $TCC_w(C)$

The original *TCC* considers the two methods are related if the share some common instance variable in use. The revised version of *TCC* is defined as the ratio of the sum of the coherency weight of every pairs of methods to the number of method pairs.

$$TCC_w(C) = \frac{2X\Sigma W_{MM}(m_i, m_j)}{(m_i, m_j) \in Mp(c)} \Big/ \frac{}{|M(c)| \times |M(c) - 1|}$$

4. PROFILER IMPLEMENTATION

One way to collect the read write interactions of methods with variable is to intercept the joint points, well defined point in the program, where the variables of the class are read or write by the method on invocation.

A trivial implementation of the idea finding read write interactions is to insert intercepting code at the appropriate places in the program. Modifying existing code implies extra costs of testing and maintenance. To avoid this, new AOP (Aspect Oriented Programming) techniques are used. The interceptive code is developed as an interdependent programming unit and merged it with the target program using a weaving tool. In AOP [19, 20], intercepting code constitutes an aspect or concern whose code crosscuts that of the system's core concerns. For our programs in Java, we used Aspect *J* to implement intercepting code and then insert it at appropriate joint points. Aspect *J* is an extension of java with new language constructs such as pointcuts, advices and aspects that allow the separation of aspects from core concerns.

The interceptive code consists of four pointcuts. These pointcuts intercepts the objects which are instantiated, methods which are executed and the variables which are referenced and written. Figure 1 shows snapshots of the output of a student program while program is in running state. It shows the line number, class whose object is instantiated, method called and fields which are referenced or written.

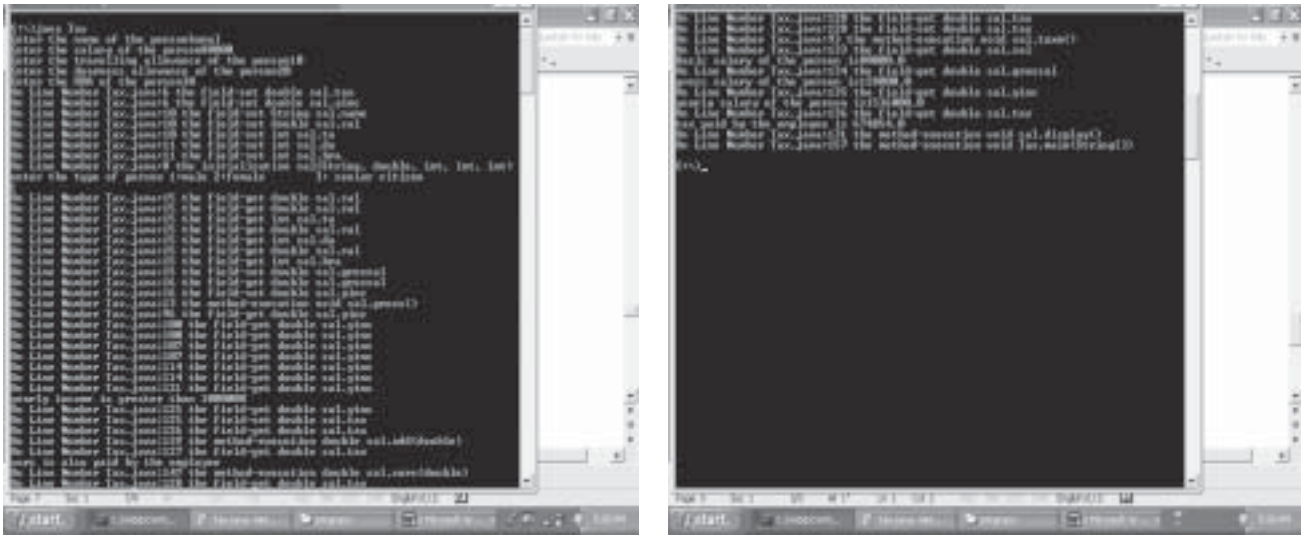

Figure 1: Snapshots of Dynamic Analysis of a Java Program

5. STATIC ANALYSIS

Here are interaction patterns formed, for sample program, from the data collected (read/write interactions) from the dynamic analysis of the java programs. From these interaction patterns weights are calculated which are used to find the revised cohesion measures.

5.1 Interaction Patterns (Dynamically)

Statically, in the interaction pattern, all the cases or conditions are considered collectively whereas dynamically interaction patterns are made case by case basis depending upon the input given by the user at run time.

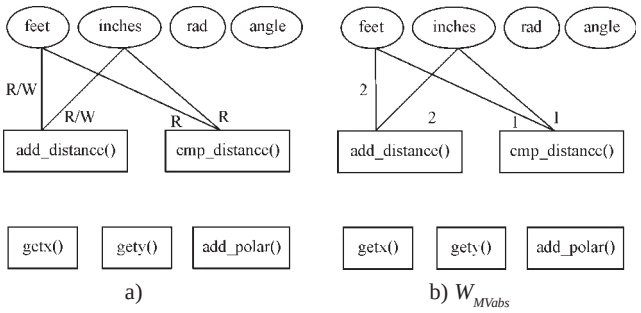

Figure 2: a) Interaction Pattern
b) Absolute Coherency Weight between Method & Variable

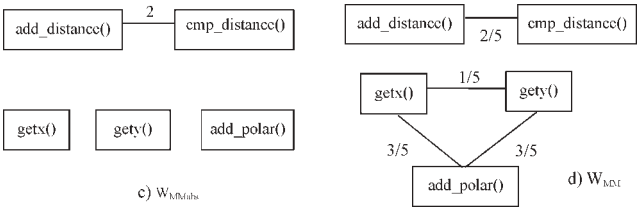

Figure 2: c) Absolute Coherency Weight between Method Pairs
d) Coherency Weight between Method Pairs

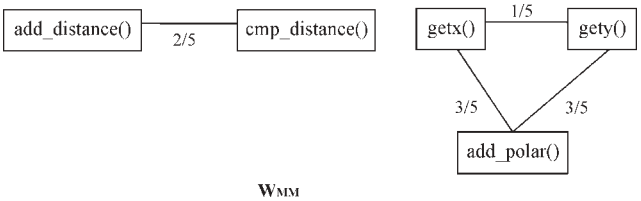

Figure 3: Absolute Coherency Weight between Method Pairs

6. EMPIRICAL VALIDATION

For empirical validation five student java programs are taken as Input. Read/Write interactions are found from the dynamic analysis of the programs which are used in the interaction patterns to find weights and finally, the revised cohesion measures $LCOM1_w(C)$, $LCOM2_w(C)$ and $TCC_w(C)$.

LCOM1w(C):

Table 1

Programs	*Dynamically*	*Statically*
P1	3.7	3.7
P2	7.5	5
P3	9.1	8.2
P4	14.5	13.2
P5	19	14.7

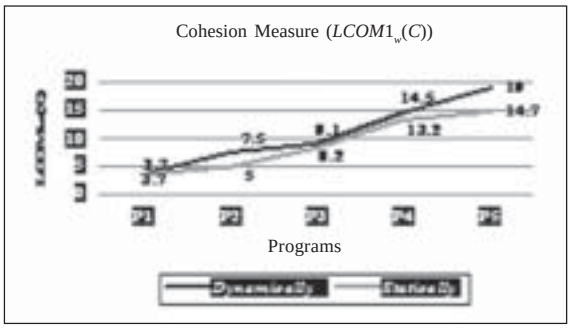

Figure 4: Comparison of Dynamic and Static $LCOM1_w(C)$

LCOM2w(C):

Table 2

Programs	*Dynamically*	*Statically*
P1	1.5	1.5
P2	5	0
P3	8.2	6.4
P4	14.1	11.3
P5	17	8.4

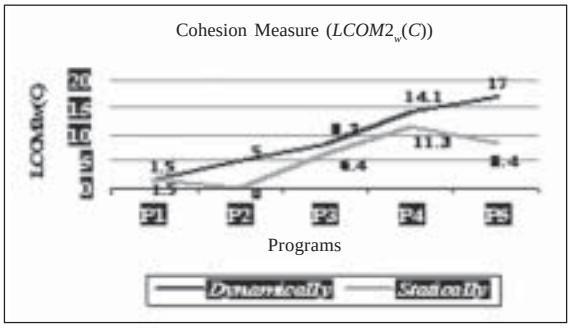

Figure 5: Comparison of Dynamic and Static $LCOM2_w(C)$

TCCw(C):

Table 3

Programs	*Dynamically*	*Statically*
P1	0.4	0.4
P2	0.2	0.5
P3	0.1	0.2
P4	0.03	0.1
P5	0.1	0.3

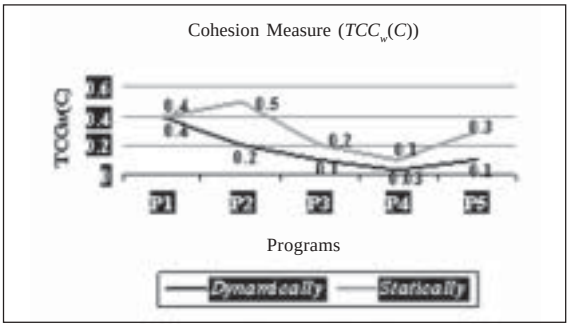

Figure 6: Comparison of Dynamic and Static $TCC_w(C)$

6.2 Analysis of Results:

Results shows that, For $LCOM1w(c)$ & $LCOM2w(c)$ dynamic values are more than static values but for $LCOM3w(c)$ dynamic values are less than static values because cohesion values are scaled by total number of edges in the interaction pattern due to method-method interaction.

Above comparison reveals that cohesion measured dynamically can also be equal to static value. Static values can be zero also as shown by $LCOM2w(c)$ i.e lack of cohesion is zero. Graphically dynamic curve lies above the static curve. So it is concluded from the values programs are less cohesive when these values are measured dynamically and more cohesive when measured statically.

7. CONCLUSION AND FUTURE SCOPE

In this paper, we presented the comparative analysis of dynamic cohesion metric (revised) and static cohesion metric. To perform the analysis first of all dynamic analysis of the Java programs is done using language AspectJ. The output of dynamic analysis, Read/Write interactions is used to form the interaction patterns. These Read/Write interactions are given weights and using these weights revised cohesion measures $LCOM1w(C)$, $LCOM2w(C)$ & $TCCw(C)$ are found both dynamically & statically.

Above results shows that dynamic cohesion values are greater than static cohesion values. So it shows that program is less cohesive when cohesion is measured dynamically rather than statically.

Dynamic cohesion values depends upon the input values provided to the program because depending upon input values corresponding methods are called so the corresponding method-variable interaction & method-method interaction values are used in cohesion measure. Dynamic cohesion values can be equal to the static cohesion values but can not be less than static values.

As a future work, this work can be extended for the other cohesion measures also. Revised coupling metrics can also be found based on the Read/Write interactions and these coupling metrics can also be made dynamic. Relationship between the revised cohesion measures and other quality attributes such as fault proneness can also be explored.

REFERENCES

[1] Gyun Woo, Heung Seok Chae, Jian Feng Cui, Jeong-Hoon Ji, Revising Cohesion Measures by Considering the Impact of Write Interactions between Class Members, Science Direct, *Journal of Information and Software Technology* (2008).

[2] Mitchell, A. and Power, J. F., Towards a Definition of Run-Time Object-Oriented Metrics" Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE' 2003), Darmstadt, Germany, (2003).

[3] Gupta, N. and Rao, P. "Program Execution Based Module Cohesion Measurement,"16th International Conference on Automated on Software Engineering (ASE '01), Saniego, USA, (2001).

[4] H. S. Chae, Y. R. Kwon, D. H. Bae, A Cohesion Measure for Object-Oriented Classes, Software Practice and Experience **30**(12) (2000) 1405–1431.

[5] H. S. Chae, Y. R. Kwon, D. H. Bae, Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables, *IEEE Transactions on Software Engineering* **30**(11) (2004) 826–832.

[6] Cleland–Hunang J., Chang C. K., Kim H. and Balakrishnan A. "A Requirements–Based Dynamic Metric in Object–Oriented Systems" IEEE Proceedings on 5th International Symposium on Requirements Engineering, (2001) 212–219.

[7] Bansiya J., Etzkorn L., Davis C. and Li W.: "A Class Cohesion Metric for Object-Oriented Designs", *The Journal of Object-Oriented Programming,* **11**(8) (1999) 47-52.

[8] Briand, L. C., Daly, J. W. and Wust, J. K., "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," *Empirical Software Eng.* : An Int'l J., **3**(1) (1998) 65–117.

[9] Fenton, N. E. and Neil M., "Software Metrics: Successes Failures and New Directions," *The Journal of Systems and Software*, **47** (1999) 149–157.

[10] L. C. Briand, J. W. Daly, J. Wüst, A Unified Framework for Cohesion Measurement in Object-Oriented Systems, *Empirical Software Engineering* **3**(1) (1998) 65–117.

[11] Wei Li Another Metric Suite for Object-Oriented Programming. *The Journal of Systems and Software,* (1998).

[12] Java 2 Platform, available at *Sun Microsystems: http:// www.java.sun.com.*

[13] 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) (1996) 751–761.

[14] Bieman J. M., Kang B. K. Cohesion and Reuse in an Object-Oriented System. Proceedings of ACM Symposium on Software Reusability, (1995) 259–262.

[15] Hitz M., Montazeri B. Measuring Coupling and Cohesion in Object-Oriented Systems. Proceedings of International Symposium on Applied Corporate Computing, (1995).

[16] Chidamber, Shyam R. and Kemerer, Chris F., "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering , **20**(6) (1994).

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

[18] H. Sellers, *Software Metrics*, Prentice-Hall (1996).

[19] AspectJ Home Page at: *http://www.eclipse.org/aspectJ/*.

[20] Aspect Oriented Software Development (AOSD) Research Projects Located at: *http://www.aosd.net/technology/research.php*.