Principal Component Analysis of Coupling Measures for Developing High Quality Object Oriented Software

K. M. Azharul Hasan
Khulna University of Engineering & Technology
Khulna-9203, Bangladesh
azhasan@gmail.com

Mohammad Sabbir Hasan
Khulna University of Engineering & Technology
Khulna-9203, Bangladesh
shabbir_cse03@yahoo.com

Abstract— **The object oriented programming paradigm often claimed to allow a faster development pace and higher quality of software. Within the design model, it is necessary for design classes to collaborate with one another. However, collaboration should be kept to an acceptable minimum i.e. better designing practice will introduce low coupling. If a design model is highly coupled, the system is difficult to implement, to test and to maintain overtime. In case of enhancing software, we need to introduce or remove module and in that case coupling is the most important factor to be considered because unnecessary coupling may make the system unstable and may cause reduction in the system's performance. So coupling is thought to be a desirable goal in software construction, leading to better values for external software qualities such as maintainability, reusability and so on. To test this hypothesis, a good measure of class coupling is needed. In this paper, the major issue of coupling measures have been analyzed with the objective of determining the most significant coupling measure.**

Keywords- Object Oriented Systems, Software Metrics, Coupling, Cohesion, Software Measures, and Principal Component Analysis.

I. INTRODUCTION

Object-oriented (OO) system development is gaining wide attention both in research environment and in industry. A severe problem is encountered, however, is the quickly increasing complexity of such systems and the lack of adequate criteria and guidelines for good designs. To cope with this problem it is imperative to better understand the relations and interactions of the modules of object-oriented systems. The interactions between the internal components of software provide several goals in software construction leading to better values for external attributes such as maintainability, reusability, and reliability. Coupling is the internal software attribute, which tells how tightly the components of software modules are bound together in design or implementation. Good OO software should obey the principle of low coupling and high cohesion. Strong coupling makes a system more complex, highly inter-related modules are harder to understand or change. Designing the systems with the weakest possible coupling between modules can reduce complexity. A software designer must, therefore, strive to minimize coupling to reduce the risk of error propagation across modules [1]. In this paper an experimental analysis is presented for selecting most responsive coupling measure to answer the question "which coupling measure a project should use, and which one shows the most perfect interactions between components in the object oriented environment?"

In OO paradigm coupling describes the interdependency between methods and between object classes, respectively [2][3]. Stevens *et al.*, who first introduced coupling, in the context of structured development techniques, define coupling as "the measure of the strength of association established by a connection from one module to another"[4]. Braind et al. defined some properties to be satisfied by the coupling measure for empirical validation [5]. There are differences between necessary and unnecessary coupling. The rationale is that without any coupling the system is useless. Consequently, for any given software solution there is a basic or necessary coupling level. Such unnecessary coupling does indeed needlessly decrease the reusability of classes. There is also static and dynamic coupling measure for object oriented systems [8]. As the polymorphic method invocation is determined by run time, the coupling on this method belongs to dynamic coupling.

After years of research, many coupling metrics have been proposed to find the inter module dependencies. In this paper a study on software evaluation with the help of coupling metrics has been performed. We have categorized the coupling measures in two categories namely ratio oriented and ratio less. To validate the interactions we have selected three different types of industrial software for case study. For analyzing the coupling measures for object oriented system based on different interactions of the classes of the system a parser is developed named "*Design Analyzer*" to find the design pattern of the program. Such design analyzer is useful to get internal software architecture of a software developed in Object Oriented Programming Paradigm[7]. No modification was done in the software during the analysis presented in this paper. Finally, using principal component analysis [6] the two categories of measures are analyzed for selecting the most responsive coupling measure.

The rest of the paper is organized as follows. Section 2 gives an over view of selected industrial software, Section 3 has the overview of some existing coupling measures, section 4 presents the issues to be considered for software design taking example of the existing coupling measures, section 5 shows the results and discussion and finally, some conclusions are outlined in section 6.

II. Description of the software selected for Case study

In this section brief descriptions of the software that are taken as case study are given.

Software 1-Turtle Chat: This is chatting software like Yahoo! Messenger. This software can send and receive instant messages over Internet Protocol. There are two parts of this software: Server Part and Client Part. Here the Client side of contains **20** user defined classes and the Server side contains 4 user defined classes.

Software 2-Com Chat: This is also chatting software, which sends and receives data through the communication port of a personal computer. The software uses Java Communication API. The main purpose of this software is to simulate the seven layers of the OSI Model. The software implements Broadcasting, Checksum and routing techniques. This software contains **11** user defined classes.

Software 3-Admission Test Management System: This is mainly database software which can be used for admission test management of any university by storing information of the applicants, information of allowed candidate for examination, merit list of selected candidates, waiting list and so on. This software contains **13** user defined classes.

III. The Coupling Measures

In this research work the following couplings metrics have been chosen for the analysis of inter module dependencies. A brief definition of the measures is given here. All the measures determine the coupling between components. A survey of the metrics can be found in [10][11].

1. **Number of used classes by dependency relation (NUCD)**: This measure is used to count the total number of distinct classes with whom a particular class is creating dependency relation. Only one evidence for dependency relation would be enough, caused by any of dependency types (e.g. parameter, local variable, return type) to recognize the dependency between two classes.

2. Total number of evidences for "Used classes by dependency relation" (TNUCD): This measure is used to count total number of evidences for a particular class of "Used classes by dependency relation." All types of dependencies (e.g. parameter, local variable, return type) will be used to count such evidences.

3. Ratio of NUCD to TNUCD (RNUCD): This metric measures the ratio between TNUCD and NUCD only for the classes where NUCD count is greater than 1.

RNUCD=(TNUCD/NUCD) [NUCD>1]

4. Number of user classes for a class through dependency relation (NUCC): This measurement represents the total number of distinct classes who are using a particular class through dependency relations.

5. Total number of evidences for "User classes through dependency relation" (TNUCC): This measurement counts the total number of usage evidences of a particular class by the other classes in OO design.

6. Ratio of NUCC to TNUCC (RNUCC): This metric measures the ratio between TNUCC and NUCC only for the classes where NUCC count is greater than 1.

$$RNUCC = TNUCC / NUCC \quad [\mathrm{NUCC} > 1]$$

7. Class Coupling: The Class Coupling (CLC) is the summation of Client Coupling and Server Coupling of the class. It is the measure of the summation of out degree and in degree of a node in Class Class Interaction Graph(CCIG).

Class Coupling= Client Coupling (CC) + Server Coupling (SC).

The number of Coupling Relations for which a class is a client to other class is called Client Coupling for the class. It is the measure of out degree of a node in CCIG. The number of Coupling Relations for which a class is a server to other class(s) is called Server Coupling for the class. It is the measure of in degree of a node in CCIG.

8. Visible Member: The measure "Visible Member" shows the amount of members (attributes and methods) visible to other class numerically. This measure is used to find the over all members which can be called or used by other classes. Visible members are the most required criteria for direct coupling

9. Attribute Hiding Factor (AHF): AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration.

10. Method Hiding Factor (MHF): MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration.

11. Ratio of Public Attribute: The Ratio of Public Attribute is defined as the ratio of the sum of the public attributes in the class to the total number of attribute. Greater the public attributes in a class greater the chance of getting unexpected value if the attributes are used by several running process of different classes.

IV. Criteria of Measuring Coupling

For analysis purpose we divided the coupling metrics into two categories:

- Metrics without ratio: This category includes the metrics NUCD, TNUCD, NUCC, TNUCC, Class Coupling and Visible member. In the following these metrics are called as **Category1** coupling metric.
- Metrics with ratio. This category includes those metrics that are developed using a ratio of two quantities namely Ratio of NUCD and TNUCD, Ratio of NUCC and TNUCC, AHF, MHF, Ratio of public factor. In the following these metrics are called as **Category2** coupling metric.

The metric categorizing is necessary because the Principal Component Analysis will produce undesirable values as the magnitude that are found in these two categories of metrics are different.

A. Experiment Design

To make a study of coupling measure we want to determine the best coupling metrics defined above. We are going to apply these measures on the software under observation. The above measures are applied on these softwares through principal component analysis.

Principal Component Analysis:
Principal component analysis [6] is typically used to reduce the dimensionality and/or to extract new uncorrelated features from the original data. Principal component analysis involves an Eigen analysis on a covariance matrix. If the input data is represented as a matrix **X** of 'n' rows and 'm' columns:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ . & . & . & . \\ . & . & . & . \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}$$

Where n = total number of classes and m = total measures.
Then, the sample mean μ_i is computed for each column, where

$$\mu_i = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \qquad \text{for } j = 1, 2 \ldots m.$$

Then X can be centered to form X^*

$$X^* = \begin{bmatrix} x_{11}-\mu_1 & x_{12}-\mu_2 & \cdots & x_{1m}-\mu_m \\ x_{21}-\mu_1 & x_{22}-\mu_2 & \cdots & x_{2m}-\mu_m \\ . & . & \cdots\cdots & . \\ . & . & \cdots\cdots & . \\ x_{n1}-\mu_1 & x_{22}-\mu_2 & \cdots\cdots & x_{nm}-\mu_m \end{bmatrix}$$

Then covariance matrix $R = (1/n)[X^*]^T X^*$ is computed. An Eigen analysis on the covariance matrix R yields a set of positive Eigen values $\{\lambda_1, \lambda_2, \ldots\ldots\ldots, \lambda_m\}$ [6]. If the Eigen values are sorted in descending order (i.e., $\lambda_1 > \lambda_2 > \ldots\ldots > \lambda_m$), their corresponding Eigen vectors, $\{v_1, v_2, \ldots., v_m\}$, are the principal components. The first principal component retains the most variance, if the feature vectors are projected onto the first principal component, more variance will be retained than if the vectors are projected onto any other principal component. The second component retains the next highest residual variance, and so on. A smaller Eigen value contributes much less weight to the total variance. In many cases, the first few components can retain nearly all of the variance. If the 'd' most significant principal components are selected for projection of the data, then the variance (V) retained by this approximation is [6]:

$$V = \frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{m} \lambda_i}$$

V is also called the degree of accuracy for the approximation.

B. Finding the Design Pattern For Java Based Object Oriented Systems

Gamma at el [9] defines design patterns as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." A design pattern names, abstracts, and identifies the key aspects of a common design structure that makes it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and their instances, their roles and collaborations, and the distribution of responsibilities. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved In this paper, a design pattern named *Design Analyzer* is developed that finds the interactions, roles, collaborations and relationships of the software in a graphical format. These interactions can occur in there different ways between the classes in a Java based object-oriented software. These inter module relationships are as follows

1. Inter-module relationship through Return Type
2. Inter-module relationship through Argument Passing in a member function
3. Inter-module relationship through Object Declaration
4. Inter-module relationship through Inheritance

From these four types of relationship, we have defined two types of interactions as follows

The Operation-Operation interaction (O-O) is defined as the interaction between two operations of two or more different objects or classes. Let O_C be an operation of class C. There is an operation-operation interaction between classes C and D, if class D is the type of a parameter of operation O_C or class D is the return type of O_C. The first two categories (relationship through return type and argument passing) above are included in O-O interaction.

The Class-Class interaction (C-C) is defined as the interaction between two classes if any one of the above two interaction occurs (i.e. interaction through object declaration or inheritance). Let C and D be two classes of an object oriented system. There is an C-C interaction between the classes C and D, if an object O_d of D is declared inside class C or D is derived from class C through inheritance. The last two categories above (relationship through object declaration and inheritance) are included in C-C interaction.

We have developed tool using java for analyzing the source code of an object oriented system to get the design pattern of the system. The *Design Analyzer* implements the interactions of O-O and C-C. The input of the system is the source code of an Object Oriented program and output is an interaction graph of the design pattern of the software.

V. RESULTS AND DISCUSSIONS

For the set of x_{ij} both the Category1 and Category2 Coupling metrics are calculated as defined above for all 3 softwares under observation. Here Software-1 has 20 classes, Software-2 has 11 classes and Software-3 has 13 classes. The Principal Component Analysis produces the principal components for both Category1 coupling metrics (Shown in Table 1, Table 3 and Table 5) and Category2 coupling metrics (Shown in Table 7, Table 9 and Table 11).

For the Category1 coupling metrics, in case of Software-1 (see Table 1), the projection of the objects into the first principal component retains 76.85% of the total variance and projection of the first two principal components retains 94.99% to the total variance as in Table 2. Observation of the first Eigen vectors reveals that in case of Category1 coupling metrics for Software-1, TNUCC is the most significantly weighted measure.

For Category1 metrics, in case of Software-2 (see Table 3), the projection of the objects into the first principal component retains 85.75% of the total variance and projection of the first two principal components retains 97.76% to the total variance as shown in Table 4. Observation of the first Eigen vectors reveals that in case of Category1 coupling metrics for Software-2, “Class Coupling” is the most significantly weighted measure.

Table 5 shows the principal components of Category1 coupling metrics of Software-3, the projection of the objects into the first principal component retains 84.46% of the total variance and projection of the first two principal components retains 96.87% to the total variance as shown in Table 6. Observation of the first Eigen vectors reveals that in case of Category1 coupling metrics for Software-3, “Class Coupling” is the most significantly weighted measure.

Table 7 shows principal components o the Category2 coupling metrics of Software-1, the projection of the objects into the first principal component retains 66.56% of the total variance and projection of the first two principal components retains 99.74% to the total variance as shown in Table 8. Observation of the first Eigen vectors reveals that in case of Category2 coupling metrics for Software-1, “RNUCC” is the most significantly weighted measure.

For the Category2 coupling metrics, in case of Software-2 (see Table 9), the projection of the objects into the first principal component retains 68.18% of the total variance and projection of the first two principal components retains 98.78% to the total variance as in Table 10. Observation of the first Eigen vectors reveals that, for Software-2, “RNUCD” is the most significantly weighted measure.

For the Category2 coupling metrics, in case of Software-3 (see Table 11), the projection of the objects into the first principal component retains 67.26% of the total variance and projection of the first two principal components retains 86.10% to the total variance as shown in Table 12. Observation of the first Eigen vectors reveals that in case of coupling metrics with ratio, for Software-3, AHF is the most significantly weighted measure.

The experimental result shows that projecting the object class feature vectors onto the first two principal components retained up to a considerable range of the total variance, hence two components are sufficient to represent the entire dataset with less error. But the most significant component for all these software is not same. Hence there is no uniform coupling measure which remains the most significant component for all software.

Table 1: The Principal Components and their Eigen Values for Ratio less Coupling Metrics of Software 1.

Principal Component No.	Component Vector	Eigen Value
1	(-0.0388 , 0.0505, 0.0289, **0.7147**, 0.6960, 0.0000)	2.8132
2	(-0.3550, 0.7244, 0.1141, -0.0371, -0.0390, -0.5774)	0.6641
3	(-0.0399 , -0.0228 , 0.0026 , 0.6966 , -0.7160 , -0.0000)	0.1771
4	(-0.4562 , -0.6741 , -0.0521 , -0.0105 , 0.0365 , -0.5774)	0.0043
5	(-0.8112 , 0.0503 , 0.0620 , -0.0476 , -0.0025 , 0.5774)	0.0018
6	(-0.0690 , 0.1236 , -0.9897 , 0.0160 , 0.0118 , 0.0000)	-0.0000

Table 2: The retained variances of principal components for Ratio less Coupling Metrics of Software 1.

Number of Component	% Variance Retained
1	76.85%
2	94.99%
3	99.83%
4	99.95%

Table 3: The Principal Components and their Eigen Values for Ratio less Coupling Metrics of Software 2.

Principal Component No.	Component Vector	Eigen Value
1	(0.0795 , 0.0648 , -0.0225 , -0.5955 , **0.7344** , 0.3084)	596.6112
2	(0.5364 , 0.5395 , -0.0423, -0.4623, -0.3742, -0.2564)	83.6107
3	(0.0373 , -0.0463 , -0.2535 , -0.0989 , -0.4453, 0.8510)	11.0265
4	(0.2428 , -0.8120 , -0.2278 , -0.3757 , -0.1610, -0.2506)	3.9422
5	(0.7729 , -0.1612 , 0.3617 , 0.4093 , 0.1854 , 0.2097)	0.2327
6	(0.2196 , 0.1317 , -0.8665 , 0.3366 , 0.2492 , -0.0910)	0.3619

Table 4: The retained variances of principal components for Ratio less Coupling Metrics of Software 2.

Number of Component	% Variance Retained
1	85.75%
2	97.76%
3	99.35%
4	99.91%

Table 5: The Principal Components and their Eigen Values for Ratio less Coupling Metrics of Software 3.

Principal Component No.	Component Vector	Eigen Value
1	(-0.1548 , -0.0351 , -0.2210 , -0.5965 , **0.6043**, -0.4529)	69.2342
2	(-0.3079 , -0.1109 , -0.6325 , -0.3580 , -0.2624 , 0.5438)	10.1730
3	(-0.3139 , 0.0804 , 0.2420 , -0.3704 , -0.6980, -0.4605)	2.4804
4	(-0.6161 , 0.2338 , 0.5561 , -0.0966 , 0.2745, 0.4145)	0.0583
5	(-0.6115 , 0.0560 , -0.3701 , 0.6063 , 0.0554, -0.3394)	0.0224
6	(0.1707 , 0.9603 , -0.2152 , -0.0440 , -0.0199, 0.0037)	0.0081

Table 6: The retained variances of principal components for Ratio less Coupling Metrics of Software 3.

Number of Component	% Variance Retained
1	84.46%
2	96.87%
3	99.89%

Table 7: The Principal Components and their Eigen Values for Coupling Metrics with Ratio for Software 1.

Principal Component No.	Component Vector	Eigen Value
1	(-0.7257 , **0.6880** , 0.0075 , 0.0072 , -0.0000)	72.1786
2	(0.6878 , 0.7255 , -0.0176 , 0.0155 , -0.0000)	35.9825
3	(0.0121 , 0.0046 , 0.7057 , 0.0484 , 0.7067)	0.1926
4	(0.0066 , 0.0167 , 0.0677 , -0.9975 , 0.0005)	0.0875
5	(-0.0121 , -0.0045 , -0.7050 , -0.0477 , 0.7075)	0.0000

Table 8: The retained variances of principal components for Coupling Metrics with Ratio for Software 1.

Number of Component	% Variance Retained
1	66.56%
2	99.74%
3	99.92%

Table 9: The Principal Components and their Eigen Values for Coupling Metrics with Ratio for Software 2.

Principal Component No.	Component Vector	Eigen Value
1	(**0.7071** , -0.7047 , 0.0583 , 0.0032 , 0)	10.0188
2	(-0.8173 , 0.5739 , 0.0512 , -0.0000, 0)	4.4970
3	(-0.0032 , 0.0583 , -0.7047 , 0.7071, 0)	0.1786
4	(0 , 0 , 0 , 0 , 1.0000)	-0.0000
5	(0.0032 , -0.0583 , 0.7047 , 0.7071 , 0)	0

Table 10: The retained variances of principal components for Coupling Metrics with Ratio for Software 2.

Number of Component	% Variance Retained
1	68.18%
2	98.78%

Table 11: The Principal Components and their Eigen Values for Coupling Metrics with Ratio for Software 3.

Principal Component No.	Component Vector	Eigen Value
1	(0.2100 , -0.7591 , **0.5482** , -0.2811 , 0.0000)	0.2100
2	(-0.9049 , 0.0577 , 0.4215 , -0.0099 , 0.0000)	0.0588
3	(0.2246 , 0.4579 , 0.4135 , -0.2625 , 0.7071)	0.0301
4	(0.1898 , 0.0311 , 0.4241 , 0.8849 , -0.0000)	0.0133
5	(-0.2246 , -0.4579 , -0.4135 , 0.2625 , 0.7071)	-0.0000

Table 12: The retained variances of principal components for Coupling Metrics with Ratio for Software 3.

Number of Component	% Variance Retained
1	67.26%
2	86.10%
3	95.74%

VI. CONCLUSION

Despite of the very interesting research work and studies on coupling measures, there is still a little understanding of the motivation and empirical hypotheses behind many of the measures. It is reported that relating the measures is a difficult task in most of the cases and especially to conclude for which applications they can be used. Analysis shows that there are a lot of inconsistencies in different measures of coupling for object-oriented system. The variations of existing measures reveal that they cannot be represented in a unified framework accepted by all. Therefore, a conclusion can be drawn that the efforts for improving the understanding of object-oriented coupling and for creating a unique framework with empirical validation are useful and necessary.

REFERENCES

[1] L. Briand, J. Daly, J. and Wust, “A Unified Framework for Coupling Measurement in Object-Oriented Systems”, *Technical Report ISERN* 96-14, 1996.

[2] S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object Oriented Design”, *IEEE Transactions on Software Engineering* 20 (6), 476-493, 1994.

[3] P. Coad and E. Yourdon, “Object-Oriented Analysis”, *prentice-Hall*, second edition 1991.

[4] W. Stevens, G. Myers, and L. Constantine, “Structured Design”, *IBM Systems Journal*, 13 (2), 115-139, 1974.

[5] L. Briand, S. Morasca, and V. Basili, “Property-Based Software Engineering Measurement”, *IEEE Transactions of Software Engineering*, 22 (1), 1996, pp. 68-86.

[6] P.C. Wong, R.D. Bergeron, “Multivariate visualization using metric scaling”. Proc., Visualization 97, Phoenix, October 1997, pp. 111-118.

[7] W. AI-Ahmad, “Object-Oriented Design Patterns for Detailed Design”, Journal of Object Technology, Vol. 5 No. 2, 2006, pp. 155-169.

[8]K. M. Azharul Hasan and D. N. Batanov, “Measuring Coupling for Developing Object-Oriented Systems”, *In Porc. On ICT*, pp. 325-330, 2003.

[9] Gamma E. et al, “Design Patterns: Elements of Reusable Object-Oriented software”, Addison Wesley, 1995.

[10] Imran Baig, "Measuring Cohesion and Coupling of Object-Oriented Systems- Derivation and Mutual Study of Cohesion and Coupling", School of Engineering Blekinge Institute of Technology, Thesis No: MSE-2004:29, pp. 20-26, August 2004

[11] F. Brito e Abreu “The MOOD Metrics Set” Proc. ECOOP’95 Workshop on Metrics, 1995