CCOM – A Communicational Cohesion Metric for Object-Oriented Programming

H. B. Vincentraj
Department of Computer Science
Bishop Heber College
Trichy-17, India

S. Hari Ganesh
Department of Computer Science
H.H. The Rajah's College
Pudukottai -1, India

ABSTRACT
Object-oriented software metrics are the traditional quality assessment metrics that are aimed to ensure the goodness of the software. Almost all benefits of the OO programming have been addressed through software metrics except the factor that measures the types of cohesion incorporated in software modules. Software is measured as qualitative with the incurrence of high cohesion with low coupling. Hence, in our previous works, we have proposed certain cohesion metrics for assessing the functional and sequential level of cohesion in the software. As a continuation, in this work, a novel Communicational Cohesion Metric (CCOM) is proposed to evaluate the level of communicational cohesion of the software. The theoretical validation of the proposed CCOM is performed and compared with the traditional LCOM metric for elucidating the need for CCOM.

Keywords
Software metrics, LCOM, Cohesion, Coupling, CCOM

1. INTRODUCTION
Software metric is a quantitative measure or degree with which of quality of the software system is estimated. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development [14]. Software metrics ensure the programmers adequate confidence on the built product before its delivery.

The goal is to obtain reproducible and quantifiable measurements, which may have numerous valuable applications in scheduling and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments. Software quality metrics have been the measurement of the frequency of software defects or bugs. Measurement of the various aspects of software quality is considered to be an effective tool for the support of control activities and the initiation of process improvements during the development and the maintenance phases. These measurements apply to the functional quality, productivity, and organizational aspects of the project. Though software metrics are incorporated almost in all phases of software development life cycle, its core process is to verify the program code of the software.

The traditional way of programming the software, may either be procedure oriented and object oriented. Object oriented design is becoming more popular in software development environment. The metrics for object oriented design focus on measurements that are applied to the class and design characteristics. These measurements permit designers to access the software early in process, making changes that will reduce complexity and improve the continuing capability of the design.

The term cohesion is defined as an "intra-modular functional relatedness" in software. A highly cohesive module is often preferable as it has a direct impact on reducing the complexity of the program. There are six types of cohesion possible in a module such as coincidental, logical, temporal, procedural, communicational, sequential and functional with which the coincidental cohesion represents a poor design of the module in contrast to the functional cohesion. As the order of representation of types of cohesion gradually moves, the quality of cohesion also increases. Despite of the proposals of various cohesion metrics, identification of the types of cohesion is still required as it clearly depicts the quality of the module. Hence, in this paper, a software metric is proposed to measure the communicational cohesion of a module.

2. REVIEW OF LITERATURE
Kalantari [1] invented an approach based on fuzzy computing of cohesion and coupling. They proposed that their approach helps software engineering to calculate quality parameters with metrics and coefficient of accuracy. The intension of their study was how coupling and cohesion relations could be analyzed. They found if coupling was being high and cohesion was being low, the failure rate would be decreased and reliability would be increased.

Gehlot et al [2] introduced a new criterion that focused on the interactions between class methods and class instances and developed a cohesion measurement tool for Java programs and performed a case study on several systems. They proposed certain measures of cohesion developed to assess the reusability of Java classes. Their obtained results demonstrated that class cohesion metric, based on the proposed cohesion criteria, captured several pairs of related methods, which were not captured by the existing cohesion metrics.

Panda et al [3] proposed a graph-based cohesion metric to measure the maintainability of different program parts in an object-oriented program and predict their fault proneness. The authors computed the cohesion of the sliced component as a measure to predict its correctness and preciseness. In addition, they performed a theoretical validation with the proposed technique against the existing guidelines of cohesion measurement and compared it with some existing techniques. The proposed new cohesion metric named affected component cohesion (ACCo) was able to measure the maintainability of different program parts and predict their fault proneness.

Mal et al [4] proposed class cohesion (CC) metric and empirically validated against the open source software projects to found the effective quality factors. Their study concluded that CC continuously gave better correlation with Number Line of Code (NLOC) compared to other existing cohesion metrics. The average value of CC (CohS) of a system also predicted the natures (understandability, modifiability, and maintainability) of a system.

Qu et al [5] showed that networks formed by software methods and their calls exhibited relatively significant community structures. Based on their findings they proposed two new class cohesion metrics to measure the cohesiveness of object-oriented programs. An experiment was conducted on 10 large open-source Java programs to validate the existence of community structures and the derived metrics gave additional and useful measurement of class cohesion. As an application they showed that the new metrics were able to predict software faults more effectively than existing metrics.

Mann et al [6] presented an improved cohesion metrics through inherited elements. They suggested that the inherited elements of cohesion might increase or decrease upon the design structure of super and sub classes. They proved that their study would improve the applicability of existing cohesion metrics to measure the requirement of refactoring the classes. The results showed that there were some aspects related to inheritance such as the concepts of public, private, protected and internal elements require investigation.

Ibrahim et al [7] provided an assessment criterion for measuring the quality of a software design. In this context, inherited attributes and methods are considered in the assessment. This offered a guideline for choosing the proper Depth of Inheritance Tree (DIT) that referred to the nominated classes for refactoring. Experiments were carried out on more than 35K classes from more than 16 open-source projects using the most used cohesion metrics.

Silva et al [8] presented an initial investigation about the applicability of concern-based cohesion metric as a change proneness indicator and also checked that the metric had a correlation with efferent coupling. The authors conducted an initial empirical assessment work with two small to medium-sized systems. The results indicated a moderate to strong correlation between LCC and change proneness, and also a strong correlation between LCC and efferent coupling.

Dallal [9] conducted an empirical study by applying the LCOM metric with and without considering special methods on classes of two open source java applications and statically analyzed the results of the experiment. Their results showed that the ability of LCOM in indicating class quality slightly improved and predicted the faulty classes when excluding special methods from the LCOM computation.

Amol et al [10] introduced a framework for a comprehensive metric to address SDLC requirements

- Integration of fault detection starting from requirement and architecture.
- Making fault detection-related decisions at each phase by explicit modeling of faults.
- Developing dedicated tools for fault detection modeling; providing domain-specific application-level fault prediction mechanisms.

Okike [11] presented a pedagogic evaluation and discussion about the LCOM metric using field data from three industrial systems. Their main objectives of the study was to determine whether LCOM metric was appropriate in the measurement of class cohesion and the determination of properly and improperly designed classes in the studied systems. The result of the study showed that the LCOM metric measures class cohesiveness and was appropriate in the determination of properly and improperly designed classes in the studied system.

3. MOTIVATION

The poor design of program modules leads to the creation of complex software which in turn increases the cost of software development. Moreover, the maintenance phase of complex software is also very costly in software life cycle. The deployment of software metrics could potentially reduce the feasible defects there by increasing the ease of maintenance. The focus on developing metrics for identifying the highly cohesive code implementation saves both cost and time for maintenance and reuse of the project. As the acceptance of the module also depends upon the types of cohesion, there is a need to the invention of new metrics to classify the types of cohesion assimilated in a module in order to make a qualitative software product.

4. COMMUNICATIONAL COHESION METRIC

Communicational Cohesion is the grouping up of methods that operate on the same data within a class or module for measuring the integrity of methods. Software with high quotient of communicational cohesion ensures a good representation of class design that proves the increased integrity of methods within a module or class. Software metric that evaluates the level of communicational cohesion in software modules is a still being considered as a thrust area in research which is yet to be focused. Hence, in this paper an attempt is made to propose a communicational cohesion metric (CCOM) for assessing the percentage wise communicational cohesion that the software modules are designed with. The low level communicational cohesion suggests developers for the modification of software code by increasing the sharing of attributes within the methods of class or modules. The CCOM value of a module is the percentage fraction of sum of intersecting variables between methods by both sums of intersecting and non-intersecting variables between the methods which is denoted using formula.

$$CCOM = \frac{CM}{CM + NIVBM} \times 100\%$$

CM is the communicational measure which is derived by multiplying the sum of intersecting variables between methods by two and can be represented using the formula shown in Equation.

$$CM = 2 \times IVBM$$

IVBM represents the sum of Intersection of Variables Between Methods which is denoted using the formula for the computation of IVBM.

$$IVBM = \forall_{i=1}^{n} \sum_{j=i+1}^{n} m_i \cap m_j$$

where ‘n’ denotes the total number of methods in the module, ‘m_i’ and ‘m_j’ denotes i^{th} and j^{th} methods whereas $m_i \cap m_j$ is the intersection of attributes of m_i and m_j. Finally, NIVBM represents the sum Non-intersecting of Variables Between Methods which is depicted in Equation.

$$NIVBM = \forall_{i=1}^{n} \sum_{j=i+1}^{n} !\,(m_i \cap m_j)$$

A software module with the CCOM 100% value denotes a strong communicational cohesion and 0% value denotes weak communicational cohesion. The implementation communicational cohesion in software enhances the modularity of software program.

5. ILLUSTRATION

The illustration of CCOM metric is evaluated against the pseudo code of three java programs which is described subsequently.

5.1. Pseudo Code: 1

Class EmpPayroll

{

Declare variables bsal, da, hra, netamount as double

Method get ()

{

Assign bsal as 5000

Assign da as 500

Assign hra as 1000

Calculate netamount by adding bsal, da, hra

}

Method disp ()

{

Print bsal, da, hra, netamount

}

Method main

{

Create object for EmpPayroll

Call method get ()

Call method disp ()

}

}

The class EmpPayroll has four variables such as 'bsal', 'da', 'hra', 'netsalary' and two methods namely get () and disp (). Therefore the total number of methods (n) is 2. The first step in the calibration of CCOM is to compute the IVBM of a class. Initially the variable i is set to 1 and denotes the get () method as it in the first position in the order of method calls. Likewise, the variable j is set to i+1 which is two, denotes the disp() method as it is in the next consecutive position in the order of method calls. Hence, the intersection of variables between i th and jth methods is computed as follows:

i=1; m_{get} = {bsal, da, hra, netamount}

j=2; m_{disp} = {bsal, da, hra, netamount}

$m_i \cap m_j = m_{get} \cap m_{disp}$ = {bsal,da,hra,netamount}

$$IBVM = \forall_{i=1}^{2} \sum_{j=2}^{2} m_{get} \cap m_{disp} = 4$$

Since, there are only two methods are represented in the module the comparison is made only with those methods and the IBVM, sum of intersecting variables between methods of EmpPayroll is 4.

$$CM = 2 \times IBVM = 2 \times 4 = 8$$

The NIBVM, sum of non-intersecting variables between the methods of get() and disp() is 0. Hence, CCOM measure for EmpPayroll is computed as:

$$CCOM = \frac{CM}{CM + NIBVM} = \frac{8}{8+0} \times 100\% = \frac{8}{8} \times 100\% = 100\%$$

As the CCOM value of EmpPayroll program is 100%, the class is said to be communicational cohesive.

5.2. Pseudo Code :2

Class square

{

Declare variables a and b as double

Method first ()

{

Assign a as 10

Print square of a

}

Method second ()

{

Assign b as 20

Print square of b

}

Method main ()

{

Create object for square

Call m1 ();

Call m2 ();

}

}

Class square has two variables namely 'a' and 'b' and two methods such as first() and second(). Therefore the total number of methods 'n' is 2. As per the sequence of method calling, the method first () is called and the variables are intersected with the next consequent method second () as follows:

i=1, m_{first} = {a}

j=2, m_{second} = {b}

$m_i \cap m_j = m_{first} \cap m_{second} = \{\varnothing\}$

$$CM = 2 \times \forall_{i=1}^{2} \sum_{j=2}^{2} m_{first} \cap m_{second} = 2 \times 0 = 0$$

Since, there are only two methods are represented in the module the comparison is made only with those methods and the CM value is 0.

$$CCOM = \frac{0}{0 + \forall_{i=1}^{2} \sum_{j=2}^{2} ! (m_{first} \cap m_{second})} \times 100\% = \frac{0}{0+2} \times 100\% = \frac{0}{2} \times 100\% = 0\%$$

The CCOM value for Square class is 0% which denotes that the class is not communicational cohesive, and may be re-modified to bind the methods of the module.

5.3. Pseudo Code :3

Class Mark

{

Declare ma1, ma2, ma3, tot and avg as double

Declare name and no as string

Method getpersonal ()

{

Assign no as "ug16cs204";

Assign name as "Ramya";

}

Method getmark ()

{

Assign ma1as 78;

Assign ma2 as 64;

Assign ma3 as 72;

Calculate tot by adding ma1, ma2, ma3;

Calculate avg as tot/3;

}

Method disp ()

{

Print name, no, ma1, ma2, ma3, tot, avg

}

Method main ()

{

Create object for Mark

Call getpersonal ();

Call getmark ();

Call disp ();

}

}

The class Mark has seven variables and three methods with the order of method calls as getpersonal (), getmark () and disp (). Therefore the total number of methods 'n' is 3. Initially the variable i is set to 1 and denotes the getpersonal () method and variable j is i+1 which is set to two, denotes the getmark() method as it is in the consecutive order of method calls. Hence, the intersection of two methods is computed as follows:

i=1, j=2

i=1, $m_{getpersonal}$ = {no,name}

j=2, $m_{getmark}$ = {ma1,ma2,ma3,tot,avg}

$m_i \cap m_j = m_{getpersonal} \cap m_{getmark} = \{\emptyset\}$

$$IBVM = \forall_{i=1}^{3} \sum_{j=2}^{3} m_{getpersonal} \cap m_{getmark} = 0$$

i=1, j=3

i=1, $m_{getpersonal}$={no,name}

j=3, m_{disp}={no,name,ma1,ma2,ma3,avg,tot}

$m_i \cap m_j = m_{getpersonal} \cap m_{disp}$={no, name}

$$IBVM = 0 + \forall_{i=1}^{3} \sum_{j=3}^{3} m_{getpersonal} \cap m_{disp} = 0 + 2 = 2$$

i=1, j=4 limit exceeded

i=2, j=3

i=2, $m_{getmark}$={ma1,ma2,ma3,tot,avg,}

j=3, m_{disp}={no,name,ma1,ma2,ma3,avg,tot}

$m_i \cap m_j = m_{getmark} \cap m_{disp}$={ma1,ma2,ma3,tot,avg}

$$IBVM = 2 + \forall_{i=2}^{3} \sum_{j=3}^{3} m_{getmark} \cap m_{disp} = 2 + 5 = 7$$

i=2, j=4 limit exceeded

i=3, j=4 limit exceeded

i=4 limit exceeded

Hence, the sum of intersecting variables between methods of class Mark is 7.

$$CM = 2 \times IBVM = 2 \times 7 = 14$$

The NIBVM, sum of non-intersecting variables between the methods can be calculated as

i=1, j=2

i=1, $m_{getpersonal}$ = {no,name}

j=2, $m_{getmark}$ = {ma1,ma2,ma3,tot,avg}

! $m_i \cap m_j$=!$m_{getpersonal} \cap m_{getmark}$={no, name, ma1, ma2, ma3, tot, avg}

$$NIBVM = \forall_{i=1}^{3} \sum_{j=2}^{3} !\,(m_{getpersonal} \cap m_{getmark}) = 7$$

i=1, j=3

$m_{getpersonal}$={no,name}

m_{disp}={no,name,ma1,ma2,ma3,avg,tot}

!$m_i \cap m_j$= !$m_{getpersonal} \cap m_{disp}$={ma1, ma2,ma3,avg,tot}

$$NIBVM = 7 + \forall_{i=1}^{3} \sum_{j=3}^{3} !\,(m_{getpersonal} \cap m_{disp}) = 7 + 5 = 12$$

i=1, j=4 limit exceeded

i=2, j=3

i=2, $m_{getmark}$={ma1,ma2,ma3,tot,avg,}

j=3, m_{disp}={no,name,ma1,ma2,ma3,avg,tot}

!$m_i \cap m_j$= !$m_{getmark} \cap m_{disp}$={no, name}

$$NIBVM = 12 + \forall_{i=2}^{3} \sum_{j=3}^{3} !\,(m_{getmark} \cap m_{disp}) = 12 + 2 = 14$$

i=2, j=4 limit exceeded

i=3, j=4 limit exceeded

i=4 limit exceeded

Hence, the sum of non- intersecting variables between methods of Class Mark is 7. Hence, the CCOM value of the Mark class is derived as follows

$$CCOM = \frac{CM}{CM + NIBVM} = \frac{14}{14 + 14} \times 100\% = \frac{14}{28} \times 100\% = 50\%$$

The evaluated programs are compared with the results of standard LCOM metrics to be compared with the results of CCOM and shown in Table 1.

Table 1 . Comparison of Standard LCOM with CCOM

Program Name	LCOM	CCOM
EmpPayroll	-1	100%
Square	0	0%
Mark	-3	50%

The values -3 and -1 in LCOM represent only the existence of cohesion in methods, whereas the results of CCOM more specifically represents the amount of communicational cohesion that presents in the module with an intensive analysis on the programs. Moreover, the results of LCOM do not precisely describe the differentiation on -1 and -3, but CCOM explicates that EmpPayroll is 100%, Square is 0% and Mark is 50% communicational which would be useful for further acceptance or modification.

6. ANALYTICAL EVALUATION OF CCOM

Many researches have proposed that the acceptance of a new metric relies upon the satisfaction of certain properties that it should fulfill. For example, Basili and Reiter [12] suggest that metrics should be sensitive to externally observable differences in the development environment, and must also correspond to intuitive notions about the characteristic differences between the software artifacts being measured. Weyuker [13] has developed a formal list of properties for software metrics and has evaluated a number of existing software metrics using these properties. These properties include notions of monotonicity, interaction, non-coarseness, non-uniqueness and permutation. He developed nine properties.

Property1
Non-coarseness

$$(\exists R)(\exists S)\left(\mu(R) \neq \mu(S)\right)$$

Not all class can have the same complexity. If there are 'n' numbers of classes in the module, CCOM does not rank all 'n' classes as equally complex. .

Property 2
Granularity

Let 'z' be a non-negative number and there could be only finite number of classes have the complexity z. If the number of classes in large scale system is finite, the complexity value of CCOM is also finite. Hence this property is satisfied.

Property 3

Non-uniqueness

$$\mu(R) = \mu(S)$$

This property implies that there may be number of modules have the same complexity. CCOM abides this property, if the communicational cohesion of the modules is similar, and the complexity of the modules is also similar.

Property 4
Design details are important

$$(\exists R)(\exists S)(R \equiv S) \text{ and } \left(\mu(R) \neq \mu(S)\right)$$

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, CCOM produces different complexity values for each module.

Property 5
Monotonicity

For all modules R and S such that

$$(\mu(R) \leq \mu(R+S) \text{ and } \mu(S) \leq \mu(R+S)$$

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. CCOM abides this property if there is a possibility of combining the modules R and S and would share the attributes of the class while concatenation.

Property 6
Non-equivalence of interaction

$$(\exists R)(\exists S)(\exists T) \text{ such that } \mu(R) = \mu(S) \text{ does not imply that } \mu(R+T) = \mu(S+T))$$

This property states that if a new method is added to the two existing classes R and S which has the same class complexity, this property states that the complexities of the two new combined classes 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. CCOM for sure yields different complexity values for both classes R and S since T is dependent upon the fitness of the classes 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

$$\text{If } R \text{ is a renaming of } Q \text{ then } \mu(R) = \mu(S)$$

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. CCOM does not have any impact over the change of name of module, hence CCOM satisfies property 8.

Property 9
Interaction increases complexity

$$(\exists R)(\exists S)\left(\mu\left((R)+\mu(S)\right)<\mu(R+S)\right)$$

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 satisfied with CCOM as the complexity of the combined classes increases than the individual complexities. Summary of the CCOM validation is described in Table 2.

Table 2 – CCOM Validation against Weyuker's Metric

Metric	P1	P2	P3	P4	P5	P6	P7	P8	P9
CCOM	Y	Y	Y	Y	Y	Y	N	Y	Y

7. CONCLUSION

The lack of software cohesion metrics in OO programming has lead to the discovery of CCOM, a novel software metric that is designed to be incorporated with the testing phase of software development life cycle. The objective of CCOM metric is to measure the amount of communicational cohesion of a class. High communicational cohesion denotes high quotient of integrity of software modules which is the most preferable factor for maintainability, modifiability and understandability of software. Moreover, high communicational cohesion also reduces the complexity of the overall software. Moreover, the CCOM assists the developers to evaluate their software programs to fine-tune the coding part which necessarily cut the operational and time costs. The evaluation of CCOM metric has proven as a qualified metric as it satisfies eight out of nine properties of weyuker's metric scale. Hence, the metric may widely be deployed in software industries for building quality products.

8. REFERENCES

[1] Kalantari, Samira, Masoomeh Alizadeh, and Homayoun Motameni. 2015. Evaluation of reliability of object-oriented systems based on Cohesion and Coupling Fuzzy computing. Journal of Advances in Computer Research.

[2] Neha Gehlot and Ritu Sindhu. 2015. A Class Cohesion Measure for Evaluation of Reusability. World Engineering & Applied Sciences Journal.

[3] Panda, S., and D. P. Mohapatra. 2015. ACCo: a novel approach to measure cohesion using hierarchical slicing of Java programs. Innovations in Systems and Software Engineering.

[4] Mal, Sandip, and Kumar Rajnish. 2014. Theoretical Validation of New Class Cohesion Metric against Briand Properties. Intelligent Computing, Networking, and Informatics. Springer India.

[5] Qu, Yu, et al. 2015. Exploring community structure of software Call Graph and its applications in class cohesion measurement. Journal of Systems and Software.

[6] Mann, Ankita, Sandeep Dalal, and Dhreej Chhillar. 2013. An Effort to Improve Cohesion Metrics Using Inheritance. International Journal of computational Engineering research.

[7] Ibrahim, Safwat M. 2012. Identification of nominated classes for software refactoring using object-oriented cohesion metrics. International Journal of Computer Science.

[8] da Silva, Bruno C., Cláudio Sant'Anna, and Christina Chavez. 2011. Concern-based cohesion as change proneness indicator: an initial empirical study. Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics.

[9] Al Dallal, Jehad. 2011. Improving object-oriented lack-of-cohesion metric by excluding special methods. Proceedings of the 10th WSEAS international conference on Software engineering, parallel and distributed systems.

[10] Dange, A. S., and S. D. Joshi. 2011. Fault Prediction in Object Oriented System Using the Coupling and Cohesion of Classes. International Journal of Computer Science and Management Studies.

[11] Okike, Ezekiel. 2010. A Pedagogical Evaluation and Discussion about the Lack of Cohesion in Method (LCOM) Metric Using Field Experiment. arXiv preprint arXiv:1004.3277.

[12] Basili, Victor R., and Robert W. Reiter Jr. 1979. Evaluating automatable measures of software development. Proceedings on Workshop on Quantitative Software Models.

[13] Weyuker, E. 1988. Evaluating software complexity measures. IEEE Transactions on Software Engineering.

[14] S.Hari Ganesh And H.B.Vincentraj. 2015. A Novel Co-Functional Cohesion Complexity Metric: A Quality Based Analysis. International Journal of Applied Engineering Research.