IJCTA, 9(27), 2016, pp. 137-145 ISSN: 0974-5572
© International Science Press

A Theoretical Analysis SCOM: A Software Metric

H.B.Vincentraj* and S. Hari Ganesh**

ABSTRACT

In object oriented language the complexity of the program is defined with relationship between the classes and their methods. In object-oriented paradigm, a module is a class and the cohesion refers to the relationship among the methods of a class. Several measures exist for measuring cohesion in Object-Oriented systems. Cohesion may be categorized ranging from the weakest form to the strongest form in the following order: coincidental, logical, temporal, procedural, communicational, sequential and functional. A module exhibits one of these forms of cohesion depending on the skill of the designer. However, communicational, sequential and functional cohesions are generally accepted as the best form of cohesion in software design. Functional cohesion is the most desirable because it performs exactly one action or achieves a single goal. The existence and the types of cohesion is an important factor for assessing the quality of the software project. The weak cohesion symbolizes the poor design of a class. On the other hand the strongest cohesion represents good qualitative of a class. Though cohesion is an important criterion in the assessment of quality of a software project, there is no software metrics are currently available to evaluate the type of cohesion that presents in the software project. As a continuation of our earlier work 'Co-incidental-functional cohesion metric (CFCOM)', (Ganesh et al. 2015), which identifies whether the given module is functionally or co-incidentally cohesive, we propose a new cohesion metric for assessing a module for sequential cohesion.

Keywords: Software metric, OO metrics, Cohesion, Coupling, Types of cohesion, CFCOM

1. INTRODUCTION

Software Quality Assurance is a systematic and planned set of actions necessary to provide adequate confidence that the development process of the project establishes a good quality building in terms of software program. Software Quality in software program can be ensured using software quality metrics.

A software metric is a quantitative measure of a degree to which a software system or process possesses some property. 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. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule 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.

Object oriented design is becoming more popular in software development environment and object oriented design metrics is an essential part of software 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.

* Department of Computer Science, Bishop Heber College, Trichy, India.

** Department of Computer Science, H.H The Rajah's College, Pudukottai, India.

The term cohesion is defined as an "intra-modular functional relatedness" in software. This definition, considers the cohesion of each module in isolation: how tightly bound or related its internal elements are. Hence, cohesion as an attribute of software modules capture the degree of association of elements within a module, and the programming paradigm used determines what an element is and what a module is. As cohesion plays as an important factor for evaluating a software project, there is only less number of metrics available that too with less scope of identifying only the presence of cohesion. Hence, the present scenario motivates an exemplary research over the transparency on the type of cohesion.

2. REVIEW OF LITERATURE

Kalantari (2015) 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 (2015) 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 (2015) 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 (2014) 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 (2014) 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 (2013) 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 (2012) 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 (2011) 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 (2011) 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 (2011) 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 (2010) 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. SEQUENTIAL COHESION METRIC

Sequential cohesion refers to the communication between two methods where the output of one serves as an input to the other in a sequence of method calls. Software with sequential cohesion is accepted as it increases the possibility of integration of elements of within the methods of a module. So far, there is no such metric is proposed to calibrate the level of sequential cohesion presented in a module. Hence, as a novel attempt, we have proposed a sequential cohesion metric that evaluates the percentage of sequential cohesion involved in a module.

$$SCOM = \frac{\sum_{i=1}^{n} m_i \cap m_{i+1}}{(n-1) \times TAC} \times 100\% \qquad (1)$$

'n' denotes the total number of methods in the module, 'm_i' denotes ith method whereas $m_i \cap m_{i+1}$ is the intersection of attributes of mi and m_{i+1}, and TAC refers to the total number of attributes in a class. SCOM is the percentage of the summation of intersected variables of two consequent methods divided by the possible sequential cohesion. A software possessing 100% of SCOM denotes a strong sequential cohesion and 0% denotes weak sequential cohesion. The implementation of sequential cohesion in software enhances the modularity of software program.

5. ILLUSTRATION

The illustration of SCOM 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 get ()
Call 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 attributes (TAC) is 4 and the total number of methods (n) is 2. The sequence of the method call is organized by calling the get () method followed by the disp () method. As per the sequence of method calling, the method get () is called and the variables are intersected with the next consequent method disp () as follows:

TAC = {bsal, da, hra, netamount} = 4

m1 = get () = {bsal, da, hra, netamount}

m2 = disp () = {bsal, da, hra, netamount}

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

5.2. Pseudo Code :2

```
Class square
{
Declare variables a and b as double
Method m1 ()
{
Assign a as 10
Print square of a
}
Method m2 ()
{
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 m1 and m2. The sequence of call is organized by calling the method m1 () followed by m2 (). Therefore, Total number of attributes (TAC) in Square is 2, and total number of methods (n) in Square is 2.

TAC = {a,b} = 2

m1 () = {a}

m2 () = {b}

The SCOM value for Square class is 0% which denotes that the class is not sequentially cohesive, and may be re-modified to incur the relationship within its methods.

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 ();
}
}
```

Class Mark has seven variables and three methods. The order of method call is getpersonal () followed by getmark () followed by disp (). Total number of variables (TAC) =7. Total number of methods (n) =3.

TAC = {no, name, ma1, ma2, ma3, tot, avg} = 7
m1 = getpersonal () = {no, name}
m2 = getmark () = {ma1, ma2, ma3, tot, avg}
m3 = disp () = {no, name, ma1, ma2, ma3, tot, avg}

The sequential cohesion of Mark program is 35.7% where there is no common elements shared by getperosnal () and getmark () method unlike getmark () and disp () methods. Hence, the program needs to be reformed by combining the methods of getpersonal () and getmark (). The evaluated programs are compared with the results of standard LCOM metrics to verify the enhancements with the results.

Table 1
Comparison of Standard LCOM with SCOM

Program Name	*LCOM*	*SCOM*
EmpPayroll	-1	100%
Square	0	0%
Mark	-3	35.7%

The values -3 and -1 in LCOM represent only the existence of cohesion in methods, whereas the results SCOM more specifically represents the amount of sequential cohesion involved in the program with an in-depth analysis of the program. Moreover, the results of LCOM do not precisely describe the differentiation on -1 and -3, but SCOM explicates that EmpPayroll is 100%, Square is 0% and Mark is 35.7% sequential which would be useful for further acceptance or modification.

6. ANALYTICAL EVALUATION OF SCOM

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 [9] 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 [10] 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, SCOM 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 SCOM is also finite. Hence this property is satisfied.

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

Property 3

Non-uniqueness

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

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

Property 4

Design details are important

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, SCOM produces different complexity values for each module.

Property 5

Monotonicity

For all modules R and S such that

$$(\mu(R) \leq \mu(R+S)\ and\ \mu(S) \leq \mu(R+S))$$

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.SCOM 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

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. SCOM 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

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

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

Property 9

Interaction increases complexity

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 SCOM as the complexity of the combined classes increases than the individual complexities. Summary of the CFCOM validation is described in Table 2.

Table 2
CFCOM Validation against Weyuker's Metric

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

7. CONCLUSION

SCOM is a software metric which is to be incorporated in the testing phase of software life cycle in addition with the CFCOM metric. The software with high CFCOM and SCOM value indicates that the modules of the software product have high quotient of integrity within themselves which certifies that the development of software is qualitative. Moreover, the metrics 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 SCOM 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.

REFERENCES

[1] S.Hari Ganesh And H.B.Vincentraj. "A Novel Co-Functional Cohesion Complexity Metric: A Quality Based Analysis," International Journal of Applied Engineering Research, Issn 0973-4562 Vol. 10 No.85, 2015.

[2] Kalantari, Samira, Masoomeh Alizadeh, and Homayoun Motameni. "Evaluation of reliability of object-oriented systems based on Cohesion and Coupling Fuzzy computing." Journal of Advances in Computer Research 6, no. 1 (2015): 85-99.

[3] Neha Gehlot and Ritu Sindhu. "A Class Cohesion Measure for Evaluation of Resuablity", World Engineering & Applied Sciences Journal, ISSN 2079-2204, pp. 104-108. 2015.

[4] Panda, S., and D. P. Mohapatra. "ACCo: a novel approach to measure cohesion using hierarchical slicing of Java programs". Innovations in Systems and Software Engineering 11.4. pp: 243-260, 2015.

[5] Mal, Sandip, and Kumar Rajnish. "Theoretical Validation of New Class Cohesion Metric against Briand Properties. Intelligent Computing, Networking, and Informatics". Springer India, pp. 591-597, 2014.

[6] Qu, Yu, et al. "Exploring community structure of software Call Graph and its applications in class cohesion measurement", Journal of Systems and Software, pp: 193-210, 2015.

[7] Mann, Ankita, Sandeep Dalal, and Dhreej Chhillar. "An Effort to Improve Cohesion Metrics Using Inheritance", International Journal of computational Engineering research (IJCER) IJCER, VOL 3 ISSUE 6, 2013.

[8] Ibrahim, Safwat M. "Identification of nominated classes for software refactoring using object-oriented cohesion metrics", International Journal of Computer Science Issues 9.2, pp: 68-76, 2012.

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

[10] Al Dallal, Jehad. "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. World Scientific and Engineering Academy and Society (WSEAS), 2011.

[11] Dange, A. S., and S. D. Joshi. "Fault Prediction in Object Oriented System Using the Coupling and Cohesion of Classes", International Journal of Computer Science and Management Studies 11.02, pp: 48-51, 2011.

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