Design of Metrics for Component-Based Software System at Design Level

Kratika Yadav, Pradeep Tomar

***Abstract*— Component-Based Software Systems (CBSS) have now become more generalized approach for application as it mainly focus on assembling individual components, to develop the application. Today's applications are large, complex and are not integrated. Although they come packaged with wide range of features but most features can neither be removed, upgraded independently or replaced nor can be used in other applications. One of the most critical activities in this reuse based process is the selection of appropriate components. This paper proposes a set of metrics i.e. coupling and cohesion metrics which will help in the evaluation of component in CBSS at design level. These metrics of a component may provide an indirect measurement of its external characteristics. These propose metrics is used to decide upon a criterion against which candidate components can be evaluated in CBSS at design level. These new metrics are helpful to designers and testers in performing assessment and improvement of CBSS design quality.**

***Index Terms*— Cohesion, Coupling, CBM (Cohesion between Methods), LOC**

I. INTRODUCTION

Software engineering is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software [1]. *Stephen Schac*h defined the same as "A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements". Measurement is the process by which numbers or symbols are assigned to attribute of entities in the real world in such a way so as to describe them according to clearly defined rules. There are different types of measurements like Direct Measures (internal attributes): To measure the cost, effort, LOC, speed and memory. Indirect Measures (external attributes): To measure the functionality, quality, complexity, efficiency, reliability and maintainability.

Measurements are a key element for controlling software engineering process. Measurement are very important in software industry because

- Software metrics can help to fully understand both the design and architecture information of the software system.
- Software design metrics can aid to discover the underlying errors in the software design at the early stage of software development life cycle.

The IEEE Standard Glossary of Software Engineering Terms [1] defines a metric as 'a quantitative measure of the degree to which a system component or process possesses a given attribute'. Software metrics play a very important role in assessing and predicting various attributes of software such as complexity, reusability, maintainability, testability etc. Among these attributes complexity affects all other attributes of the software [2].

Software metrics are used to measure the software quality to check whether it satisfies the requirements. Metrics are defined as "Quantifiable measures that could be used to measure characteristics of a software system or the software development process." Software metrics are essential to plan, predict, monitor, control, evaluate, products and processes. The main goal of the software metrics is to reduce costs, Improve quality, Control/ Monitor schedule, small testing effort, many reusable fragments, to better understand the quality of the product and the program [3].

II. COMPONENT-BASED SYSTEMS

Modern approach to software re-use has been through Component-Based Software Engineering (CBSE). Component-Based Software Development (CBSD) approach is based on the idea to develop software systems by selecting appropriate of-the-shelf components and then to assemble them with a well-defined software architecture. Exactly what constitutes a software component has been a subject of much debate with the field of CBSE.

Even though there is no IEEE/ISO standard definition for a component that they know of, one of the leading exponents in this area, *Syzperski* [4] defines software component as follows: "Software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party". *Syzperski* indirectly states that components have to be composed to work together in order to build a system. The most important feature of a component is the separation of its interface from its implementation.

CBSD has become widely accepted as a cost-effective approach to software development, as it emphasizes the design and construction of software systems using reusable components. CBSD can significantly reduce development cost and time-to-market, and improve maintainability, reliability and overall quality of software systems. The CBSE process consists of two separate but related processes. The

Manuscript received April 20, 2014.
Kratika Yadav, School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh,
Pradeep Tomar, Assistant Professor, School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh.

first is concerned with the analysis of application domains and the development of domain-related components (i.e. development for reuse). The *second* process is concerned with assembling software systems from prefabricated (off-the-shelf) components (development with reuse). The two processes i.e. Development for reuse and Development with reuse are linked via a component market [5].

III. RELATED WORK

Many different metrics have been proposed for Object-Oriented Systems(OOS), there are three major sets of design metrics reported in the research literature. They are mainly for principle structures that, if improperly designed, negatively affect the design and code quality attributes. They are due Chidamber and Kemerer (CK) suite [6], Lorenz and Kidd (LK suite) [7,8] and Britto e Abreu (MOOD Suite) [9].

In CBS research mainly conducted on the two major areas .Many research papers, such as [10-12], focused on measuring the reusability of software components whereas others such as [13,14,16,17], centered on measuring interaction complexity of integrated components.

Kuljit Kaur Chahal et. al. [15] provide a good description on a metric based approach to evaluate design of software components. They are focusing on quality of internal design of a software component and its relationship to the external quality attributes of the component. They have studied the basic elements of the component based software development approach [23]. In this paper, several points of difference of the traditional software development from the modern component based software development are identified. Software development processes with new sets of activities for this paradigm are discussed. They applied *CK-Metric suite* and *Abreu's* Metric suite to a model software component. It was found that the internal design of the software component lacks quality [13]. Designers of the component have not made use of the features of the object oriented methodology.

V. L. Narasimhan and B. Hendradjaya [16] has proposed two sets of metrics to measure complexity and criticality of large software systems designed and integrated using the principles of CBSE. From the Component Interface Definition Language (CIDL) specification, they derive two suites of complexity metrics, namely, Component Packing Density (CPD) metrics and Component Interaction Density (CID). The CPD metric relates component constituents to the number of integrated components. The CID metric suite relates interactions between components to the number of available interactions in the entire system. They also define a set of criticality criteria for component integration. They proposed experimental design and the expected results are also outlined in this paper. The metrics proposed in this paper can be used to identify the complexity and criticality of the metrics. By recognizing a complex and/or critical component, it should give a contribution on the effort and cost estimation. This information should help a software project leader to estimate better.

M. Abdellatief et. al. [17] has proposed, a set of metrics based on the Component Information Flow (CIF) was developed to characterize and evaluate the effect of the component design size on the quality of Component-Based Software System (CBSS) design. A CIF based on inter-component flow and intra-component flow. CIF measurement and multidimensional approaches for measurement interpretation evaluate the area of component. The theoretical evaluation results indicated that the proposed metrics are valid size measures. An application that demonstrates the intuitiveness of the mentioned approach is also presented. Results show that multidimensional analysis of design size appears promising as a means of capturing the quality of the CBSS design in question.

Eun Sook Cho et. al. [18] has proposed a metrics for measuring the complexity, customizability, and reusability of software components. They have measured the complexity, customizability, and reusability of components produced during component development process for banking domain. Several different metrics have been for this purpose Component Complexity Metric (CCM), Component Plain Complexity (CPC), Component Static Complexity (CSC), Component dynamic complexity (CDC), and Component Cyclomatic Complexity (CCC) and Component Reuse Level (CRL). They applied CRL to measure the reuse level of developed components into component-based banking systems. They have found that the complexity of a component may help to estimate the component's size. Also, reusability and customizability of components effect on the reusability of components during component based software development.

Hironori Washizaki et. al. [19] has proposed a new metric that measures the coupling-based complexity of CBS by abstracting the target system's structure through a step-wise process and taking into consideration the characteristics of remote components. There metric can be applied to CBS based on the Enterprise JavaBeans component architecture. As a result of experimental evaluations, it is found that there metric better reflects the maintainability than conventional metrics. It is also found that there metric is non-redundant with existing metrics such as Coupling Factor.

Gui Gui et. al. [20] has proposed a set of new static metrics of coupling and cohesion developed to assess the reusability of Java components retrieved from the Internet by a software component search engine. These metrics differ from the majority of established metrics in three respects: They measure the degree to which entities are coupled or resemble each other, they quantitatively take account of indirect coupling and cohesion relationship and they also reflect the functional complexity of classes and methods. An empirical comparison of the new metrics with eight established metrics is described. Results show the new metrics are consistently superior at measuring and ranking the reusability of software components. The methodology used in metrics, to determine the strength of indirect relationships when a pair of vertices was linked by multiple paths was crude though effective: They simply chose the strength indicated by the strongest path. The consequence of this is that indirect relationships may be underestimated. It would be possible to remedy this by aggregating the weights contributed by all possible paths between two vertices. The metrics are: WTcoh for measuring cohesion and WTcoup for measuring coupling. Both the

metrics consider transitive (indirect) relationships between entities.

Chuan Ho Loh et. al. [21] has attempted to quantify the amount of cohesion in classes and components via a suite of object-oriented design metrics. They proposed four object-oriented design metrics to evaluate cohesion at the class and component level. The metrics are augmented based on different definitions of LCOM. The metrics are normalized to produce values in the range, thus yielding more meaningful values than other cohesion metrics such as LCOM1 and LCOM4. The proposed metrics attempt to evaluate whether an artifact (i.e. class or component) represents one abstraction (good) or multiple abstractions (bad). If the artifact represents multiple abstractions, it should be split up into multiple artifacts (i.e. classes and components).

IV. PROBLEM DESCRIPTION

Software engineering aims at development of high-quality software and tools to promote quality software that is stable and easy to maintain and use [22]. In order to assess and improve software quality during the development process, developers and managers use, among other means, ways to automatically measure the CBS at Design level. In CBS component design has two perspectives external or interface design that is visible to the component user (component assembler), and internal design that is initially visible to the component developer only and later to the component maintainer too[15]. It is a known fact that effort of software maintenance depends largely upon the internal design of the software [23]. If internal design of a component is not good, more cost (in terms of effort and time) will be involved in updating the component to meet changed requirements. A well designed system can be easily maintained. Good design leads to the high component reusability, low dependency among components and ease of the maintenance software. A well designed component, in which the functionality has been appropriately distributed to its various sub-components, is more likely to fault free and various will be easier to adapt. Poor quality comes from poor design, where internal structures and methods are exposed, resulting in complicated inter-dependencies that grow worse over time [15]. The bad design choices may be made because of time to market pressure. In a research literature survey, a lot of work has been reported that maps the internal measures of the object oriented designs to the external attributes of the software products [24, 25].

V. PROPOSED WORK

Components definition adopted in this paper clearly supports *Szyperski's* [4] definition. In CBS components, Interface and classes are the fundamental units. The member of a class is the attributes, constructors and methods. Similarly, the member of a component is the classes and the interface. In this paper we mainly focus on the internal attribute of a component in CBS at design level. Internal attribute of software includes size of the software component, modularity (responsibility distribution among classes), information hiding, abstraction used, level of cohesion, coupling and complexity etc. According to our research literature survey, no prior work exists on the evaluation method of component in CBS at design level.

This section describes the cohesion and coupling metrics computable with the information available at design level in CBS. At the design level, the design components that are identifiable are name of the class, classes, its attributes, object, method signature includes name of the method and its parameter list which describes name of the parameter and their types. A class does not have a detailed or algorithmic description of its methods available at this level.

A. *Cohesion Metrics*

Cohesion, originating from the structured design, refers to the relatedness of the elements in a module. A highly cohesive module is one whose elements have a close relationship among them in order to provide the sole functionality of the module . Cohesion metrics measure the extent to which the methods of a class are related to each other and evaluate the quality of a CBS at design level.

Cohesion In Class (CIC)
Cohesion in class refers to the frequency of attributes usage by the methods of class. A class is cohesive if the association of elements declared in the class is focused on accomplishing a single task.

$$FA = \sum_{i=0}^{n} f(Ai) / TM$$

$$CIC = FA/TM \qquad (1)$$

n = no. of attribute in the class
f(Ai) = Frequency of each attribute that are used by methods in the class
TM= total no. of method in the class

Cohesion Between Method (CBM)
Cohesion between method refers to the relatedness of class members i.e. its attributes and methods. This metric considers the method-method interaction. This metric can easily account for direct and transitive interaction.

If a= 0 and m> 1 then CBM =*0*
if $a > 0$ and $m = 0$ or $m = 1$ then CBM =1
otherwise,

$$CBM = \frac{\sum_{i=0}^{a} Mi(Ai)}{am(m-1)} \qquad (2)$$

Mi(Ai) = sum of the method that are used same type of attribute
m= no. of method in the class.
a= no. of attribute

Cohesion Between Component (CCom)
Cohesion between component refers to the relatedness between the component.

$$CCom = \begin{cases} \text{If } CC = TIC \text{ then } CCM = 0 \\ \text{If } CC = 0 \text{ then } CCM = 1 \\ \text{Otherwise, } \sum_{i=1}^{n} CCi/TIC \end{cases} \quad (3)$$

TIC = Total no. of interface count between the component
CC = Caller component
n = no. of Component

B. Coupling Metrics

The term "coupling" was first used in software engineering by Stevens et al in the days when structured programming was the norm. It was defined as "the measure of the strength of association established by a connection from one module to another" [26].The coupling of a class means the measurement of the interdependence of class with the other classes. Coupling and cohesion relate to particular relationships that exist between classes, and within a class, respectively. Relationships that contribute to coupling were defined by Eder et al [27]. Three types of coupling were defined: interaction coupling, component coupling and inheritance coupling.

Coupling between Class (CuC)
Coupling between class refers to the dependency on other classes.This study attempt to measure a class coupling on the basis of UML relationships.

CuC = No. of classes count that are coupled × Weight Value of each relationship between class.

$$m(m-1) \quad (4)$$

m = no. of classes paired

According to accessibility between classes, the size of weight value for the relationships is defined.
The weight value for the relationships as following priority[17].

Dependency<Association<Generalization<Aggregation<Composition

The weight value of each relationship will find through the metrics which is given by *Imrain Baig*

Coupling Between Method (CuM)

$$CuM = \sum I(mi)/M \quad (5)$$

I(mi)=count of each imported method in a class
M= total no. of methos in the class

Coupling Between Components (CuCom)
Coupling between Components refers to the dependency of a component and a impact-dependency of acomponents.

$$CC = CD + IC \quad (6)$$

CD=Component Dependency
IC= Interface Coupling
Interface Coupling (IC) = No. of inflows of a component

VI. CONCLUSION

The aim of software engineering is to develop high quality software and quality is a customer satisfaction. In software system with reusable component, the customer of a software component is interested in external product attributes like functionality, reliability, reusability, maintainability, portability, efficiency and understandability etc. It was found that internal design of a software component affect the quality of a software component. Each design constructs affects certain quality attributes of a component. In this paper we have proposed a new metrics for cohesion and coupling for a CBS. Metrics can be used to check as to up to which level a particular object oriented software component follows the principals of a good object oriented design. Good design leads to the ease of maintenance of the software component. Cohesion is considered as one of most important characteristics in design of a component. Cohesion can be used to identify the poorly designed classes and components in CBS. This new metric will helpful to designers, researchers and practioners of both parties in performing assessment and improvement of CBSS design quality.

REFERENCES

[1] ANSI/IEEE (1990), IEEE Standard Glossary of software Engineering Terminology, IEEE computer Society.
[2] Gill N.S and Grover P.S.,(2003) "Component–Based Measurement: Few Useful Guidelines"; *ACM SIGSOFT Software Engineering, Volume 28 No.6, pp.30.*
[3] Fenton N.E, (1991), "Software Metrics: A Rigorous Approach", Chapman and Hall.
[4] Szyperski C. (1998), Component Software–Beyond Object-Oriented Programming, Addison-Wesley
[5] Butler, G., Li, L., and Tjandra, I.A., (2008) **"**Reusable Object-Oriented Design", available athttp://citeseer.ist.psu.edu/butler95reusable.html
[6] S. R. Chidamber and C. F. Kemerer,"A Metrics Suite for Object Oriented-Design,"IEEE Transactions on Software Engineering, vol.20,no. 6, pp.476-493, 1994.
[7] R. Pressman. (2000) "Software Engineering: a Practitioner's Approach: European Adaptation." *5th edition, McGraw-Hill, UK.*
[8] M. Lorenz and J. Kidd, (1994) "Object-Oriented Software Metrics: A Practica Guide", Prentice Hall, Englewood Cliffs, New Jersey.
[9] F. Abreu and R. Carapuça,"Object-Oriented Software Engineering:Measuring and Controlling the Development Process", Proceedings of the 4th International Conference on Software Quality, McLean, VA,USA,1994.
[10] Washizaki, H., Yamamoto, H., Fukazawa, Y.: 'A metrics suite for measuring reusability of software components'. Proceeding of Ninth International Symposium on Software Metrics, 2003, p. 211
[11] Rotaru, O.P., Dobre, M.: 'Reusability metrics for software components'. Proc. ACS/IEEE 2005 Int. Conf. on Computer Systems and Applications, Cairo, Egypt, AICCSA-05, 2005, pp. 24–29.
[12] Gill and Balkishan's, (2008), "Dependency and Interaction Oriented Complexity Metrics of Component-Based Systems", *ACMSIGSOFT Software Engineering, Vol. 33, pp. 1-5.*
[13] Salman, N.: 'Complexity metrics AS predictors of maintainability and integrability of software components', J. Arts Sci., 2006, 5, pp. 39–50
[14] Kuljit Kaur Chahal and Hardeep Singh, (2008) "A Metrics Based Approach to Evaluate Design of Software Components", *Proceeding of IEEE International Conference on Global Software Engineering, pp. 269-272.*
[15] V. L. Narasimhan and B. Hendradjaya , (2007) "A New Suite of Metrics for the Integration of Software Components'', *Informing Science and Information Technology.*

[16] M. Abdellatief et. al.,(2012) "Multidimensional Size Measure for Design of Component-Based Software System'', *The Institute of Software and Technology, Vol. 6, pp. 350–357.*

[17] Eun Sook Cho, Dongduk Women's University, (2001) "Component Metrics to Measure Component Quality''**,** *IEEE Transaction on Software Engineering, pp. 419-426*

[18] Hironori Washizaki Tomoki Nakagawa, Yuhki Saito and Yoshiaki Fukazawa, (2006), "A Coupling-based Complexity Metric for Remote Component-based Software Systems Toward Maintainability Estimation", *IEEE Asia Pacific Software Engineering Conference, pp. 73-86.*

[19] Gui Gui and Paul. D. Scott, (2009) "Measuring Software Component Reusability by Coupling and Cohesion Metrics", *Journal of Computers, Vol. 4, pp. 737-805.*

[20] Chuan Ho Loh and Sai Peck Lee, (2009) "Towards Cohesion-based Metrics as Early Quality Indicators of Faulty Classes and Components", *proceeding of International Symposium on Computing, Communication, and Control, Vol. 1, pp. 314-319.*

[21] *V. Krishnapriya, Dr. K. Ramar, (2010)* "Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures", *proceeding of IEEE International Conference on Advances in Computer Engineering.*

[22] Briand, L.C., Morasca, S., Basili, V.R.: 'Measuring and assessing maintainability at the end of high level design'. *Proc. Conf. on, Software Maintenance, 1993, CSM-93, 1993, pp. 88–87*

[23] 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*.

[24] Chidamber, S.R., Darcy, D.P., and Kemerer, C.F., "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis*", IEEE Transactions on Software Engineering, vol. 24(8), 1998*.

[25] Heung Seok Chae, Yong Rae Kwon and Doo Hwan Bae, (2000), "A cohesion measure for object-oriented classes*", Software Practices and Experinces.*

[26] Sukainah Husein,and Alan Oxley, (2009) "A Coupling and Cohesion Metrics Suite for Object-Oriented Software" *proceeding of International Conference on Computer Technology and Development, pp. 421-425.*

[27] Abdellatief, M., Sultan, A.b.M., Abdul Ghani, A.A., Jabar, M. (2011) "Component-based system dependency metrics based on componentinformation flow measurement", *Proceeding of Sixth International Conference on Software Engineering Advances, Barcelona, Spain, pp. 76–83*