Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Coupling Metric to Measure the Complexity of Component Based Software through Interfaces

Sachin Kumar, Pradeep Tomar, Reetika Nagar, Suchita Yadav
School of Information and Communication Technology
Gautam Buddha University, India

Abstract— Component-Based Software Development (CBSD) is becoming the trend for software development by using the existing component. The existing components may be black box in nature. There are many software complexity metrics proposed by various researchers which are based on the source code of the software. But these traditional metrics are not applicable for black box component because source code is not available due to the black box nature of component. Black box nature of components creates the difficulty to measure the complexity of software. In this paper metric has been proposed to determine the coupling complexity of software which is developed by using black box component. The proposed metric is based on the interfaces between the components in the component-based system.

Keywords— Component Based Software Engineering (CBSE), Component Based Software Development (CBSD), Black Box Components, Coupling Complexity, Component Interfaces.

I. INTRODUCTION

Component Based Software Engineering (CBSE) is the process of assembling, adapting and wiring software component into a complete application. It improves reusability and provides high level abstraction. Using reusable components failure of object oriented develop to support effective reuse. In the past recent years, CBSE has generated tremendous interest among software industry and many other numerous industries. Thus, development of component based software came into existence effectively.

Component Based Software Development (CBSD) [1] is a modern approach for developing complex and large software systems. Basically this approach uses prefabricated components to develop the software by integrating these independent components. CBSD is efficient as it mainly depends on the reuse of already existing and tested software components. A software development technique from existing components has been shown in the following figure:

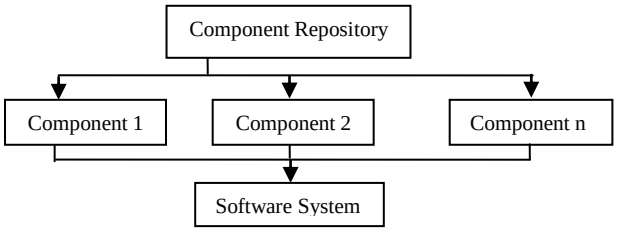


Fig. 1 CBSD Techniques

This approach helps to improve the overall quality of software system developed. It minimizes the development time, effort and cost required in any software system development. It ultimately increases maintainability and productivity of the resulting software. In component based software development, selection and evaluation suitable reusable components before integrating them to form quality software in minimum time is essential. Component can be evaluated on the basis of its various factors such as complexity, cohesion and coupling. Complexity is generally associated with operators, variables, statements and the interaction among the associated system. More interaction in system means more complex system. Cohesion refers to the degree to which the elements of a module relate together [2,3]. Thus, it is a measure of connectivity among components elements and depends on the source code of a software module . Coupling is the degree of dependency with which each program module interacts and relies on other modules [2,3]. It refers to inter-dependencies among software components and is independent of source code. Component complexity plays most important role because it affects cost, time, effort, testability, etc. of software development. Many software metrics exits for product and process but these are not sufficient for measuring the complexity of components and Component-Based Systems (CBS) Measuring complexity of components having source code is quite easy. But nowadays black box components are available for reuse. Black box components means source code of component is not available which causes problem in determining complexity of components and developed component based system. Thus, a metrics need

to be developed for components and component based systems for measuring component complexity to improve the overall quality and reduce the development time, effort and cost required in any software system development. In this paper, software metric Coupling Complexity of Black Box Component (CCBC) has been proposed for measuring complexity of coupling component, where components are black box in nature. CCBC metrics depends on component interfaces.

The paper is organized as follows. Section II discusses about already existing software metrics. Section III gives a brief idea about black box components. Section IV describes the proposed metric. Section V discusses results and section VI discusses conclusion.

II. EXISTING SOFTWARE METRICS

A. Object-Oriented Software Metrics

There are some object-oriented complexity metrics which are used for measure the complexity of object-oriented software. But few metrics are also used for component-based system. Proposed by Chindambar and Kemeres [3].

1) *Weighted Method per Class (WMC):* This metric measure the complexity of software by sum of complexity of local methods of class. High value of this metrics means system is complex.
2) *Depth of Inheritance (DIT):* This metric measure the complexity of software by sum of complexity of local methods of class. High value of this metrics means system is complex.
3) *Response for Class (RFC):* RFC is used to count the invoked method that can execute in response to a message sent to an object in the class. High value of this metrics means complexity of class is high.
4) *Coupling between Object (CBO):* This metrics deals between classes. CBO measure the number of other classes to which the classes is coupled. Higher value of this metric means poor design, decrease in reuse, low understand ability and high maintenance effort.
5) *Lack of Cohesion (LCOM):* It deals within the class. LCOM measure how the local Methods are related to the local instance variable in the class. High cohesion means good design.
6) *Number of Children (NOC):* NOC is used to immediate successors of the class. It deals with number of node in the inheritance tree. High value means effective reuse, poor design and more testing effort.

B. Component-Based Software Metrics

These are few existing metrics which are used to measure the complexity of software which is developed by using the black box component. These existing metrics are proposed by Narasimhan & Hendradjaya [4, 5].

1) *Component Packing Density (CPD):* The CPD metric measures the component constituents to the number of integrated components. This metric is used to identify the density of integrated components. Thus, a higher density represents a higher complexity.

$$CPD(constituent\ type) = \frac{\#Constituent}{\#Components}$$

Where, #Constituent is the number of lines of code, operations, classes, and modules in the related components.

2) *Component Interaction Density (CID):* The CID metric measures the ratio of actual number of interactions to the available number of interactions in components.

$$CID = \frac{\#I}{\#Imax}$$

Where, #I and #Imax represent the number of actual interaction and maximum available interaction respectively

3) Component Incoming Interaction Density (CIID): The CIID metric measures the ratio of actual number of in-coming interactions to the maximum available incoming interactions in a component.

$$CIID = \frac{\#Iin}{\#Imax\_in}$$

Where, #Iin and #Imax_in represent the actual number of incoming interactions and maximum number of incoming interactions available in a component respectively. High density shows that a particular component requires so many interfaces.

4) Component Outgoing Interaction Density (COID): The COID metric measures the ratio of actual number of outgoing interactions to the maximum number of outgoing interactions available in a component.

$$COID = \frac{\#Iout}{\#Imax\_out}$$

Where, # Iout and # Imax_out represents the actual number of outgoing interactions used and maximum number of out-going interactions available in a component respectively

5) *Component Average Interaction Density (CAID):* The CAID metric is a sum of interaction densities for each component divided by the number of components in software system.

$$CAID = \sum_{i=1}^{i=n} \frac{\text{CIDn}}{\#\text{components}}$$

Where, Σ CIDn represents the sum of interaction densities for components 1...n and # components represents the number of existing components in the software system.

6) *Link Criticality Metric (CRITlink):* Link Criticality metric is defined as the number of compo-nents which have links more than a threshold value.

$$CRITlink = \#linkcomponents$$

Where, # linkcomponents represents the number of components, with their links more than a critical value.

7) *Bridge Criticality Metric (CRITbridge):* Bridge Criticality metric is defined as the number of bridge components in a component assembly

$$CRITbridge = \#bridgecomponents$$

Where, # bridge component represents the number of bridge components. A bridge component may be defined as a com - ponent which links two or more components/ application. If there is a defect in bridge, the whole application might mal-function. More number of bridge components means more chances of failure.

8) *Inheritance Criticality Metric (CRITinheritance):* Inheritance Criticality metric is defined as the number of components, which become root or base for other inherited components. of Inheritance (DIT): This metric measure the complexity of software by sum of complexity of local methods of class. High value of this metrics means system is complex.

$$CRITinheritance = \#rootcomponent$$

Where, # rootcomponent represents the number of root components which has inheritance. It is the number of components which act as a parent/root/base for other components.

9) *Size Criticality Metric (CRITsize):* Size Criticality metric is defined as below :

$$CRITsize = \#sizecomponents$$

Where, #sizecomponents represents the number of components which exceed a given critical size value. The size is defined in terms of LOC, number of classes, operations and modules in the application

10) Criticality Metric: The #Criticality Metric (CRITall) is defined as the sum of all critical metrics.

$$CRITall = CRITlink + CRITinheritance + CRITsize$$

III. PROPOSED METRICS

Many coupling metric exist which is based on source code. But in case of black box components source code is not available. Thus, components functionality is determined by their specifications only. The proposed metric is to determine the coupling component complexity based on components specification and interfaces as components for which complexity is to be determined are black box in nature. This metrics uses interfaces and connections of components to measure their complexity. Our proposed coupling complexity of black box component (CCBC) metric is as given:

$$CCBC = IIc + OIc$$

Where,

IIc are the incoming or required interfaces.

OIc are the outgoing or provided interfaces.

CCBC metric is to be applied on individual component separately.

Then to measure the average coupling complexity of system above proposed metric is to be used.

$$\text{Average Coupling Complexity (ACC)} = \sum_{i=1}^{i=n} \frac{\text{CCBCi}}{n}$$

Where,

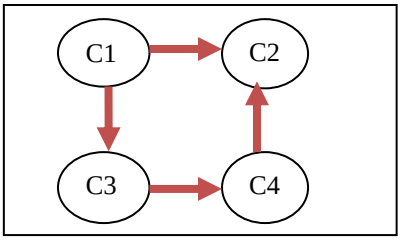
$CCBC_i$ is the complexity of i^{th} component.

n is the total number of components.

IV. RESULTS

Let us consider four cases, that is, four black box components systems having different complexity due to interfaces differences.

Case 1: First system having four components with four interfaces.



Now, coupling complexity of black box component (CCBC) is given as- CCBC = IIc + OIc

CCBC is to be calculated separately for individual. So, complexity of different components is as follows:

CCBC (C1) = 0 + 2 = 2
CCBC (C2) = 2 + 0 = 2
CCBC (C3) = 1 + 1 = 2
CCBC (C4) = 1 + 1 = 2

ACC = $\sum_{i=1}^{n} CCBC_i / n.$
ACC= (2+2+2+2)/4 = 2

Case 2: Second system having four components with five interfaces.

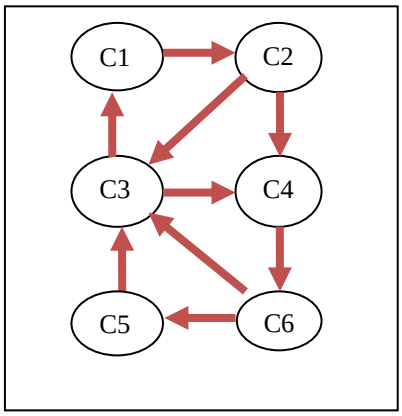
Now, coupling complexity of black box component (CCBC) is given as- CCBC = II + OI

CCBC is to be calculated separately for individual. So, complexity of different components is as follows:

CCBC (C1) = 1 + 2 = 3
CCBC (C2) = 2 + 0 = 2
CCBC (C3) = 1 + 1 = 2
CCBC (C4) = 1 + 2 = 3

ACC = $\sum_{i=1}^{n} CCBC_i / n.$
ACC= (3+2+2+3)/4 = 5/2 = 2.5

Case 3: Third system having six components with nine interfaces.



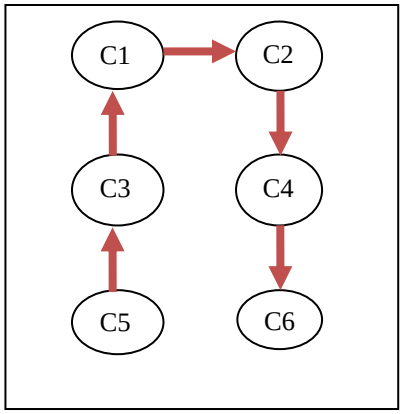
Now, coupling complexity of black box component (CCBC) is given as- CCBC = II + OI

CCBC is to be calculated separately for individual. So, complexity of different components is as follows:

CCBC (C1) = 1 + 1 = 2
CCBC (C2) = 1 + 2 = 3
CCBC (C3) = 3 + 2 = 5
CCBC (C4) = 2 + 1 = 3
CCBC (C5) = 1 + 1 = 2
CCBC (C6) = 1 + 2 = 3

ACC = $\sum_{i=1}^{n} CCBC_i / n.$
ACC= (2+3+5+3+2+3)/6 = 3

Case 4: Fourth system having six components with five interfaces.



Now, coupling complexity of black box component (CCBC) is given as- CCBC = II + OI

CCBC is to be calculated separately for individual. So, complexity of different components is as follows:

CCBC (C1) = 1 + 1 = 2
CCBC (C2) = 1 + 1 = 2
CCBC (C3) = 1 + 1 = 2
CCBC (C4) = 1 + 1 = 2
CCBC (C5) = 0 + 1 = 1
CCBC (C6) = 1 + 0 = 1

ACC = $\sum_{i=1}^{n} CCBC_i / n.$
ACC= (2+2+2+2+1+1)/6 = 5/3= 1.66

TABLE 1
REPRESENTING NUMBER OF COMPONENTS AND INTERFACES

Case No.	Components	Interfaces	ACC
1	4	4	2
2	4	5	2.5
3	6	9	3
4	6	5	I.66

The results presented in table1 shows how the number of interfaces affects the coupling of software. There are four cases has been taken with different number of component and interfaces to validate the proposed metrics. Case1 and case2 are having same number of component but number of interfaces is different. So the coupling of case2 is greater than case1 and again considers the case3 and case4 both have same number of component and interfaces are different and the value of case 3 is greater than case 4 because case 2 and case 4 are having more number of interfaces then the component after analysis the coupling value of all cases.

V. CONCLUSION

Coupling metric for the component-based system which is developed by using black box component has been proposed in this paper. Four cases with different number of component and interfaces are used to validate the proposed metrics using graphic representation. The result of all cases shows how the number of interfaces affects the complexity of software. More number of interfaces means high coupling which increase the complexity of software. Higher coupling means higher complexity which increase the testing and maintenance efforts of the software.

REFERENCES

[1] Luiz Fernando Capretz and Miriam A. M. Capretz, "Component-Based Software Development," *The 27th Annual Conference of the IEEE Industrial Electronics Society*, 2001.
[2] E.B. Allen, T.M. Khoshgoftaar and Y. Chen, "Measuring coupling and cohesion of software modules: an information-theory approach", *Proc. metrics*, Published by *the IEEE Computer Society*, pp.124, 2001.
[3] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design", in *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.
[4] V. L. Narasimhan and B. Hendradjaya, "A New Suite of Met-rics for the Integration of Software Components," University of Newcastle , Australia.

[5] V. L. Narasimhan and B. Hendradjaya, "Some theoretical considerations for a suite of metrics for the integration of software components", *Information Sciences*, vol. 177, pp. 844-64, 2007.
[6] N. S. Gill and P.S. Grover, "Component-Based Measure-ment: Few Useful Guidelines", *ACM SIGSOFT SEN*, vol. 28, no. 6, pp. 30, 2003.
[7] N. S. Gill and P. S. Grover, "Few important considera-tions for deriving interface complexity metric for compo-nent-based systems", *ACM SIGSOFT Software Engineering*, Notes, vol. 29, March 2004.
[8] U. Kumari and S. Upadhyaya, "An Interface Complexity Measure for Component-based Software Systems", *International Journal of Computer Applications*, vol. 36, no. 1, December 2011.
[9] A. Sharma, R. Kumar and P. S. Grover, "Evaluation of Complexity for Software Components", *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, Issue 5, pp.: 919-931, November 2008.
[10] L. Kharb and R. Singh, "Complexity metrics for component-oriented software systems", SIGSOFT Softw. Eng. Notes, vol.33, pp. 1-3, 2008.
[11] N.S. Gill and Balkishan, "Dependency and interaction oriented complexity metrics of component-based systems", SIGSOFT Softw. Eng. Notes, vol.33, pp. 1-5, 2008.
[12] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Trans. Softw. Eng*., vol.7, pp. 510-8S. M, 1981.
[13] G. Gui and P. Scott, "Ranking reusability of software components using coupling metrics," *Journal of Systems and Softwar*e, vol.80, pp. 1450-9, 2007.
[14] G. Gui and P.D. Scott, "Measuring Software Component Reusability by Coupling and Cohesion Metrics," *Journal of Computers*, vol.4, pp. 797-805, 2009.
[15] N. Kaur and A. Singh, "A Complexity Metric for Black Box Components," *International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307*, volume-3, Issue-2, May 2013.

