Intern. J. Fuzzy Mathematical Archive
Vol. 7, No. 2, 2015, 185-194
ISSN: 2320 –3242 (P), 2320 –3250 (online)
Published on 22 January 2015
www.researchmathsci.org

International Journal of
Fuzzy Mathematical
Archive

A Review on Component Based Software Metrics

***A.Aloysius*[1] and *K.Maheswaran*[2]**

[1]Department of Computer Science, St. Joseph's College (Autonomous), Trichy – 620002 TamilNadu, India. e-mail: aloysius1972@gmail.com
Corresponding Author

[2]Department of Computer Science, St. Joseph's College (Autonomous), Trichy – 620002 TamilNadu, India. e-mail: mahes161@gmail.com

Received 2 November 2014; accepted 15 December 2014

Abstract. In the technological world every day number of software's are develop and available in the market but measuring the complexity and quality of the software is still a very big challenge. Component based software is emerging field and now-a- days, most of the software are developed by using the technique of component based software development (CBSD). So that the complexity, time, error factors were reduced and reusability is achieved. The success of the CBSD projects can be ensured only from the metrics that are previously proposed. In this paper, various component based metrics proposed by the researchers have been discussed and then suggested the future enhancement.

Keywords: Components, component based software development, metrics, black box

1. Introduction

Component-based software development (CBSD) is one of the most important, modern paradigm and is expected to be at the forefront of new approaches to the construction of large and complex software systems. The main objective of this approach is to minimize the development effort, time and cost by means of reuse and also improves the quality, productivity and maintainability of the software [8]. These advantages are mainly provided by the reuse of already built-in software components. A software component is a self-contained piece of software that provides clear functionality, has open interfaces and offers plug-and-play services. It can be regarded as a reusable software element such as a function, file, module, class or subsystem [5].

2. Existing metrics

Number of software metrics related to software complexity and quality assurances has been developed in the past and are still being proposed.

2.1. Metrics for structured and object oriented systems

Several traditional metrics was designed for structured systems among them developers often found that Wang [25], McCabe's Cyclomatic complexity metric, Halstead's complexity metric and Kafura's and Henry's fan-in, fan-out are most commonly used metrics [11,7,10]. For object oriented systems Chidamber and Kemerer metrics [6] is a

base of all metrics, Misra [21] suggested Complexity Metric of OOP's Based on Cognitive Weights and many researchers like Arockiam et al. [1,2], Misra et. al [20,21] proposed the various level of metrics of object oriented programs based on their perspective including cognitive aspect.

3. Metrics for component based systems

Many researchers like Vernazza et al. extended the CK metric [22], Salman's [12] considered components, connectors, interfaces, and composition trees as main attributes that determine structural complexity of a component based system. Bertoa et al. [4] proposed the metrics for software components to access their usability, Sharma et al. proposed interface complexity metric for software components by considering interface methods and their associated properties, arguments types and return types [3]. In this section, various metrics have been discussed corresponding to their complexity, quality characteristics and reusability of software component.

3.1. An interface complexity measure of a component

Software complexity cannot be computed by a single parameter of a component / program / software because it is multidimensional attribute of software. The major factors which contribute to complexity of a component-based software system are size and interface of each component. By taking only interface complexity into account, Usha Kumari [23] suggested that, In CBSD, a component is linked with other components and hence has interfaces with them. Two or more components are said to be interfaced if there is a link between them, where a link means that a component submits an event and other components receive it. The direction of the link indicates that which component requests the services or dependent on the other. Interface between two components can be through incoming and outgoing interactions. These both types of interactions add complexity to a component-based software system.

$$\text{Average Incoming Interactions Complexity (AIIC)} = \frac{\sum_{i=1}^{m} II_i}{m} \quad (1)$$

$$\text{Average Outgoing Interactions Complexity (AOIC)} = \frac{\sum_{i=1}^{m} OI_i}{m} \quad (2)$$

Average Interface Complexity of a Component Based System

$$\text{(AIC (CBS))} = \frac{\sum_{i=1}^{m} II_i}{m} + \frac{\sum_{i=1}^{m} OI_i}{m} \quad (3)$$

where m = Number of components in the Component Based System (CBS)
II = Incoming Interactions OI = Outing Interactions
$\sum$= Summation symbol, i = Index variable

In Kumari [23] an interface complexity measure has been proposed which takes into account – interaction complexity, an important aspect of complexity of a component-based system. The results show that the effect of this parameter on complexity of a component-based system is quite significant. The results agree with the intuition that higher interaction between components increases the complexity because of more coupling among components. The same has also been theoretically evaluated against Weyuker's properties.

3.2. Software quality metrics using component reusability

Trivedi et al. [14] estimate the software reusability in terms of software components we estimate the following metrics.

3.2.1. System coupling metrics (SCOUP)

The system coupling metrics (SCOUP) for Component Based Software System (CBSS) will be SCOUP = $\sum_{j=1}^{m} \frac{MV_j}{m}$ (4)

Here MV_j is the Coupling metrics for component j and m is the number of the components in CBSS [8].

3.2.2. System cohesion metrics (SCOM)

The systems Cohesion Metrics (COM) for CBSS will be SCOM = $\sum_{j=1}^{m} \frac{COM_j}{m}$ (5)

Here COM_j is the Cohesion metrics for component j and m is the number of components in CBSS [8].

3.2.3. System actual interface metrics (SAIM)

SAIM is the integration of the interface metrics of the total number of components

SAIM = $\sum_{j=1}^{m} \frac{AIMj}{m}$ (6)

Here m is the number of components.

3.2.4. Sole system complexity metrics (SSCM)

Sole System Complexity Metrics (SSCM) is the combination of above three system metrics with different weights for each metrics [8].

SSCM = α' * SCOUP +β' * SCOH + γ' * SAIM (7)

Here' α', β' and γ' and are the weights for system coupling metrics, cohesion metrics, interface metrics with the condition as α' +β'+ γ' =1.

However, when such data are used to compare the complexity levels among several software systems, the developers will know which CBSS needs more people and more time during the coding and testing stages, or they may expect the vulnerabilities will happen in which component according to the complexity metrics. The above metrics help to estimate the software reusability in a software program. Software components are one of the major factors that provide the software reusability. The system will check that the use of the component based approach in the system is favorable to the system or not.

3.3. Dependency analysis for component based systems

In Component-based development (CBD) paradigm, Component-based software system (CBSS) are established using a set of mutually dependent components which work together. Some of these components may be developed in-house, while others may be third-party components, without source code. The main objective of this approach is to minimize the development effort, time and cost by means of software reuse. CBSD advances quality, productivity, reliability and maintainability of the software system [24]. Dependency between components can be defined as the reliance of a component on other components to support a specific

functionality; therefore, we consider dependency as a binary relationship between two components: dependent and antecedent. Dependent component is one that related to its antecedents where changes in them might lead dependent to malfunction or fail. Antecedent is the component that has an effect on the dependent one if it is removed or modified. Sometimes it may occur that a component has to take help of other components to perform its functionality. A component A is dependent on component B means that A must be checked if B changes. Maximization of such components builds a CBS complex [16]. Interaction in component-based systems (CBS) takes place when a component delivers an interface and other components use it, and also when a component submits an event and other component receives it. Dependencies are promoted by interactions. Higher dependency leads to a complex system, which results in poor understanding and a higher maintenance cost. [19] Rani et al. proposed a minimum spanning tree approach to analyze dependency in Component Based Systems (CBS) [9].

3.3.1. Component dependency graph (CDG)

Component Dependency Graph of a CBS is defined as G=(S,D,s,t), is a directed graph, where S is a non empty set of vertices each represents a component in the system, D is a set of dependency edges between two vertices each represents a direct dependency between components, s is a starting node, t is a terminating node. Fig 1 describes the direct dependency, where
D={(A,B),(B,D),(C,D),(C,B),(C,A),(E,B),(E,D)}

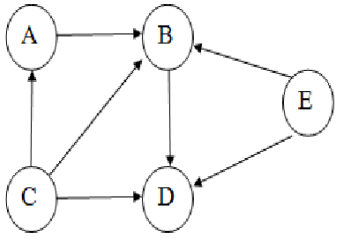

Figure 1: Component dependency graph

A new approach to analyze dependency in Component Based System (CBS). This approach contains the following steps:
1. Construct a Component Dependency Graph (CDG) of a Component Based System (CBS).
2. Assign weights to every edge of Component Dependency Graph.
3. Calculate the minimum spanning tree for CDG by any one of the existing algorithms (Prim's algorithm or Kruskal's Algorithm).
4. The dependency of the individual component is the minimum weight of that component.

First we calculate the dependency of each component using Minimum Spanning Tree (MST) in component based system and then calculate the dependency of each component using Analytical Hierarchal Process. Finally we calculate the Correlation Coefficient of the two techniques which shows that the technique is valid.

3.4. A complexity metric for black box components CCM (BB)

Measuring component complexity plays an important role in determining CBS system complexity. Because component complexity affects the complexity of whole CBS system, the component complexity is an important factor affecting the integration complexity, understandability, testability, maintainability etc of CBS system. But now a day's black box components are being provided by component vendors for reuse and most of the time source code is not provided with components which create difficulty in measuring component complexity. Kaur et al. [13] proposed a complexity metric for black box component CCM (BB). This metric is based on the component interface specification and use the concept of assigned weights. In black box components most of the times source code is not available so it is very difficult to guess or find the variables and it is also not possible to find the cyclometric complexity of methods in absence of source code. Thus the metric includes the concepts of interface methods complexity and coupling complexity between the components, which can be determined on the basis of component specification. Thus the black box component complexity may be defined as the sum of interface methods complexity and coupling complexity. The CCM (BB) metric has been defined to determine the overall complexity of a black box component.

3.4.1. Interface method complexity metric for black box component IMCM (BB)

$$IMCM(BB) = W_r + PCM(M) \quad (8)$$

where W_r represents the weight assigned to the category of return value's data type and PCM(M) is Parameters Complexity Metric for Method which calculates the complexity caused by parameters.

3.4.2. Parameters complexity metric for method PCM (M)

$$PCM(M) = a*W_{vs} + b*W_s + c*W_m + d*W_c + e*W_{vc} \quad (9)$$

where a,b,c,d,e represent counts and W_{vs},W_s,W_m,W_c,W_{vc} represent the assigned weights for very simple, simple, medium, complex and very complex data type categories for parameters of a method . High value of IMCM (BB) shows decrease in understandability and increase in testing effort.

3.4.3. A component coupling complexity metric for black box component CCCM(BB)

$$CCCM(BB) = FICM(BB) + FOCM(BB) \quad (10)$$

where FICM(BB) is Fan-in Complexity Metric which measures the coupling complexity due to incoming data from the other components and FOCM(BB) is Fan-out Complexity Metric which measures the coupling complexity due to outgoing data to other components.

Advantages of CCM (BB)

Easy to understand and use. No need of source code, it is based on component specification only. Interface Method Complexity Metric provides the estimation of testing effort and understandability. High value of IMCM(BB) for all the interface methods shows more testing effort and less understandability. Many coupling metrics consider only the number of interactions to show the extent of coupling. But CCCM (BB) not only considers the number of incoming and outgoing interactions but it also considers

the other factors affecting coupling complexity like number of components having impact on the considered component (f_{in}), number of components which may be affected by considered component (f_{out}), number of data items being passed between components and how many of them are creating data type incompatibility problem. Thus it provides more precise value of component coupling complexity. This metric provides the good indication of component integration and testing effort. High coupling complexity means more integration and testing effort. CCM (BB) includes interface methods complexity and coupling complexity. Thus it gives the overall complexity of component.

3.5. A metrics suite for measuring software components

Venkatesan et al. [24] proposed set of metrics for software components based on functional and non-functional aspects of the software. The author defines seventeen metrics for seven component characters. It includes three functional characters namely the suitability, accuracy and complexity and four non-functional characters namely the usability, maintainability, reusability and performance. The metrics are arrived at, based on a metric model and metric tree was shown in Fig. 2.

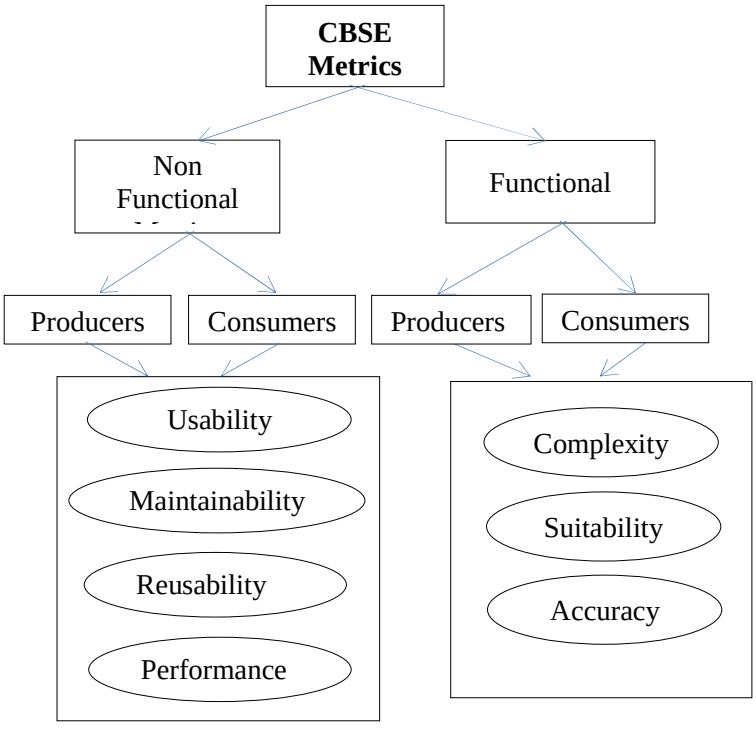

Figure 2: Metric tree

3.5.1. Suitability metric

Required Functionality (RF)

$$\boldsymbol{RF} = \frac{\text{Number of required functionalities provided by the component}}{\text{Total number of functionalities required by the Component} - \text{based}}$$

Extra Functionality (EF)

$$EF = \frac{\text{No. of extra functionalities provided by the component}}{\text{Total no. of functionalities required by the Component} - \text{based system}}$$

3.5.2. Component complexity metric

Component Coupling (COC)

$$\boldsymbol{COC} = \frac{\text{Number of other components sharing attribute or method}}{\text{Total number of possible sharing pairs in the component} - \text{based application}}$$

Constraints Complexity (CTC)

$$CTC = \frac{\text{Number of constraints}}{\text{Number of properties and operations in an interface}}$$

Configuration Complexity (CFC)

$$CFC = \frac{\text{Number of Configuration}}{\text{Number of context of use of the component}}$$

3.5.3. Reusability metric

The various classifications of reusability metrics is shown in Figure 3

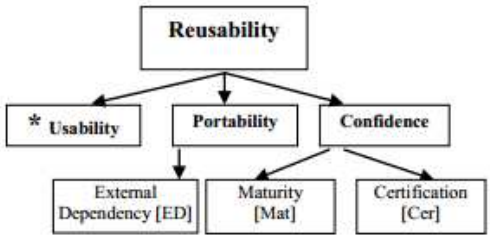

Figure 3. Component Reusability Tree

Portability:

External Dependency (ED)

$$ED = \frac{\text{Number of methods with parameters passed and returns values}}{\text{Total number of Methods (Excluding Read/Write Methods)}}$$

Confidence

Maturity [Mat]
Mat = DF + CR
DF = No. of Faults Detected.
CR = No. of Change Requests.

Certification [Cer]

$$Cer = \begin{cases} 1 \text{ (If Certification Exists)} \\ 0 \text{ (otherwise)} \end{cases}$$

In above eight different metric are shown with respect to suitability, complexity and reusability of a software component. Venkatesan et al. [24] proposed another nine metrics with respect to usability, Maintainability, Performance Accuracy. All the Seventeen metrics are validated with case study.

4. Future directions

CBSD is increasingly adopted technique in software development, but selecting the more appropriate less complex components for CBS to keep its complexity low, is still a difficult task. Thus appropriate evaluation of component complexity is a critical activity in the component selection process. Many researchers proposed various types of complexity metrics for measuring component complexity. But many of the existing metrics are based on the source code or internal details of component which may not be available in case of black box components. Another important issue of measuring the metrics is cognitive aspect of programming. None of the researchers were concentrate Cognitive complexity of component based programming. Existing metrics are not appropriate for determining component complexity. Thus from Table-1 the following aspects in metrics to be developed for component based software in future.

1) To choose less complexity based on the dependency between the components.

2) To measure the component complexity with / without going into internal details (Black box) of components.

3) To identify Cognitive complexity of software components.

Component / **Metric**	Interface complexity Metric	Quality metric using reusability	Dependency analysis	Complexity metric for Black Box	A Metrics Suite for Measuring Software Components
AIIC	✓	-	-	-	
AOIC	✓	-	-	-	
AIC	✓	-	-	-	
SCOUP [Coupling]	-	✓	-	-	✓
SAIM-	-	✓	-	-	
SSCM- [Complexity]	-	✓	-	-	✓
MST using CDG	-	-	✓	-	
IMCM(BB)	-	-	-	✓	
PCM(M)	-	-	-	✓	
CCCM(BB)	-	-	-	✓	
Reusability					✓
Suitability					✓
Cognitive Complexity	X	X	X	X	X

✓ - available X – not available

Table 1: Case study of various metric

5. Conclusion

This survey presents various metrics of component based software using their interface complexity, quality aspect using reusability, dependency and complexity of black box. Even though the component based software development is increasingly being adopted for software development. But measuring the black box component complexity during component selection is still a difficult task. By using metrics we can guess the component understandability, testability, integration effort, complexity of an interface, black box, analyze the dependency using minimum spanning tree approach with component dependency graph and quality aspect of component using reusability. Thus there is a need of complexity metric that can measure the component complexity with all the aspects of the software.

REFERENCES

1. L.Arockiam, A.Aloysius and J.Charles Selvaraj, Extended weighted class complexity: a new of software complexity for objected oriented systems, Proceedings of International Conference on Semantic E-business and Enterprise computing (SEEC), pp. 77-80, 2009.
2. L.Arockiam and A.Aloysius, On validating class level cognitive complexity metrics, *CiiT International Journal of Software Engineering and Technology*, 2(3) (2010) 152-157.
3. A.Sharma, R.Kumar and P.S.Grover, Evaluation of complexity for software components, *International Journal of Software Engineering and Knowledge Engineering*, 19(5) (2008) 919-931.
4. M.F.Bertoa, J.M.Troya and A.Vallecillo, Measuring the usability of software components, *Journal of Systems and Software*, 79(3) (2006) 427-439.
5. L.F.Capretz, A new component-based software life cycle model, *Journal of Computer Science*, 1(1) (2005) 76-82.
6. S.R.Chidamber and C.F.Kemerer, A metrics suite for object oriented design, *IEEE Transactions on Software Engineering*, 20(6) (1994) 476-49.
7. Halstead, Elements of Software Science, New York: Elsevier North Holland, 1977.
8. J.Chen et.al, Complexity metrics for component-based software systems, *International Journal of Digital Content Technology and its Applications*, 5 (3), (2011)235-244.
9. Jyoti Rani and Kirti Seth, Dependency analysis for component based systems using minimum spanning tree, *International Journal of Computer Applications*, 1 (2012), 11-16.
10. D.Kafura and S. Henry, Software quality metrics based on interconnectivity, *Journal of Systems and Software*, 2(2) (1981) 121-131.
11. McCabe, A Complexity Measure, *IEEE Trans. on Software Engg.*, SE-2 (4) (1976) 308-320.
12. N.Salman, Complexity metrics as predictors of maintainability and integrability of software components, *Journal of Arts and Sciences*, 2 (2006) 39-50.
13. N.Kaur and A.Singh, A complexity metric for black box components, *International Journal of Soft Computing and Engineering*, 3(2) (2013) 179-184.

14. P.Trivedi and R.Kumar, Software metrics to estimate software quality using software component reusability, *International Journal of Computer Science*, 9(2) (2012)144-149.
15. R.S.Chillar et al, Measuring complexity of component based system using weighted assignment technique, 2nd International Conference on Information Communication and Management, 55, (2012),
16. Ratneshwer and A.Tripathi, Dependence analysis of component based software through assumptions, *International Journal of Computer Science Issues*, 8(4) (2011) 149-159.
17. S.Khimta, P.S.Sandhu and A.S.Brar, A complexity measure for java bean based software components, *World Academy of Science Engineering and Technology*, 32 (2008) 479-482.
18. S.Misra and K.I.Akman, Weighted class complexity: a measure of complexity for object oriented system, *Journal of Information Science and Engineering*, 24 (2008) 1689-1708.
19. A.Sharma, P.S.Grover and R.Kumar, Dependency analysis for component-based software systems, 34(4) (2012) 1-6.
20. S.Misra, I.Akman and M.Koyuncu, An inheritance complexity metric for object-oriented code: A cognitive approach, *Indian Academy of Sciences*, 36(3) (2011) 317–337.
21. S.Mishra, An object oriented complexity metric based on cognitive weights, Proc. 6th *IEEE International Conference on Cognitive Informatics* (ICCI'07), 2007.
22. T.Vernazza and G.Granatella, Defining metrics for software components, D Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, XI (2000) 16-23.
23. U.Kumari and S.Upadhyaya, An interface complexity measure for component-based software systems, *International Journal of Computer Applications*, 36(1) (2012) 46-52.
24. V.P.Venkatesan and M.Krishnamoorthy, A metrics for measuring software components, *JCIT*, 4(2) (2009) 138-153.
25. Y.Wang and J.Shao, A new measure of software complexity based on cognitive weights, *Can. J. Elect. Comput. Eng.*, 28(2) (2003) 69-74.