Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Component Complexity Metrics : A Survey

Navneet Kaur, Ashima Singh
Computer Science and Engineering Department
Thapar University, Patiala, India

Abstract— Measuring and controlling the software complexity is an important aspect during each software development paradigm. Because the complexity of software affects many other attributes like testability and maintainability etc. So the researchers proposed many software complexity metrics and most of the metrics are based on the source code of the software . But nowadays Component Based Software Development (CBSD) is becoming the trend for software development. This approach is based on constructing the software system by integrating the prefabricated software components. The quality of resulting system depends upon the complexity of the composed components. So evaluation of component complexity is a critical activity in the component selection process for CBSD. In this paper the various complexity metrics relevant for measuring the component complexity have been discussed with their limitations and a solution has been proposed.

Keywords— CBSD , component complexity, complexity metrics, software complexity.

I. INTRODUCTION

Nowadays Component Based Software Development (CBSD) is becoming the modern approach for the construction of large and complex software systems . This approach focuses on developing software systems by integrating prefabricated software components. The main objective of this approach is to minimize the development effort, time and cost by means of reusing the existing software components . CBSD improves quality, productivity and maintainability of the resulting software. The following Fig. 1 represents the technique for CBSD.

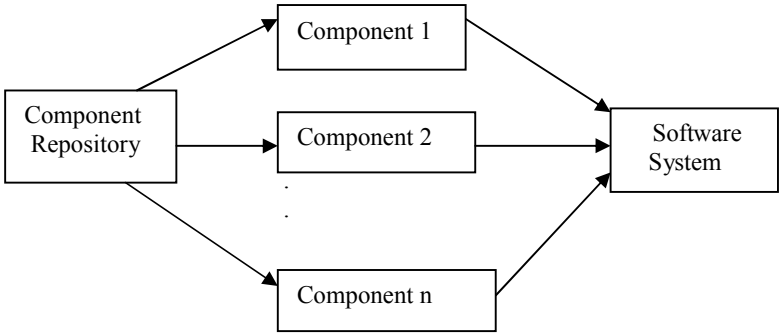

Fig.1 Representing CBSD technique

But measuring the component complexity during the component selection is also an important aspect of CBSD . So various types of complexity metrics have been proposed for measuring the software complexity. But due to black box nature of components many of existing metrics cannot be applied for measuring component complexity. In this paper the various complexity metrics relevant for measuring the component complexity have been discussed with their limitations and a solution has been proposed.

II. STUDY OF VARIOUS EXISTING METRICS

Software complexity cannot be removed completely but it can be controlled . But , for controlling the software complexity effectively , software complexity metrics are required to measure it. So many researchers have proposed various metrics for evaluating and predicting software complexity. This section describes various metrics which may be applied for measuring component complexity.

A. Traditional Software Complexity Metrics

Traditional software complexity metrics have been designed and applied for measuring the software complexity of structured systems since 1976 . Among these metrics developers often found that Lines of Code , McCabe's Cyclomatic complexity metric, Halstead's complexity metric and Kafura's & Henry's fan-in, fan-out are most commonly used metrics [9,10].

Metric 1: Lines of Code (LOC)

LOC metric is based on the size of methods. This metric gives measure of physical lines , statements , and/or comments. High value of this metric shows more complexity[6,10].

Metric 2: McCabe's Cyclomatic Complexity Metric

McCabe's Cyclomatic Complexity Metric is based on the program graph and is defined as[1,21] :

$$V(G) = e - n + 2p$$

Where e, n and p represents the number of edges, number of nodes in the graph and no. of connected nodes respectively. This metric gives the measure of independent algorithmic test paths. More independent paths means more testing effort.

Metric 3: Halstead's Complexity Metric

Halstead's Complexity Metric attempts to estimate the programming effort[1,10]. It measures complexity by summarizing the number of operators and operands contained in a program. The measurable and countable properties are:

- n1 = number of unique or distinct operators appearing in that implementation
- n2 = number of unique or distinct operands appearing in that implementation
- N1 = total usage of all of the operators appearing in that implementation
- N2 = total usage of all of the operands appearing in that implementation

Then the vocabulary , n of the program is defined as:

$$n = n1 + n2$$

The implementation length, N of the program is defined as :

$$N = N1 + N2$$

From the length and vocabulary, the volume, V of the program is defined as :

$$V = N \log_2 (n)$$

The difficulty, D of the program is defined as :

$$D = (n1 * N2) / (2 * n2)$$

And effort, E is defined as:

$$E = D * V$$

Metric 4: Henry's and Kafura's Metric

Henry and Kafura also proposed the complexity metric based on the number of local information flows entering (fan-in) and exiting (fan-out) in each procedure. This metric is given as :

$$\text{Complexity} = (\text{Proc. Length}) * (\text{fan-in} * \text{fan-out})^2$$

Where length is any measure of length such as Lines of Code or alternatively McCabe's Cyclomatic complexity is sometimes substituted. All these four metrics can be applied for measuring the component complexity. But these metrics are applicable at the method level and they are based on the availability of source code of component which may not be available in the case of black box components. Thus traditional software complexity metrics are not appropriate for measuring component complexity.

B. Object Oriented Software Complexity Metrics

The most impressive findings related to Object – Oriented metrics were the one proposed by Chidamber and Kemerer. They have proposed six complexity metrics[1,18,22]. These complexity metrics are:

Metric 1: Weighted Methods Per Class (WMC)

Weighted Methods Per Class metric is intended to count the combined complexity of local methods in a given class. WMC is defined as the sum of complexity of a class's local methods.

Consider a Class C, with methods M1... Mn that are defined in the class and let c1... cn be the complexity of the methods. Then:

$$WMC = \sum_{i=1}^{n} Ci$$

If we consider the complexity of the method as unity, then WMC is equal to the Number of Methods in a class (NOM) . The greater value of this metric shows more complexity, increase in testing effort and decrease in understandability .

Metric 2: Depth of Inheritance (DIT)

Depth of Inheritance metric is for class . The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree. It is measured by the number of ancestor classes. High value of DIT shows high design complexity. But it represents the greater potential for reuse of inherited methods.

Metric 3: Response For Class (RFC)

The RFC gives the count of all methods that can be invoked in response to a message to an object of the class or by some method in the class. The high value of RFC shows high complexity of the class. If a larger number of methods can be invoked in response to a message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester.

Metric 4: Coupling Between Objects (CBO)

For a given class, this metric measures the number of other classes to which the class is coupled. Excessive coupling prevents reuse. The more independent a class is, easier it is reused in another application. High value of this metric shows the poor design, difficulty in understanding, decrease in reuse and increase in maintenance effort.

Metric 5: Lack of Cohesion Method (LCOM)

The cohesion of a class is characterized by how closely the local methods are related to the local instance variables in the class. LCOM gives the number of disjoint sets of local methods. A highly cohesive module should stand-alone and high cohesion indicates good class subdivision. It implies simplicity and high reusability.

Metric 6: Number of Children (NOC)

Number of Children metric is based on a node (class) of inheritance tree. This metric gives the number of immediate successors of the class. High value of NOC shows more reuse, poor design and increase in testing effort. Although CK metric suite has been accepted widely but empirical validations of these metrics in real world software development settings are limited. Various flaws and inconsistencies have been observed in this suite of six class-based metrics. Among these six metrics WMC metric is based on the source code which may not be available in case of black box components. Thus this metric suite is not sufficient for measuring component complexity. Mishra[12] proposed a metric for determining the class complexity at method level by considering internal structure of the methods. Fothi et al. [13] designed a class complexity metric which is based on the complexity of control structures, data and relationship between data and control structures. But the metrics proposed by Mishra and Fothi et. al also depends upon the availability of source code . Thus these metrics are also not suitable for measuring black box component complexity.

C. Component Complexity Metrics

Many researchers have proposed various metrics for determining component complexity by considering different aspects , some of the existing component complexity metrics have been discussed below :

a) Tullio Vernazza et al.'s (2000) Metrics

Tullio Vernazza et al. extended the CK metrics (Chidamber and Kemerer ,1994). Authors proposed new metrics corresponding to each CK metric[23] . The definitions of these metrics have been given below **:**

Extensions for NOM : weighted classes per component and number of classes

A component may consists of a group of classes and the complexity of the various classes influence the complexity of the resulting component. More complex classes means ,more difficulty in understanding and maintenance , and consequently the component will be complex and difficult to maintain. Thus Weighted Classes Per Component (WCC) metric has been defined as:

$$WCC = \sum_{i=1}^{m} NOM(C_i)$$

Where $NOM(C_i)$ is the complexity of i^{th} class and m represents the number of classes in the component.

Extensions for DIT: maximum of the DIT and mean of unrelated trees

The high value of DIT is used to identify classes that are hard to maintain. The effort in maintaining group of classes can therefore be indicated by the values of DIT. But the extended metric considers the highest value of DIT, MAXDIT, and the mean of DIT of unrelated trees (MUT). Where the mean of DIT of unrelated trees is the mean of the values of MAXDIT for each independent class hierarchies.

Extensions for NOC: number of children for a component

This metric gives the number of children of all the classes in the component. This metric is called as number of children for a component (NOCC). It appears that NOCC is an indicator of reuse inside the component.

Extensions for CBO: external CBO

This metric measures the level of coupling for a component . It gives the number of external classes coupled to the component. This metric is called external CBO (EXTCBO).

$$EXTCBO = \sum_{i=1}^{m} e_i$$

Where e_i is the number of external classes coupled to the class C_i and m is the number of classes in a component.

Extensions for RFC: response set for a component

The response set of a component (RFCOM) gives the number of all the methods in the member classes and the methods called by those classes. RFCOM is the sum of the values of RFC for all the classes in the set.

$$RFCOM = \sum_{i=1}^{m} RFC(C_i)$$

Measuring Component Cohesion

Authors have discarded LCOM because it is not possible to verify it with the BMB properties. Instead they proposed for measuring component cohesion, which gives the number of internal classes to which a class is coupled normalized with the number of the possible coupling relationship among the classes.

These metrics are validated against the theoretical properties by Briand et.al (1999). But there is no empirical validation conducted for these metrics against any industry project, thus leaving the work incomplete.

b) Salman's (2006) System Complexity Metric

Salman proposed several metrics for measuring a component based system's complexity by mainly focusing on its structural complexity[14]. Author considered components, connectors, interfaces, and composition trees as main attributes that determine structural complexity of a component based system . The different metrics have been given below:

Component Metrics : These metrics characterize the components in a system , these metrics are : Total Number of Components (TNC) in a system, Average Number of Methods per Component (ANMC) and Total Number of Implemented Components (TNIC).

Connector Metrics : These metrics characterize the connectors in a system, these metrics are : Total Number of Links (TNL) – total number of links in the design of a system, Average number of links between components (ANLC) and Average number of links per interface (ANLI).

Interface Metrics : These metrics characterize interfaces in the system, these metrics are: Average Number of Links Per Interface (ANLI) and Average Number of Interfaces Per Component (ANIC).

Composition Tree Metrics : These metrics characterize the composition tree , these metrics are : Depth of the Composition Tree (DCT) and Width of the Composition Tree (WCT).

This metric set is validated using a set of properties adopted from the existing proposals (Weyuker, 1988). However , the proposed metrics are very basic in nature . These are based on just the total number and does not consider the complexity of individual interface.

c) Bertoa et al.(2006) Metrics

Bertoa et al. proposed the metrics for software components to access their usability[15]. Usability is a quality criterion in ISO 9126 quality model, which covers five sub – characteristics namely, Understandability – it is the capability of the component to enable the user to understand whether the component is suitable, and how it can be used for particular tasks and conditions of use, Learnability - it is the capability of the software component to enable the user to learn the application, Operability – it is the capability of the software component to enable the user to operate and control it, Attractiveness – it is the capability of the software component to be attractive to the user, Usability Compliance – it is the capability of the software component to adhere to standards, conventions, style guides or regulations relating to usability. Out of these, only first three are considered relevant for determining software component usability. Several measurable concepts and attributes related with these three aspects of quality are explored. But most of the work proposed here is subjective and very difficult to measure on a real time application. So these metrics are not appropriate for measuring component complexity.

d) Cho et al.'s (2001) Metrics

Cho et al. proposed a complexity metric for software components[11]. This Component Complexity Metric takes into account 4 types of complexity metrics : Component Plain Complexity (CPC), Component Static Complexity (CSC), Component Dynamic Complexity (CDC), and Component Cyclomatic Complexity (CCC). The CPC gives the sum of elements of the component (classes, abstract classes, and interfaces), added complexity of all classes, and added complexity of all methods of the classes. The CSC metric measures the complexity of internal structure of a component. The CDC metric measures the complexity of message passing occurring internally in a component. All these metrics are used at design stage. But the CCC metric is applied after implementation. CCC metric uses McCabe's complexity metric

to measure complexity of methods of a class. But all these metrics are based on the analysis of source code of component. So these metrics cannot be used for measuring the black box component complexity, as the source code of these components is not available .

e) Gill and Grover (2004) Metric

Gill and Grover proposed a complexity metric called Component Interface Complexity Metric (CICM) [7]. CICM assumes interface signatures, constraints on the interfaces and packaging for different context of use , for determining the complexity. For each of these aspects, a definition is also proposed. However, work still lacks of any empirical evaluation and validation of the proposed metric.

f) Ardimento et al.'s (2004) Metrics

Ardimento et al. provided the study of impact of characteristics of individual components on the quality of the component based systems in which they are integrated. Some characteristics of the components are: Adequateness of the component with respect to the target system, which further depends upon the Functional Coverage and Compliance of component, Costs required for training people while using the component, Familiarity of the working team with the component, Level of support provided by the component's vendor. But the metrics proposed for these component characteristics are subjective and some of them have not been empirically validated which leaves the work incomplete.

g) Sharma et al.'s (2008) Interface Complexity Metric

Sharma et al. proposed interface complexity metric for software components by considering interface methods and their associated properties, arguments types and return types[20]. This metric has been evaluated on several Java Bean components and validated against execution time, readability and customizability. The results concluded that complex components take more execution time and these are difficult to maintain.

III. PROPOSED WORK

Component Based Software Development (CBSD) is becoming the trend for software development which is based on constructing the software system by integrating the prefabricated software components. But the quality of the resulting system depends upon the complexity of the composed components. So evaluation of component complexity is a critical activity in the component selection process. One necessary step is to define a set of metrics that offer useful and simple results for the component selection process . In this paper, various existing complexity metrics relevant for measuring the component complexity have been discussed in section II. But , as discussed above in section II, most of the existing metrics are based on the availability of source code or the internal details of component which may not be available in case of black box components. And some of the existing metrics are subjective in nature. So many of the existing metrics are not appropriate for measuring the component complexity.

Thus for the selection of less complex components for CBS, there is a need of complexity metric that can measure the component complexity without going into internal details of components.

We have defined a component complexity metric which is a combination of some of the metrics mentioned above with a new approach. The proposed metric is based on component interface specification only. This metric calculates the component complexity in terms of its interface complexity, which gives good indication of component reusability. Thus this metric can be used for selecting a less complex and more reusable component. We are trying to have sufficient experimental results to prepare a new paper (other than this survey).

IV. CONCLUSION

Although CBSD is increasingly being adopted for 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. So many of the existing metrics are not appropriate for determining component complexity. Thus for selecting the appropriate less complex components for CBS, there is a need of complexity metric that can measure the component complexity without going into internal details of components.

REFERENCES

[1] Seyyed Mohsen Jamali, "Object Oriented Metrics," Department of Computer Engineering ,Sharif University of Technology , January 2006.

[2] Luiz Fernando Capretz and Miriam A. M. Capretz, "Component-Based Software Development," The 27th Annual Conference of the IEEE Industrial Electronics Society,2001.

[3] Ben Whittle and Mark Ratcliffe, "Software Component Interface Description for Reuse," Software Engineering Journal, November 1993.

[4] P. Edith Linda, V. Manju Bashini and S. Gomathi, "Metrics for Component Based on Measurement Tools," International Journal of Scientific & Engineering Research, Vol 2, Issue 5, May-2011.

[5] Li, Henry, "Object-oriented metrics that predict maintainability," Journal of Systems and Software , Volume 23, Issue 2, pg: 111-122,1993.

[6] Dr. P. K. Suri and Neeraj Garg ,"Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse," International Journal of Computer Science and Network Security, VOL.9, No.5, May 2009.

[7] Nasib S. Gill and P. S. Grover, "Few important considerations for deriving interface complexity metric for component-based systems," ACM SIGSOFT Software Engineering Notes, Vol 29 , March 2004,

[8] D. Kafura and S. Henry, "Software Quality Metrics Based on Interconnectivity," Journal of Systems and Software, pp 121-131, June 1981.

[9] Usha Kumari and Shuchita Upadhyaya, "An Interface Complexity Measure for Component-based Software Systems," International Journal of Computer Applications , Vol 36, No.1, December 2011.

[10] Sheng Yu and Shijie Zhou, " A Survey on Metric of Software Complexity ," School of Computer Science and Engineering, University of Electronic Science and Technology ,China.

[11] Eun Sook Cho, Min Sun fim and Soo Dong Kim, "Component Metrics to Measure Component Quality," College of Computer & Information Science, Dongduk Women's University' Korea,2001.

[12] Sanjay Mishra, " An Object Oriented Complexity Metric Based on Cognitive Weights," Proc. 6th IEEE International Conference on Cognitive Informatics (ICCI'07), 2007.

[13] A. Fothi, J. Nyeky-Gaizler and Z. Porkolab, "The Structured Complexity of Object-Oriented Programs," Mathematical and Computer Modeling, 2003.

[14] Nael Salman, "Complexity Metrics As Predictors of Maintainability and Integrability of Software Components," Journal of Arts and Sciences, 2006.

[15] M. F. Bertoa, J. M. Troya and A. Vallecillo, "Measuring the usability of software components," Journal of Systems and Software, Vol. 79, No. 3, pp.427-439, 2006.

[16] Sheng Yu and Shijie Zhou , " A Survey on Metric of Software Complexity," School of Computer Science and Engineering, University of Electronic Science and Technology , China..

[17] Victor R. Basili, Lionel Briand and Walcélio L. Melo, "A Validation of Object –Oriented Design Metrics As Quality Indicators ," Technical Report, Univ. of Maryland, Dep. of Computer Science, College Park,USA,April 1995.

[18] Parvinder Singh Sandhu and Dr. Hardeep Singh, "A Critical Suggestive Evaluation of CK Metric," Guru Nanak Dev Engineering College, Ludhiana,Punjab.

[19] Rajender Singh Chillar, Priyanka Ahlawat and Usha Kumari, "Measuring Complexity of Component Based System Using Weighted Assignment Technique," 2nd International Conference on Information Communication and Management, Singapore,2012.

[20] Arun Sharma, Rajesh 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.

[21] McCabe T, "A Software Complexity Measure", IEEE Trans. Software Engineering SE-2 (4), pp 308-320,1976.

[22] Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design" IEEE Transactions on Software Engineering, 20, No 6, pp. 476-49,1994.

[23] Tullio Vernazza, Giampiero Granatella, "Defining metrics for software components," Dipartimento di Informatica, Sistemistica e Telematica Università degli Studi di Genova

