International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-3, Issue-11)

Research Article **November 2014**

A Study of Component Based Complexity Metrics

Pooja Rana
Department of Computer science and Application
DAV Institute of Mgmt, Faridabad, India

Rajender Singh
Department of Computer Science and Applications,
MD University, Rohtak, India

Abstract:

In Component-Based Software Engineering (CBSE), software systems are mainly constructed with reusable components, such as third-party components and in-house built components. Component Based Software Development (CBSD) is used for making the software applications quickly and rapidly. In Component Based Development (CBD), the software product is built by gathering different components of existing software from different vendors. This process reduces cost and time of the software product. For the purpose of quality software, it is a measure of some property of a piece of software or its specifications. Software metrics are measures of the attributes of the software products and processes. Software metrics are quantifiable measures that could be used to measure different characteristics of a software system or software development process. For component-based systems, complexity metric are based on complexity attributes like interaction, coupling, cohesion, interface etc. In this paper we are trying to explore the different type of component based complexity metrics, attributes and limitation of the traditional complexity metrics.

Key Words Component based software development, complexity metrics, coupling, cohesion.

I. INTRODUCTION

Component-based software development approach is based on the idea to develop software systems by selecting appropriate off-the-shelf components and then to assemble them with a well-defined software architecture [15]. Component-based software development is associated with a shift from statement-oriented coding to system building by plugging together components. Component-based software engineering (CBSE) denotes the disciplined practice of building software from pre-existing smaller products, generally called software components, in particular when this is done using standard or de-facto standard component models. The popularity of such models has increased greatly in the last decade, particularly in the development of desktop and server-side software, where the main expected benefits of CBSE are increased productivity and timeliness of software development projects.

In component based software development, selection of suitable components at a proper time is a prerequisite to achieve objectives of improved product quality within time and budget constraints. Component evaluation is a critical activity in the component selection process. A component has to be evaluated technically (functionality and quality) as well as non-technically (cost, vendor support) (Brereton and Budgen, 2000). Several component quality attributes such as reusability, and maintainability depend upon the structural properties of its design (Cai *et al.*, 2000). One method of component evaluation is to evaluate its design for various concepts such as complexity, coupling, and cohesion.

Software metrics are useful in many ways to create quality software products within budget and time constraints. They help in project estimation and progress monitoring, evaluation of work products, process improvement, and experimental validation of best practices (Grady, 1994). Software Metrics can be defined as “The application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products.” The software has no physical attributes, conventional metrics are not much helpful in designing metrics for software. Number of metrics has been proposed to quantify things like size, complexity, and reliability of software product. Metrics provides the scale for quantifying the qualities, actual measurement must be performed on a given software system in order to reuse metrics for quantifying characteristics for a given software.

This paper is organized as follows. Firstly, it gives an introduction to the Component-Based Software Development process. Section 2 contains the types of software metrics and their limitations. Section 3 contains the related research papers used for the survey process. Section 4 contains the complexity metrics and there uses. Section 5 concludes the paper and the future work is present in the last section.

II. TYPES Of SOFTWARE METRICS

A) Traditional Software Metrics

As applied to the software product, software metrics that are usually related to the software quality basically measure or quantify the characteristics of the software. Traditional metrics have been applied to the measurement of software complexity of structured systems since 1976. Some common traditional software metrics are: 1) source lines of code (SLOC); 2) cyclomatic complexity; 3) function point analysis (FPA); 4) bugs per lines of code; and 5) code coverage.[5]

1) *Source lines of code*: The simplest software metric is the number of lines of code (LOC or KLOC for thousands of lines of code) used to measure the size of a software program by counting the number of lines in the text of the

program's source code. It was, and still is, used routinely to predict the amount of effort that will be required to develop a program, i.e., Effort = f (LOC), as well as to estimate programming productivity such as LOC/person-month or cost ($/LOC) once the software is produced.

2) *Cyclomatic complexity:* Cyclomatic complexity directly measures the number of linearly independent paths through a program's source code. It is used to measure code complexity by taking into account the program flow graph under the assumption that the effective complexity of a program lies in its structure rather than in a mere statement count.
3) *Function point analysis:* A function point is a unit of measurement to express the amount of functionality the system provides to the user. FPA metric is to compute the total function point value for the system based upon the number of user inputs, outputs, files, inquiries and interfaces. These function-point counts are then weighed (multiplied) by their degree of complexity, e.g., the weights for the five counts can be 4, 5, 10, 4 and 7at the average complexity, respectively.
4) *Bugs or faults per line of code:* The number of bugs or defects observed in a software product provides a metric of software quality. Some alternative measures have been proposed since there is no effective procedure for counting the bugs or defects in the program: (1) number of design changes, (2) number of errors detected by code inspectors, (3) number of errors detected in program tests and (4) number of code changes required.
5) *Code coverage:* Code coverage, an indirect measure of quality, is a quantitative measure used in software testing. It measures the code lines that are executed for a given set of software tests and describes the degree to which the source code of a program has been tested. Meanwhile, it finds areas of a program not exercised by a set of test cases and asks for additional test cases to increase coverage. It is a form of testing that inspects codes directly and is therefore a form of white box testing.

B) Limitations of traditional software metrics

Traditional software metrics are usually applicable to small programs, whereas the metrics for CBSS should depend mainly on the granularity and interoperability aspects of the components. Size of a component is normally not known to the component developers, whereas most of the traditional size metrics such as SLOC and bugs (faults)/code line and code coverage are based on lines of code, which is not applicable to CBSS. Traditional cyclomatic complexity metric suite cannot be applicable either in CBSS because operator and operand counts are not known in CBSS and the number of linearly independent paths cannot be measured [5]. FPA depends on the weights that were developed in a particular environment, which arises about the validity of this method for general application even though some improved measures like adjusting the counting method have been taken. There are many inherent differences in CBSS and non-CBSS so that the traditional software metrics are inappropriate for CBSS. Besides, the traditional software metrics do not address the interface complexities and integration-level metrics, which are also not applicable to CBSS [5].

C) Component general metrics

The important and relevant metrics applicable for the component quality analysis during design stage are [18]

1) *Component Size Metric (CSM):* CSM should be based on the total number of sub-components such as classes or use cases
2) *Weighted Methods per Component (WMC):* The number of local methods defined in the component. WMC is related to size complexity. WMC is the indicator of development and maintainability complexity.
3) *Depth of Inheritance Tree (DIT):* The maximum depth of the component in the Inheritance tree. The deeper the component is in the inheritance hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict the component’s behavior.
4) *Number Of Children (NOC):* The number of immediate sub-components of component or the count of derived components. NOC measures inheritance complexity.
5) *Count of Base Components (CBC):* The number of base components. Like NOC, CBC measures inheritance complexity.
6) *Response set For a Class (RFC):* The set of methods that can potentially be executed in response to a message received by an object of that component. RFC is simply the number of methods in the set, including inherited methods.

III. LITERATURE REVIEW

Nael SALMAN(2006) [10] introduced a set of metrics for component oriented software systems. The work focuses mainly on the complexity that results mainly from factors related to system structure and connectivity. Structural complexity of component oriented systems has been defined. Metrics for measuring CO systems’ structural complexity have been also been defined and validated. Also, a new set of properties that a component-oriented complexity metric must possess are defined. They characterized the structured complexity using the attributes like components in the system, connectors between components, interfaces of each component, composition tree. There are further an opportunities to examine the relationships between metrics values and several other product quality factors like performance and reliability. **V. L. Narasimhan and B. Hendradjaya (2007) [8]** they propose two sets of metrics to measure complexity and criticality of large software systems designed and integrated using the principles of Component Based Software Engineering (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. **Gill and Balkishan's (2008) complexity metric [9]** A large number of dependencies among components of a system raise its level of complexity. Authors propose dependency oriented metrics to measure complexity of CBS. They classify dependencies in components of a component based system in two categories: internal dependencies (intra-component), and external dependencies (inter-component). Using the concept of external dependencies, they define two metrics for black box components: Component Dependency Metric (CDM), and Component Interaction Dependency Metric (CIDM). **Jianguo Chen; Hui Wang; Yongxia Zhou; Stefan D. Bruda(2011) [12]** investigating the improved measurement tools and techniques, i.e., through the effective software metrics. Upon the research on the classical evaluation measures for software systems, they argue the traditional metrics are not suitable for CBSS. Therefore they provide an account of novel software measures for component by adequate coupling, cohesion and interface metrics. The complexity metrics combined with three metrics on the CBSS level is also investigated. Their focuses are to evaluate both individual component and assembly relation between components at the design stage of CBSS development life cycle. They believe that their efforts may help to manage the complexity of CBSS and to validate the component/system at a very early stage in the software development process. **Shuchita Upadhyaya and Usha Kumari (2011) [4]** They have attempted to design an interface complexity metric for black-box components to quantify an important aspect of complexity of a component-based system. The proposed measure takes into account one major type of complexity of a component. It is due to its interactions (interfaces) with other components. Graph theoretic notions have been used to illustrate interaction among software components and to compute complexity. **Rajender singh chhillar and Praveen Kajla (Dec 2012)[20]** Component Composition Metric (CCM) and Component Ratio Metrics(CRM) CCM is used to determine the composition of Software Development using CBSD and thus helps in determining the efforts in the early stages of software development.CRM is used to determine the ratio of components in the system and thus indicates the percentage of components in the system . Using proposed metrics and effort calculation it observed that more than 50% cases lies in good and average category. The comparisons of 5-Components with 10-Components illustrates that the above percentage increases from small to big projects. **Navneet kaur and ashima singh (2013)[19]** in this paper a component complexity metric has been proposed which is based on component interface specification. By using this metric we can guess the component understandability, testability, integration effort (which is required to integrate this component with other components) and overall component complexity. Thus by measuring the component complexity during the component selection for component based software development and selecting a less complex component the overall complexity of CBS can be reduced. This will help in reducing the integration and testing effort, and increasing the maintainability. **Umesh Tiwari and Santosh Kumar (Jan 2014)[21]**
The purpose of this paper is to propose a method of finding the Cyclomatic complexity when integrated software system is characterized by various interacting links between the components. Some example cases are used to illustrate the computation of Cyclomatic complexity for interacting modules. McCabe's Cyclomatic complexity model given for a single component is used as the base. The proposed technique logically computes the number of independent paths (Cyclomatic complexity) for an integrated software system where multiple components interact with each other.

IV. COMPLEXITY METRICES

For component-based systems, complexity metrics are based on complexity attributes like interaction, coupling, cohesion, interface etc. Most of the metrics proposed so far are based on the source code of the component and therefore cannot be used by the application developers, who do not have the source code of these components. So, there is strong demand and need for designing of complexity metrics for black-box components, which may be used by the application developers to choose the best components and then finally produce better quality CBS. For black-box components, major complexity parameters are interface, integration and semantics. Interface complexity measures are the estimates of the complexity of interfaces. Interface defines provided services of a component and acts as a basis for its use and implementation. It acts as one of the major definitive source for component understanding and may be the only available source. An interface consists of a set of operations, which act as access points for interaction with the outside computing environment. Integration metrics are the measures of efforts required in the integration process of components and semantic measures estimate the complexity of relationship of components to application.
There are many component based software complexity Metrics.

A) Weighted Methods Per Class (WMC)

This metric gives the combined complexity of local methods in a given class. The greater value of this metric shows more
complexity, increase in testing effort and decrease in understandability.

B) Depth of Inheritance (DIT)

This metric is for class . It gives maximum length from the class node to root. More length means more complexity.

C) Response For Class (RFC)

The RFC metric gives the number of methods that can execute in response to a message sent to an object with in this class, using to one level of nesting**.**

D) Coupling Between Objects (CBO)

For a given class, this metric measures the number of other classes to which the class is coupled. High value of this metric shows the poor design, difficulty in understanding, decrease in reuse and increase in maintenance effort.

E) 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 is defined as the number of disjoint sets of local methods. High value of this metric shows good class subdivision.

F) Number of Children (NOC)

This metric is based on a node (class) of inheritance tree. It gives the number of immediate successors of the class. High

value of this metric shows more reuse, poor design and increase in testing effort.

G) Lines of Code (LOC)

This metric is based on the size of methods. It gives measure of physical lines, statements, and/or comments. High value of this

metric shows more complexity .

H) Cyclomatic Complexity (CC)

This metric measures the complexity of methods. It gives the measure of independent algorithmic test paths. More independent

paths means more testing effort.

I) Metrics for the Integration of Software Components

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.

$$\text{CPD}<\text{constituent\_type}> = \frac{\#<\text{Constituent}>}{\#\ \textit{Components}}$$

Where #<Constituent> is the number of lines of code, operations, classes, and/or 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 a component.

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

Where #I and #Imax represents the number of actual interactions and maximum available interactions respectively. If one

component provides interface and another components use it or if one component submits an event and another component

receive it, then it is called an interaction. When the density of interaction increases, complexity increases.

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

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

Where # Iin and # Imax_in represents the actual number of incoming interactions and maximum number of incoming interactions available in a component respectively . The incoming interaction may be defined as a received interface that is

required in a component or a received event that arrives at a component. 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.

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

Where # Iout and # Imax_out represents the actual number of outgoing interactions used and maximum number of outgoing

interactions available in a component respectively. The outgoing interaction may be defined as any provided interface used or a

source of event consumed.

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 .

$$\text{CAID}= \sum_{i=1}^{i=n} \frac{CIDn}{No\ of\ Components}$$

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

J) Criticality Metrics

1) *Link Criticality Metric (CRITlink) :*The CRITlink metric is defined as the number of components 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. The threshold is
considered as 8 links. The links are created from the facets of other components. If facets increase, criticality of that component increases.

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

CRITbridge = # bridge_component

Where # bridge_component represents the number of bridge components . A bridge component may be defined as a component
which links two or more components/ application. If there is a defect in bridge, the entire application might malfunction. More
number of bridge components result in more chances of failure. All the links provided by a bridge component are assigned a
similar weight in order to show that they belong to the same bridge component.

3) *Inheritance Criticality Metric(CRITinheritance):* The CRITinheritance metric is defined as the number of components, which become root or base for other inherited components.

CRITinheritance = # root _ component

Where # root_component 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 .

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

CRITsize = # size_component

Where # size_component represents the number of components which exceed a given critical size value. The size is determined
by considering the factors like LOC, number of classes, operations and modules in the application. Narasimhan and
Hendradjaya defined the threshold value as 1000 lines of code or 50 classes. So, the value for this metric is given as 1 if it
exceeds the threshold value.

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

CRITall = CRITlink + CRITbridge + CRITinheritance + CRITsize

K) Composition Metrics

To determine component composition and the efforts for the composition during the system development at the design stage of the CBSD a Component Composition Metric (CCM) and Component Ratio Metrics(CRM) are introduced in exploring component composition.

1) Component Composition Metrics: Component Composition Metric (CCM) is defined as

$$\text{CCM}= CC_{rc} + CC_{nc} + CC_{mrc}$$

where CC_{rc} is the sum of all the components from the Reusable Component pool, CC_{nc} is the sum of all the components which
developed from the beginning and CC_{mrc} is the sum of all the components which are modified from the existing pool.

Thus CC_{rc} is defined as

$$CC_{rc}= CC_{rc1} + CC_{rc2} +\ldots\ldots\ldots.+ CC_{rcn}$$

$$CC_{rc}=\sum_{i=0}^{n} CCrci$$

CC_{nc} is defined as

$$CC_{nc}= CC_{nc1} + CC_{nc2} +\ldots\ldots\ldots.+ CC_{ncm}$$

$$CC_{nc}=\sum_{j=0}^{m} CCncj$$

CC_{mrc} is defined as

$$CC_{mrc} = CC_{mrc1} + CC_{mrc2} + \ldots\ldots\ldots.+ CC_{mrcp}$$

$$CC_{mrc} = \sum_{k=0}^{p} CCmrck$$

Therefore CCM is defined as
$$CCM = \sum_{i=0}^{n} CCrci + \sum_{j=0}^{m} CCncj + \sum_{k=0}^{p} CCmrck$$

L) Composition Ratio Metrics (CRM)

From the Component Composition Metric (CCM) a few new metrics are also proposed to determine the ratio of composition and these metrics are called as Composition Ratio Metrics (CRM). CRM are helpful in determining the ratio of reusable component, modified reusable component and new components required for a software development. This composition helps in determining precious factors of development like time, cost, adaptability etc. So Component Ratio Metrics (CRM) are defined as

$$CRM_{rc} = \left(\frac{CC_{rc}}{CCM}\right) * 100$$

$$CRM_{mrc} = \left(\frac{CC_{mrc}}{CCM}\right) * 100$$

$$CRM_{nc} = \left(\frac{CC_{nc}}{CCM}\right) * 100$$

V. CONCLUSION And FUTURE SCOPE

Today, most of the software systems are developed by using the existing code or the available components. This concept of using existing code is called Reusability. Reusability is achieved by performing some interfacing between different software components. The software reusability is presented either in terms of some code or in terms of component objects. Component based software metrics are discussed at two levels: system level, and component level. Research in component based software metrics is still immature. No metric proposal (except Washizaki *et al.* (2003)'s) is based on any formal component specification format. There is lack of automated metric collection tools and due to this the number of empirical studies in this area is also very less. However, application of conclusions to real life situations needs further study and empirical support using data from industrial projects to validate these findings and to derive more useful and generalized results. Using data from industry implemented projects will provide a basis to examine the relationship between proposed metric values and several quality attributes of component-based systems.

REFERENCES

[1] A.W.Brown, K.C. Wallnau, "The Current State of CBSE," IEEE Software , Volume: 15 5, Sept.-Oct. 1998, pp. 37 – 46.
[2] Arti Chhikara and R.S.Chhillar (2011) Impact of Aspect Orientation on Object
[3] Cristian CIUREA (2011) Using Genetic Algorithms for Building Metrics of Collaborative Systems, *Informatica Economică* vol. 15, no. 1/2011
[4] G. Pour, "Component-Based Software Development Approach: New Opportunities and Challenges," Proceedings Technology of Object-Oriented Languages, 1998. TOOLS26., pp. 375-383
[5] Gill, N.S, Balkishan (2008): Dependency and Interaction Oriented Complexity Metrics of Component-Based Systems, ACM SIGSOFT Software Engineering Notes, 33 (2), pp. 1-5.
[6] Jianguo Chen and Hui Wang (2011); Complexity Metrics for Component-based Software Systems; International Journal of Digital Content Technology and its Applications. Volume 5, Number 3
[7] Majdi Abdellatief, Abu Bakar Md Sultan (2011), Component-based Software System Dependency Metrics based on Component Information Flow Measurements, ICSEA 2011 : The Sixth International Conference on Software Engineering Advances.
[8] N. S. Gill, P.S. Grover, "Component-Based Measurement: Few Useful Guidelines", ACM SIGSOFT Software Engineering Notes, vol. 28, no. 6, pp:1-6,2003.
[9] Navneet Kaur, Ashima Singh "A Complexity Metrics for Black Box Components", International Journal of soft computing and engineering. Vol 3, issue 2,May 2013
[10] Noel SALMAN (2006): Complexity Metrics As Predictors of Maintainability and Integrability of Software Components, Journal of Arts and Sciences, 5, pp. 39-50.
[11] Outi Raiha(2008) Applying Genetic Algorithms in Software Architecture Design, M.Sc. thesis, University of Tampere, Department of Computer Sciences
[12] Parul Gandhi and Pradeep Kr Bhatia " Analytical Analysis of Generic Reusability: Weyuker's Properties", International Journal of Computer Science, vol 9, issue 2, No 1 March 2012

[13] Rajender Singh Chhillar and Praveen Kajla " New Component Composition Metrics for Component Based Software Development", International Journal of Computer Application, Vol 60, No15, Dec 2012

[14] Sahra Sedigh-Ali and Arif Ghafoor (2003): Metrics and Models for Cost and Quality of Component-Based Software, Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)

[15] Sengupta, S., Kanjilal, A. (2011): Measuring Complexity of Component Based Architecture : A Graph Based Approach, ACM SIGSOFT Software Engineering Notes, 36 (1), pp. 1-10.

[16] Sharma, A., Grover, P.S., Kumar, R. (2009): Dependency Analysis for Component-Based Software Systems, ACM SIGSOFT Software Engineering Notes, 34 (4), pp. 1-6.

[17] Sonu Mittal and Pradeep Kr Bhatia "Predicting Quantitative Functional Dependency Metric Based Upon the Interf ace Complexity Metric In Component Based Software", International Journal of Computer Application, Vol 73, No 2, July 2013

[18] Umesh Tiwari and Santosh Kumar "Cyclomatic Complexity Metric for Component Based Software", ACM SIGSOFT Software Engineering Notes page 1 vol 39 No1, Jan 2014

[19] Usha Chhillar, Sucheta Bhasin (2011): A Journey of Software Metrics : Traditional to Aspect-Oriented Paradigm, 5th National Conference on Computing For Nation Development, 10th -11th March, 2011, New Delhi, pp. 289-293.

[20] *Usha Kumari and* Shuchita Upadhyaya *(2011):* An Interface Complexity Measure for Component-based Software Systems *International Journal of Computer Applications (0975 – 8887) Volume 36– No.1*

[21] V. Lakshmi Narasimhan, P. T. Parthasarathy, and M. Das (2009): Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE), Issues in Informing Science and Information Technology Volume 6, 2009

[22] W. Kozaczynski, G. Booch, "Component-Based Software Engineering," IEEE Software Volume: 155, Sept.-Oct. 1998, pp. 34–36.

[23] Xia Cai, Michael R. Lyu, Kam-Fai Wong (2000) Component-Based Software Engineering:Technologies, Development Frameworks, and Quality Assurance Schemes.