Volume 4, Issue 5, May 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Metrics Suite for Accessing the Reusability of Component-Based Software

Suchita Yadav, Dr. Pradeep Tomar, Sachin Kumar
School of ICT
Gautam Buddha University, Greater Noida, India

Abstract - In Component-Based Software Engineering (CBSE), it is necessary to measure the reusability of components in order to realize the reuse of components effectively because reusability is an effective way to improve productivity in CBSD; it is required to measure the reusability of components. This study will propose a modified reusability metrics suite and reusability assessment model by using complexity, reuse and adaptability factors. Whereas the complexity of a software component determines that how easy it is to adapt the component in the new context of use, Reuse of the component can also be used to infer how usable and how easy to adapt it, adaptability determines that how easy it is to adapt a component to the context of the developer.

Keywords - CBSE, CBS, Component Reusability, Complexity, Adaptability, Reuse

I. INTRODUCTION

Effective reuse of knowledge, processes, and products from previous software developments can increase productivity and quality of software projects [1].
According to *Szyperski*[2] "A software component is a unit of composition by third parties and it contains only contractually specified interfaces and explicit context dependencies. It can be deployed independently".
Component-Based Systems (CBS) have become more generalized approach for application development. The main advantages of CBS are reduced development time, cost and efforts along with several others. These advantages are mainly contributed by the reuse of already built-in software components. In order to realize the reuse of components effectively in CBS, it is required to measure the reusability of components.
In Component-Based Development (CBD) [3], applications are built from existing components, primarily by assembling and replacing interoperable parts. Thus a single component can be reused in many applications, giving a faster development of applications with reduced cost and high quality. Also, as components are reused in various applications, they are likely to be more reliable than software developed ab initio. The reason is that these components are tested under varieties of situations before being used in the application(s). However, in spite of all such advantages of reuse, there are so many cases where reuse may lead to the software failure. These reasons may be due to the lack of experience for reuse, lack of documentation, tools and methodology for reuse along with several others.
The CBD is characterized by two activities: development with reuse, and development of software systems for reuse of component. The activity of development with reuse is realized by developing software with reusable components, because reusability is the degree to which a component can be reused.

- **Software Reusability**

Software reusability is an effective way to improve productivity. Software reusability development differs from the traditional way of software development in that it affects software measure, therefore new way of software reusability metric is needed, and moreover new model of software reusability is needed to be established. In order to improve the quality and reusability of component, reusability model for software component is studied in order to given scientific and accurate evaluation standard.

- **Component-Based Reusability**

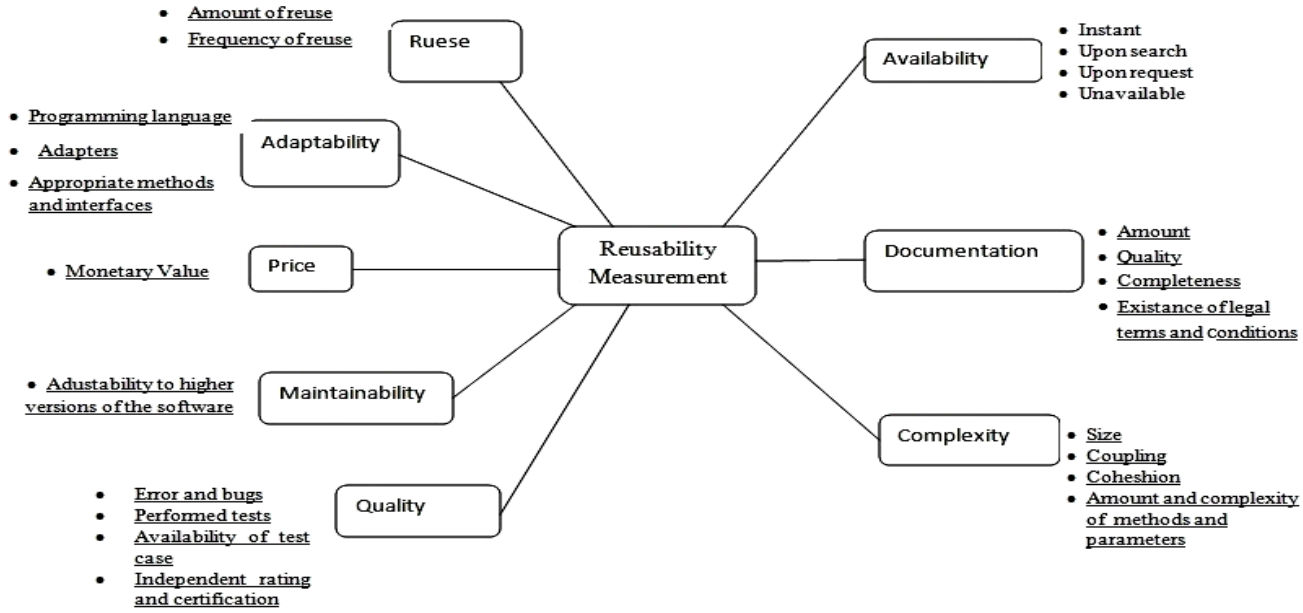
Reusability is the main aspect of soft component which differs from other software. Reusability is used to judge the degree of one component can be reused; it evaluates the reusability of component in order to help the developer to recognize the useful parts from the legacy system. The information of component reusability can also provide helpful choice for component user, and select the components that can be used in future software system.

- **Core elements of Reusability**

According to Danial Hristov et.al. [4] The core measurement factors are given below that can directly affect the reusability of component:

- **Availability:** The availability of a software component can determine how easy and fast is to retrieve it.

- **Documentation:** A good documentation can make the software component more reliable since it makes it easier to understand.
- Complexity The complexity of a software component determines how usable it is and how easy it is to adapt the software component in the new context of use.
- **Quality:** The quality of the component directly determines how usable it is in a given context. The quality of a component is regarded as a characteristic which describes how good it fulfills its requirements and also how error- and bug-free it is.
- **Maintainability:** The maintainability of a software component directly determines how usable it is for reuse. After the integration into the new system, the component should be able to adjust to the changes in the system along with its evolution.
- **Adaptability:** Adaptability is the ease with which a component can be adapted to fulfill a requirement that differs from that for which it was originally developed [4].
- **Reuse:** The actual reuse of the component can also be used to infer how usable and how easy it is to adapt it. The amount and frequency of reuse, especially in contexts similar to that of the developer can serve as reference points and she or he may select the component with the higher amount and frequency of reuse.
- **Price:** The price of the software component determines how expensive it is to reuse.



According to Danial hristov et.al. [4] The overview of the elements influencing reusability as discussed above is presented in the following figure 1.1.

II. RELATED WORK

Component-based software engineering (CBSE) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.

Hironori Washizaki et.al. [5] Proposed the importance of reusability of components in order to realize the reuse of components effectively and propose a Component Reusability Model for black-box components from the viewpoint of component users. The model defines a set of metrics to define quality factors that affect reusability. These metrics are:

I. Existence of Meta-Information (EMI) checks whether the BeanInfo class corresponding to the target component C is provided. The metric can be used by the user to understand the component's usage.

II. Rate of Component's Observability (RCO) is a percentage of readable properties in all fields implemented within the Façade class of a component C. The metric indicates that high value of readability would help user to understand the behavior of a component from outside the component.

III. Rate of Component's Customizability (RCC) is a percentage of writable properties in all fields implemented within Façade class of a component C. High value of the metric indicates the high level of customizability of component as per the user's requirement and thus leading to high adaptability. But if a component has too many writable properties, it will loose the encapsulation and can be used wrongly.

Self-completeness of Component's Return Value (SCCr) is the percentage of business methods without any return value in all business methods implemented within a component C, while Self-completeness of Component's Parameter (SCCp)

is the percentage of business methods without any parameters in all business methods implemented within a component C. The business methods without return value/parameter will lead to self completeness of a component and thus lead to high portability of the component.

G. Gui et.al. [6] Proposed an account of new measures of coupling and cohesion developed to assess the reusability of Java components retrieved from the internet by a search engine. These measures differ from the majority of established metrics in two respects: they reflect the degree to which entities are coupled or resemble each other, and they take account of indirect couplings or similarities. An empirical comparison of the new measures with eight established metrics shows the new measures are consistently superior at ranking components according to their reusability. Proposed new metrics are:

- Cohesion:

They develop a cohesion metric that takes account of both the degree of cohesion and transitive (i.e indirect) cohesion between methods. The methods of the class are the vertices. Suppose a class has a set of method members $M \equiv \{ M1, M2,\ldots Mm\}$ and let. $Vj \equiv \{Vj,1, Vj,2, \ldots. Vj,n\}$ be the instance variables accessed by method *Mj*.

They therefore define *SimD*(*i*,*j*), our measure of direct similarity of two methods, *Mi* and *Mj*, as:

$$\text{SimD}(i,j) = \frac{\left| V_i \cap V_j \right|}{\left| V_i \cup V_j \right|}$$

This measure is used to provide a measure of the cohesion of the class, *ClassCoh*

$$ClassCoh = \frac{\sum_{i,j=1}^{m} \text{Sim}(i,j)}{m^2 - m}$$

Where *m* is the number of methods in the class. Finally, the weighted transitive cohesion of the complete software system, *WTCoh*, is defined as the mean cohesion of all the classes of which it is comprised:

$$WTCoh = \frac{\sum_{i,j=1}^{m} ClassCoh}{n}$$

Where *n* is the number of classes in the system.

- Coupling

We begin by regarding any object-oriented software system as a directed graph, in which the vertices are the classes comprising the system. Suppose such a system comprises a set of classes $C \equiv \{ C1, C2,\ldots Cm\}$. Let $Mj \equiv \{Mj,1, Mj,2, \ldots. Mj,n\}$ be the methods of the class *Cj*, and *Rj*,*i* the set of methods and instance variables in class *Ci* invoked by class *Cj* for $j \neq i$ (*Rj*,*j* is defined to be null).

- We therefore define CoupD(i,j), our measure of direct coupling of class *Ci* to *Cj*, as:

$$\text{CoupD}(i,j) = \frac{\left| R_{i,j} \right|}{\left| R_i \right| + \left| R_j \right|}$$

- The weighted transitive coupling(*WTCoup*) of a system is thus defined:

$$WTcoup = \frac{\sum_{i,j=1}^{m} \text{Coup}(i,j)}{m^2 - m}$$

Where m is the number of classes in the system.

Boxall et.al.[7] proposed that the Understandability of a software component's interface is a major quality factor for determining reusability. To measure this, they have defined a set of metrics, including values such as Interface Size, Identifier Length or Argument Count. They have selected 12 components from different software systems in C and C++ to empirically validate their metrics and developed simple tools to automatically calculate them. The derived values have been compared against the expert knowledge of judging the reusability of these components.

Eun Sook Cho et.al. [8] Proposed a metrics suite for measuring the complexity, customizability, and reusability of software components. Complexity metric can be used to evaluate the complexity of components. Customizability is used to measure how efficiently and widely the components can be customized for organization specific requirement. Reusability can be used to measure the degree of features that are reused in building applications.

And for measuring the reusability of software components they proposed two approaches to measure the reusability of component. The one is a metric that measures how a component has reusability, while the other is a metric that measures how a component is reused in a particular application.

The first approach is component itself reusability (CR). CR metric may be used at design phase in a component development process. CR is calculated by dividing sum of interface methods providing commonality functions in a domain into the sum of total interface methods. We define the CR as following formula:

$$CR = \frac{\sum_{i=1}^{n}(\text{count } (CCMi))}{\sum_{j=1}^{n}(\text{count } (CIMj))}$$

Where:
Count (CCM): The count of each interface method for providing common functions among several applications in a domain, and
Count (CIM): The count of methods declared in interfaces provided by a component.
The second approach is a metric to measure particular component's reuse level per application in a CBSD. We call that Component Reuse Level (CRL). CRL is divided into CRL_{LOCs} and $CRL_{Func.}$ While CRL_{LOCs} is measured by using Lines of Code (LOC), CRL_{Func} is measured by dividing functionality that a component supports into required functionality in an application. The CRL_{LOCs}, expressed as a percentage, for a particular application is given by**:**

$$CRLLOCs = \frac{\text{Reuse } (C)}{\text{Size } (C)} * 100\ \%$$

where:
Reuse(C): The lines of code reused component in an application,
Size(C): The total lines of code delivered in the application.
The CRL_{Func} is expressed with:
$CRL_{Func}(C)$ = Sum of supported functionality in a component.
Li Yingmei et.al.[13] obtains reusability metric model based on the reusability of component. Method that was used to adjust the values according to the feedback information is described. They propose a Reusability measure Value (RMV) for component that can be obtained by denoting different weights for different sub-feature value, e.g. the following formula,
RMV=W1*F+W2*R+W3*U+W4*M+W5*P
Where *Wi*(i=1,…,5)denotes weights, F the functionality, R reliability, U utilizability, M maintainability, P portability. These weights have different values for different application field. For example, in finance system, the weight of reliability is relatively larger. The metrics ensure the quantified index; it can also evaluate the sub-feature of component reusability.
Rotaru [12] et.al. measure the reusability by proposing some metrics for adaptability and interface complexity of the component. Adaptability refers to the accommodation of the changes required in its environment.

III. PROPOSED WORK

This study is going to propose a reusability model and metrics suite that finds out the reusability of the component by considering on Complexity, Adaptability, and Reuse factors that directly affects the reusability.

- **Complexity**

The complexity of component is the main consideration over all the studies to find out many aspects of the component. This study is using complexity to find out the reusability of components by considering three main co-factors of complexity, i.e. cyclomatic complexity that is used after the implementation of component is finished and Cyclomatic Complexity of Component (CCC) is given as:

$$CCC = \sum_{i=1}^{m} CCi + DIM$$

Where:
$\sum_{i=1}^{m} CCi$: The sum of complexity of each method inside a class.
DIM: Depth of Inherited method inside a class.

Other co-factors are Coupling and Cohesion. Where the coupling is extent to which modules that make up a system are interdependent, a quantitative measure is to count the way in which one module may dependent on other. This study shows the coupling between components which will predict the interdependency of components on each other. This component coupling is given as Component Coupling Average (CCA_{coup}):

$$CCA_{coup} = \frac{\mathbf{MCFC}}{\mathbf{TC}}$$

Where
MCFC = method complexity based on class.
TC = total number of class in component.

Cohesion specifies the similarity of methods between classes. It is a measure of the extent to which the various functions performed by an entity are related to one another. Here this study proposes the Component Cohesion Average (CCA_{coh}) to measure the overall degree of similarity of methods in the class within a component. The CCA_{coh} with in a component is given as:

$$CCA_{coh} = \begin{cases} \frac{TCCh}{TC} & TCCh > 0 \\ 0 & otherwise \end{cases}$$

Where
TCC_h= Total Class Cohesion based on methods which is given by the maximum number of similarity of method situation in class.

TC = Total no classes in a component.

- **Adaptability**

The adaptability is also an important aspect for reusability that can be given by availability of appropriate methods and interfaces for adapting the component [8]. The adaptability of the component is given as Rate of Component Adaptability (RCA). RCA is depends on the Method Complexity (MC) and Interface Complexity (IC). MC is the methods dependency with in the class and given as

$$MC= \frac{\sum_{i=0}^{n} CCi}{n}$$

Where:

$\sum_{i=0}^{n} CCi$= sum of cyclomatic complexity of methods.

n = total numbers of methods in class.

Interface complexity: component may interact with other components only through their well-defined interface as they are black box in nature. Interfaces give the source of information to understand and reuse the component. For better reusability the interface complexity should be as low as possible. *Narasimhan and Hendradjaya* [15] proposed the complexity metrics [15] that have been widely accepted. Here for interface complexity i.e. Component Interaction Density (CID). The CID metric measures the ratio of actual number of interactions to the available number of interactions in a component.

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

Where #I and #Imax represents the number of actual interactions and maximum available interactions respectively. When the density of interaction increases, complexity increases.

- **Reuse**

The reuse can determine by the frequency of reuse that denotes the importance of a specific component for other component that depends on it. It can be expressed by class reusability and method reusability that gives us the reuse frequency of the classes and methods and its dependency on other classes and methods respectively. By defining the rank of each class and methods respectively be using the graph.

$$f(C_i) = \frac{\mathbf{r(Ci)}}{\sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{Ck}}$$

Where

$r(C_i)$ is the sum of all classes ranks

Where c is a class that used by the classes c1. . . cn.

And same we do for the method also

$$f(M_i) = \frac{\mathbf{r(Mi)}}{\sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{Mk}}$$

Where:

$r(M_i)$ is the sum of all Methods ranks

Where m is a method that used by the methods $m_1 . . . m_n$.

A class and method has a higher Reuse Frequency if many classes and methods depend on it respectively, or if some classes and methods depend on it that have high Reuse Frequency.

IV. RESULTS AND CONCLUSION

We are applying these metrics on single component code where for example we are considering the code of calculator in java having six methods in it namely add(), sub(), mul(), div(), mod(), pow() for concluding results from these metrics. So here the flow graph that is drawn by the code is given in figure 1.3 below:

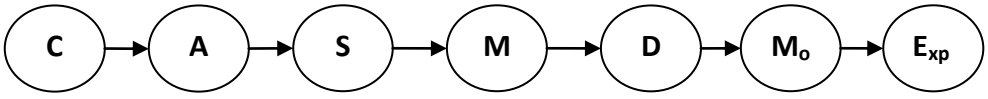


Figure 1.3: Flow graph of calculator

Now for each individual methods.

- **Now for adaptability**

- MC= $\frac{\sum_{i=0}^{n} CCi}{n}$

$\sum_{i=0}^{n} CCi$ = sum of cyclomatic complexity of methods = 1+1+1+1+1+2= 7

n = total no of methods = 6

MC= 7%6 =1.167

- Interface complexity (IC) = 0 (as there is a single component so no interactions available with other components)

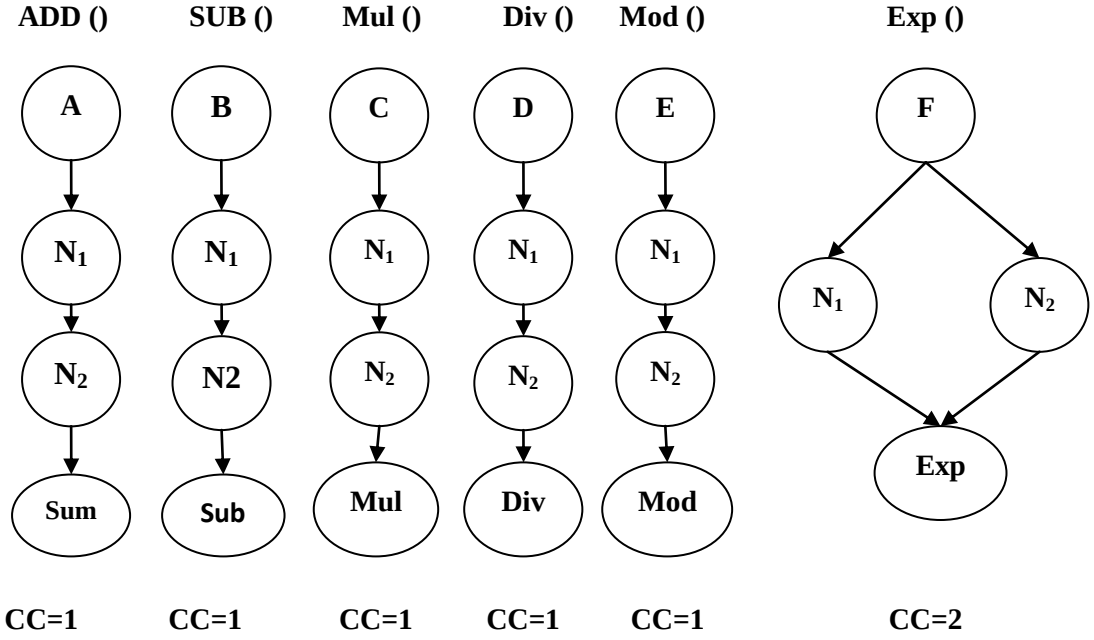
➢ **On the Basis of Reuse**

- $f(C_i)= \frac{\mathbf{r(Ci)}}{\sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{Ck}}$

where $r(C_i)$ is the sum of all classes ranks as we are having only one class in our class hierarchy so we can give it a rank 1 and the sum will be the 1 itself.
Total no of class is also 1 so denominator is 1
$f(C_i)$=1/1 = 1

- $f(M_i)= \frac{\mathbf{r(Mi)}}{\sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{Mk}}$

Where $r(M_i)$ is the sum of all Methods ranks it is sequential control flow so when we give the rank to the methods then all nodes rank will be 1.
So total rank= 1+1+1+1+1+1=6
Total no of methods = 6
$f(M_i) = 6/6 = 1$



➢ **Now for complexity**

- CCC=$\sum_{i=0}^{m} CCi$ + DIM

$\sum_{i=0}^{m} CCi$= The sum of cyclomatic complexity of each method inside a class = 1+1+1+1+1+2=7
DIM= Depth of Inherited method inside a class = single class= 0
CCC=7+0=7

Coupling

$CCA_{coup} = \frac{MCFC}{TC}$

MCFC = method complexity based on class = no. of methods in the class= 6
TC = total number of class in component = 1

$CCA_{coup}= \frac{6}{1} = 6$

Cohesion

$$CCA_{coh} = \begin{cases} \frac{TCCh}{TC} & TCCh > 0 \\ 0 & otherwise \end{cases}$$

Where
TCC_h= Total Class Cohesion based on methods which is given by the maximum number of similarity of method situation in class (there are six different methods in this example so there is no similarity of methods) = 0
TC = Total no classes in a component = 1
Here the otherwise condition is applicable for this example. There is 0 cohesion. This shows that this class is having high coupling and low cohesion so its is less reusable.
As in this study this has been seen that the continuous increase of the reused component number, in order to develop software, how to choose the component with improved reusability from the component library is a crucial problem for the developers of the component library and the persons of reusing components. As there is no structured and appropriate

metrics that can be found in literature is there for describing adaptability, reuse and complexity in the reusability of the component collectively. As by using these proposed metrics suite users of components can easily select those components with higher reusability and also this study's research approach can help and promote the activity of development with the reuse of existing black-box components.

REFRENCES

[1] Gianluigi Caldiera, Victor R. Basili, "Identifying And Qualifying Reusable Software Components", *IEEE-Computer, Volume: 24, Issue: 2, Page no. 61, 1991*.

[2] Szyperski, "*Component Software: Beyond Object-Oriented Programming", 2nd ed. Boston, MA: Addison-Wesley, 2002.*

[3] Arun Sharma, Rajesh Kumar and P S Grover, "Few Useful Considerations for Maintaining Software Components and Component-Based Systems", *ACM SIGSOFT Software Engineering Notes, Vol. 32, Issue 4, 2007.*

[4] Danail Hristov, Oliver Hummel, Mahmudul Huq, Werner Janjic, "Structuring Software reusability Metrics for Component-Based Software Development", *ICSEA 2012 : The Seventh International Conference on Software Engineering Advances,2012.*

[5] Hironori Washizaki, Hirokazu Yamamoto and Yoshiaki Fukazawa. "A Metrics Suite for Measuring Reusability of Software Components", *Proceedings of the 9th International Symposium on Software Metrics September, IEEE Computer Society, Washington, DC, USA, 2003, pp.211-223, 2003.*

[6] G. Gui and P.D. Scott, "New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability", *Proc. Of the Intern. Conference for Young Computer Scientists, pp.1181-1186, 2008.*

[7] Shrddha Sagar, NW Nerurkar, Arun Sharma, "A soft computing based approach to estimate reusability of software components", *ACM SIGSOFT Software Engineering Notes vol.35,No.-5,pp. 1-5, 2010.*

[8] M.A.S. Boxall and S. Araban, "Interface Metrics for Reusability Analysis of Components", *Australian Software Engineering Conference (ACWEC'04), Melbourne, Australia, pp.40-50, 2004*.

[9] EunSook Cho, Min Sun fim and Soo Dong Kim, "Component Metrics to Measure Component Quality", *Proceedings of the eighths Asia-Pacific Software Engineering Conference, 1530-1362/01,2005.*

[10] O. Hummel and C. Atkinson, "Using the Web as a Reuse Repository, Reuse of Off-the-Shelf Components", *Lecture Notes in Computer Science, vol. 4039, Springer, pp.298-311, 2006*.

[11] J. Poulin, "Measuring Software Reusability*", Proceedings of the 3rd International Conference on Software Reuse: Advances in Software Reusability, pp. 126–138, IEEE Computer Society Press, Los Alamitors, 1994.*

[12] Rotaru O P and Dobre M, "Reusability Metrics for Software Components", *In Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications, pp: 24-I, 2005.*

[13] Li Yingmei, Shao Jingbo, Xia Weining, "On Reusability Metric Model for Software Component", *Software Engineering and Knowledge Engineering: Theory and Practice Advances in Intelligent and Soft Computing Volume 114, pp 865-870,2012*.

[14] L.H. Etzkorn, W.E. Hughes Jr., and C.G. Davis, "Automated Reusability Quality Analysis of OO Legacy Software", *Information and Software Technology, vol.43, pp. 295-308, 2001.*

[15] V. L. Narasimhan , B. Hendradjaya, "A New Suite of Metrics for the Integration of Software Components" , *Proceedings of the The First International Workshop on Object Systems and Software Architectures, 2004*.

