World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:10, 2007

# Theoretical Considerations for Software Component Metrics

V. Lakshmi Narasimhan, and Bayu Hendradjaya



***Abstract*—**We have defined two suites of metrics, which cover static and dynamic aspects of component assembly. The static metrics measure complexity and criticality of component assembly, wherein complexity is measured using Component Packing Density and Component Interaction Density metrics. Further, four criticality conditions namely, Link, Bridge, Inheritance and Size criticalities have been identified and quantified. The complexity and criticality metrics are combined to form a Triangular Metric, which can be used to classify the type and nature of applications. Dynamic metrics are collected during the runtime of a complete application. Dynamic metrics are useful to identify super-component and to evaluate the degree of utilisation of various components. In this paper both static and dynamic metrics are evaluated using Weyuker's set of properties. The result shows that the metrics provide a valid means to measure issues in component assembly. We relate our metrics suite with McCall's Quality Model and illustrate their impact on product quality and to the management of component-based product development.

***Keywords*—**Component Assembly, Component Based Software Engineering, CORBA Component Model, Software Component Metrics.

## I. Introduction

COMPONENT-BASED Software Engineering (CBSE) has played a very important role for building larger software systems. Cost reduced [3,24] and shorter development time gives a good prospect for this type of development. Components are connected by assembling, adapting and wiring into a complete application. Although there is no IEEE/ISO standard definition that we know of, one of the leading exponents in this area, Szyperski [24], defines a software component as follows:

*"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".*

The definition allows assembling many components into a complete application; however we need a Standard way to make components work together. Some well-known Standards are Sun's Java, Microsoft .NET and Object Management Groups' (OMG) Corba Component Model (CCM). CCM uses the syntax of Component Interface Definition Language (CIDL)[30] for component integration and XML description for component assembly. However problems in using Standards still exist as described in [1,2,5,6,16].

Traditional metrics has to be redefined or enhanced to comply with component-based software development [1,13,15,22]. This research is an attempt to build a suite of software metrics, with particular emphasis on software component assembly. The suite accommodates static and dynamic aspects of component assembly. The static metrics measure the complexity and criticality of component assembly, while the dynamic metrics characterize the dynamic behaviour of a software application by recognizing component activities during run-time.

The suite can be used at different phase of development life cycle, particularly at the design, implementation and testing stages and the metrics can be used as indicators of activities at these stages. We expect the suite to aim software project managers to: 1) identify potential risks, and 2) uncover problem areas before they become harder to handle, so that software project managers can adjust the workflow or tasks in their projects**.**

The paper is organized as follows: section II presents the need for component integration metrics, while section III discusses the definition of the metrics and their description. The metrics are evaluated using the nine Weyuker properties in section IV. A discussion about the evaluation and how the metrics could be used over software projects is provided in sections V. Our conclusions of the research are described in section VI.

## II. Research Problem

Component based metrics have been proposed by several researches. For example, Dolado [9] validates the use of Lines Of Code (LOC) in counting the size of software, while Verner and Tate [26] estimate the number of LOC early in the software life-cycle. But since the use of LOC depends on the language of implementation, it is hard to predict the size of the software prior to implementing. This is more so for software components, which do not have information on their source code. Most other metrics on CBSE have aimed at the reusability of components [12, 20, 27, 28], while [10, 21] focus on the process and measurement framework in

Manuscript received September 22, 2005.

The authors are with the Faculty of Electrical Engineering and Computer Science, The University of Newcastle, NSW 2308, Australia (phone: +614921 6953; e-mail: lakshmi.narasimhan@newcastle.edu.au; url: http://www.cs.newcastle.edu.au/~narasimhan).

developing software components. Ebert [11] suggests some classification techniques to identify critical items in software project, which can be applied for a CBSE project, but he does not tackle the criticality aspects of component integration.

Specific issues on integration are discussed by Sedigh-Ali et al. [21], where the complexity of interfaces and their integration is interpreted as quality metrics. Cho et. al. [8] define metrics for complexity, customisability and reusability. They calculate the complexity of metrics by using the combination of the number of classes, interfaces and relationship among classes. They combine the calculation of cyclometric complexity with the sum of classes and interfaces, which need information from the source code. This is a shortcoming of their metrics, which is usable only when the developer has access to the source code.

We propose static metrics and dynamic metrics for component assembly[1]. Static metrics cover the complexity and the criticality within an integrated component. Static metrics are collected from static analysis of component assembly, while dynamic metrics are gathered during execution of complete application. The complexity and criticality metrics are intended to be used early during the design stage, while dynamic metrics are meant to be used at later stages. We consolidate our work and validate the metrics suite through Weyuker's evaluation criteria [29]. Furthermore, the suite integrates existing metrics available in the literature as an integrated package.

## III. Component Metrics

The proposed metrics use graph connectivity as a medium to represent a system of integrated components. Each node and link represent a component and their relationship with other components respectively. Interactions happen through interfaces and events arising or arriving in. If a component 'X' requires an interface that is provided by another component 'Y', then 'Y' will be the incoming interaction for 'X'. If a component 'X' publishes an event which is consumed by component 'Y', then 'X' is said to raise an outgoing interaction to 'Y'. OMG [30] defines a *provided interface* as a 'facet', a *required interface* as a 'receptacle', a *published event* as an 'event source' and a *consumed event* as an 'event sink'.

Fig. 1 shows a link between two components of system P and system Q where one could consider component B to be more complex than components A, D or X, because it has more links than the others. Therefore in one sense, B can be termed as a critical component. On the other hand, component X may not be complex, but it is critical for the correct operation of the integrated system. Here, component X functions as a bridge between two systems P and Q. The definition for criticality does not stop here and it has other dimensions (see more in section III, subsection B).

Similarly the complexity of a system depends on packaging density of components. For example, vertically connected components can be easily integrated as compared to multiply connected components[2]. In addition, facets of components also have roles in connectivity and quality metrics. Further, dynamic characteristics of components along with their constraints can aid the design of new type of metrics for quality, management and reliability measures. In order to compute the value for criticality and complexity, we have identified several conceptual views of the components and systems, based on the analysis of the static and dynamic characteristics of integrated components.

The proposed static metrics are Complexity Metrics, Criticality Metrics and Triangular Metric. Their descriptions are given in Table I. Table II shows the dynamic metrics descriptions. Dynamic metrics are collected during the execution of a complete application. Even though they cannot be used during the design phase, the results are still useful for testing and maintenance purposes. Gathering data for dynamic metrics is possible by instrumenting the source code, prior to compilation.

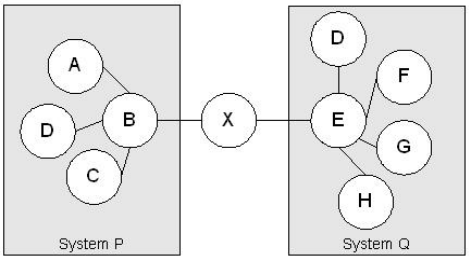


Fig. 1 Interacting systems and components

The three metrics, namely, CPD, CAID and $CRIT_{all}$ have different points of view. While *Component Packing Density* (CPD) is calculated from the density in the integrated components, *Average Interaction Density* (CAID) is derived from density of interaction within an integrated component. Both metrics, however, represent the complexity of the system. The last metric ($CRIT_{all}$) is based on the criticality of the component.

In order to get a better view of the complexity metrics, we can combine CPD and CAID into a two-axes diagram. By examining their value, we deduce the characteristic of software as follows (see illustration in Fig. 2):

Case 1: A low CPD value and a low CAID value: This condition might happen within a system, having low data processing and low computation, such as a simple transaction processing system.

Case 2: A low CPD value and a high CAID value. This condition might happen within a system, having low data

[1] In this paper, we differentiate the terms Component Assembly and Component Integration as follows: A 'Component Assembly' refers to system in which components have been integrated, whereas 'Component Integration' refers to the process of integrating components.

[2] Vertically connected components are linked consecutively so that every component has a maximum of two components linked. A component, which links to more than two other components is called multiply connected component.

processing and high computation, such as a compute-intensive real time system.

*Case 3*: A high CPD value and a low CAID value. This might suggest a transaction processing system, which is characterised by high volume of data processing with many components, but has low interaction among components.

*Case 4*: A high CPD value and a high CAID value. This condition represents a very complex system, which might has many classes or constituents within its components and high interactions among components.

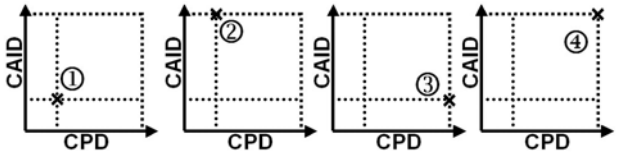


Fig. 2 Four possible values of CPD and CID

By combining information from the two axes diagram with new axis of criticality ($CRIT_{all}$), we can further characterize a system as shown in Fig. 3. A real-time system usually has higher criticality compared to a transaction-based system. A business application tends to have more components to access data, than a real time application. A list of application types can be found in [17], while distinguishing characteristics between business type applications and real-time systems can be found in [4]. Our results concur with these observations for transaction and real-time system. Two detailed examples developed can be obtained from the authors.

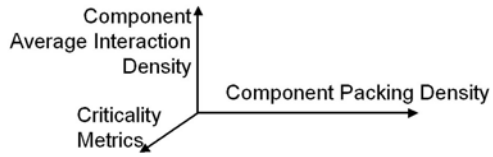


Fig. 3 Three axes of component complexity and criticality

## IV. Validating the Metrics using Weyuker Properties

Weyuker has proposed an axiomatic framework for evaluating complexity measures [29]. The properties are not without critique and these have been discussed in [12] and [14]. The properties, however, have been used to validate the C-K metrics by Chidamber & Kemerer[7] and, as a consequence, we will employ the same framework for compatibility's sake. We show the properties below, which are the modified definitions as provided by [12]; the original definitions are available at [29]. The properties are:

**Property 1**: There are programs P and Q for which M(P) ≠ M(Q)

**Property 2**: If c is non-negative number, then there are only finitely many programs P for which M(P)=c

**Property 3**: There are distinct programs P and Q for which M(P)=M(Q)

**Property 4**: There are functionally equivalent programs P and Q for which M(P) ≠ M(Q)

**Property 5**: For any program bodies P and Q, we have M(P) ≤ M(P;Q) and M(Q) ≤ M(P;Q)

**Property 6**: There exist program bodies P, Q and R such that M(P)=M(Q) and M(P;R) ≠ M(Q;R)

**Property 7**: There are program bodies P and Q such that Q is formed by permuting the order of statements of P and M(P) ≠ M(Q)

**Property 8**: If P is a renaming of Q, then M(P) = M(Q)

**Property 9**: There exist program bodies P and Q such that M(P)+M(Q) < M(P;Q)

*Property 1: There are programs P and Q for which M(P) ≠ M(Q)*

- An integrated component comprises various components having different constituents and different value of criticalities. As a consequence, the *Component Packing Density* (CPD) metric,

TABLE I
STATIC METRICS DESCRIPTIONS

| Name | Formulae | Description |
|---|---|---|
| Component Packing Density Metric | $CPD_{constituent\_type} = \frac{\#<constituent>}{\#components}$ | #<constituent_type> is one of the following: LOC, object/classes, operations, classes and/or modules in the related components. |
| Component Interaction Density Metric | $CID = \frac{\#I}{\#I_{max}}$ | #I is the number of actual interactions and #Imax is the number of maximum available interactions. |
| Component Incoming Interaction Density | $CIID = \frac{\#I_{in}}{\#Imax_{in}}$ | where, $\#I_{IN}$ is the number of incoming interactions used and $\#Imax_{IN}$ is the number of incoming interactions available. |
| Component Outgoing Interaction Density | $COID = \frac{\#I_{out}}{\#Imax_{out}}$ | $\#I_{OUT}$ is the number of outgoing interactions used and $\#Imax_{OUT}$ is the number of outgoing interactions available. |
| Component Average Interaction Density | $CAID = \frac{\sum_n CIDn}{\#components}$ | $\Sigma_n CID_n$ is the sum of interactions density of n component and #components is the number of the existing component |
| Bridge Criticality Metrics | $CRIT_{bridge}$=#bridge_component | #bridgecomponent is the number of bridge components. |
| Inheritance Criticality Metrics | $CRIT_{inheritance}$=#root_component | #root_component is the number of root components which has inheritance. |
| Inheritance Criticality Metrics | $CRIT_{inheritance}$=#root_component | #root_component is the number of root components which has inheritance. |
| Size Criticality Metrics | $CRIT_{size}$=#size_component | #size_component is the number of component which exceeds a given critical value. |
| #Criticality Metrics | $CRIT_{all}$= $CRIT_{link}$+$CRIT_{bridge}$ +$CRIT_{inheritance}$+$CRIT_{size}$ | <see above> |

*Interaction Density Metric*(IDM) and *Criticality Metrics* (CM) satisfy property 1.

- The use of CPD, CAID and CM in *Triangular Metric* (TM) yields different values for each component assembly, therefore TM satisfy property 1.
- During run time of various applications, we can always find different number of cycles, so the *Number of Cycle metric* (NC) satisfies property 1.
- Any executions of various component assemblies yield different number of active components at a time. Thus *Active Component* (AC) metrics satisfy property 1.

*Property 2: If c is non-negative number, then there are only finitely many programs P for which M(P)=c*

- For every application, there are a finite number of components with a finite number of constituents, a finite number of interactions within a component. So this property is met by the *CPD* and *IDM* metrics.
- Every component has its own criticality. With a given criticality number, there is only a finite number of components, thus property 2 is satisfied.
- The TM satisfies property 2, as there are only a finite number of components in component assembly.
- The same NC value can be found from different executions of various component assemblies. Therefore property 2 is satisfied.
- We can always find different applications with the same AC metrics value, thus property 2 is satisfied.

*Property 3: There are distinct programs P and Q for which M(P)=M(Q)*

- It is always possible to create a minimum of two combinations of components with their constituents and the metric value of *CPD* is the same. Therefore *CPD* metric satisfies this property.
- For IDM metrics, we can configure different interactions in more than one component that results in the same value, thus satisfying property 3.
- In integrated components, we can always find configuration of components, which have the same CM value and therefore property 3 is satisfied. Along the same logic property 3 is satisfied by TM.
- Various component assemblies can yield the same measurement for the NC and AC metrics. Therefore property 3 is satisfied by both metrics.

*Property 4: There are functionally equivalent programs P and Q for which M(P) ≠ M(Q)*

- If there are two integrated components, which perform the same functions, this does not imply that the CPD metric value will be the same. A given function can be built by several components and with different constituents. So the CPD metric satisfies this property. By the same logic, IDM also satisfies property 4.
- This property is satisfied by CM, since the same functionality for a given implementation can have different criticality.
- A given function of component assembly can be built using several types of components. Thus the TM and all *dynamic metrics* (NC and AC) also satisfy property 4.

*Property 5: For any program bodies P and Q, we have M(P) ≤ M(P;Q) and M(Q) ≤ M(P;Q)*

Let X and Y be two component assemblies such that Y consists of X and another component assembly.

- CPD value of X is no more complex than CPD value of Y. Therefore CPD metric satisfies property 5.
- For IDM, Y has the same or more interactions than X. So the IDM metrics satisfy property 5.
- The Combination of components yield equal or higher criticality for Y than X. Thus property 5 is satisfied by CM.
- It is implied that TM satisfies property 5, since property 5 is satisfied by CPD, IDM and CM.
- Execution of Y yields more cycles than X as the number of active components is higher. Therefore NC and AC satisfy property 5.

*Property 6: There exist program bodies P, Q and R such that M(P)=M(Q) and M(P;R) ≠ M(Q;R)*

- Let the component assemblies P,Q and R have respective CPD values of a/b, c/d and e/f, where a, c & e represent the number of constituents and b, d & f represent the number of components respectively. If the measurement on P is equal to Q, then ad = bc. Integration of (P;R) and (Q;R)yields (a+e)/(b+f) and (c+e)/(d+f), respectively. By working through the equation, we can conclude that (P;R) and (Q;R) will not have the same value, except when a = b = c = d = e = f = 1. This means that more than one component exists in the component assembly, which has more than one constituent. Therefore property 6 is satisfied.
- Using the above logic, one can note that the IDM metrics also satisfy property 6.
- Adding more components increases the probability of increase in the criticality value. Therefore property 6 is satisfied by CM.
- TM satisfies property 5, since property 5 is satisfied by CPD, IDM and CM.
- Adding more components increases the probability of more number of cycles and the number of active components at run-time. Therefore, Dynamic metrics (NC & AC) satisfy property 6.

*Property 7: There are program bodies P and Q such that Q is*

TABLE II
DYNAMIC METRICS DESCRIPTIONS

| Name | Formulae | Description |
|---|---|---|
| Number of Cycle (NC) | NC = # cycles | Where, #cycles is the number of cycles within the graph |
| Average Number of Active Components | $ANAC = \frac{\#activecomponents}{T_e}$ | #activecomponents is the number of active component and $T_e$ is time to execute the application (in seconds) |
| Active Component Density (ACD) | $ACD = \frac{\#activecomponent}{\#components}$ | #activecomponent is the number of active components and #component is the number of available components. |
| Average Active Component Density | $AACD = \frac{\sum_n ACD_n}{T_e}$ | $\Sigma_n ACD_n$ is the sum of ACD and $T_e$ is time to execute the application (in seconds). Execution time can be any of execution of a function, between functions or execution of the entire program. |
| Peak Number of Active Components | $AC_{\Delta t} = \max \{ AC_1,..,AC_n\}$ | $\#AC_n$ is the number of active component at time n and $\Delta t$ is the time interval in seconds. |

*formed by permuting the order of statements of P and M(P) ≠ M(Q)*

Permutation on component assembly does not affect on the metric values statically and dynamically. Therefore all proposed metrics satisfy property 7.

*Property 8: If P is a renaming of Q, then M(P) = M(Q)*

Renaming the components does not affect on all the metrics proposed, since the measurement only concerns the number of components and their constituents and the number of interactions. So all proposed metrics satisfy property 8.

*Property 9: There exist program bodies P and Q such that M(P)+M(Q) < M(P;Q)*

- If we have two component assemblies having x1 constituents and y1 components and another configuration with x2 constituents and y1 components, then we can compute CPD1=x1/y1 and CPD2=x2/y2 and we can integrate both configurations. The resultant CPD3=(x1+x2)/(y1+y2). The value of *CPD1+CPD2* will always be lesser than *CPD3*. Therefore property 9 is not satisfied.
- The same logic can be used for the *IDM* metrics, hence they do not satisfy property 9.
- For the CM metrics, combining more components will always add more links, bridges, size or possibly inheritance. So the resultant value of each types of criticality will increase the probability of the previous value. Therefore CM satisfies property 9.
- For TM, let two different component assemblies P and Q, have triangular metrics of (x1,y1, z1) and (x2, y2, z2) respectively. Combining P and Q into one component assembly with (x3, y3, z3) as its metrics value, we cannot always have x1+x2<x3, y1+y2<y3 or z1+z2<z3. Therefore triangular metrics does not satisfy property 9.
- Let P and Q be two different component assemblies with certain number of cycles Combining P and Q into one component assembly do not always increase the number of cycle possible. Therefore property 9 is not satisfied by NC metric.
- The above logic holds good for *active component metrics* also. The combination of component assemblies will increase the number of active components and therefore the AC metrics satisfy property 9.

## V. Discussion

We summarize the results of the paper through Table III, which shows that all the proposed metrics satisfy property 1-6 and 8, but fail to satisfy property 7. Property 9 is satisfied by *Criticality* and *Active Component* metrics only.

Property 7 requires that permutation should affect the value of complexity. But in the component integration process, component ordering is not significant and therefore, this issue is not highly relevant.

Property 9 is not satisfied by CPD, *interaction density* and *triangular metrics*. Chidamber and Kermerer metrics [8] do not satisfy property 9 either and they suspect that this property is not suitable at the design level. We believe that for the same reason CPD, CID, CIID, COID and AID metrics do not satisfy property 9. In fact the proposed metrics can be used both at the design level and at the implementation level. For the number of cycle metric, property 9 requires an increase in measurement value if we combine two component assemblies. But this metric relates to the behavior within each assemblies, which is not always affected when components are combined. Finding a new super-component with NC metrics has close relation to the design process and this conclusion is consistent with the results presented by Cho et al. [8].

TABLE III
Summary of Metric Properties

| Metrics | Property | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| CPD | Y | Y | Y | Y | Y | Y | N | Y | N |
| IDM | Y | Y | Y | Y | Y | Y | N | Y | N |
| CM | Y | Y | Y | Y | Y | Y | N | Y | Y |
| TM | Y | Y | Y | Y | Y | Y | N | Y | N |
| NC | Y | Y | Y | Y | Y | Y | N | Y | N |
| ANAC | Y | Y | Y | Y | Y | Y | N | Y | Y |
| ACD | Y | Y | Y | Y | Y | Y | N | Y | Y |
| AACD | Y | Y | Y | Y | Y | Y | N | Y | Y |
| PNAC | Y | Y | Y | Y | Y | Y | N | Y | Y |

An important issue in software development is the overall quality of the software end-product. A set of quality factors for software component assembly can be inferred from the given software metrics. Fig. 4 shows the relationship between some of our metrics using the various quality factors defined in the McCall's quality model [12]. Thus, the CPD metrics provide an indication of the I/O volume, I/O Rate, Storage Efficiency, Error Tolerance and Simplicity[3]. The CID metrics provide an indication of the efficiency of executing an application, the size of I/O interactions, the Error tolerance and simplicity of the application. In other words, the CID metrics are useful towards measuring, the Usability factors, Efficiency, Reliability, Maintainability and Testability. The CM metrics provide an indication of the criticality of I/O Volume and Rate. They also indicate the impacts on Error Tolerance and Simplicity. Thus, the CM metrics impacts on the Usability, Reliability, Maintainability, and Testability quality factors. The NC metric facilitates the identification of super-components, hence it greatly influences the Reusability factor. Other factors that are influenced by the NC metric are Efficiency, Maintainability, Testability, Flexibility and Portability. Typically an application has to be instrumented in order to collect the NC and AC metrics, which leads to the development of good Instrumentation criteria and consequent impact on various Testability factors. The AC metrics also have an effect on the Usability and Efficiency quality factors.

[3] The criterion for simplicity illustrates the nature of the component assembly. For example, a simple component assembly can limit the propagation of faults and ease the inspection and maintenance processes.

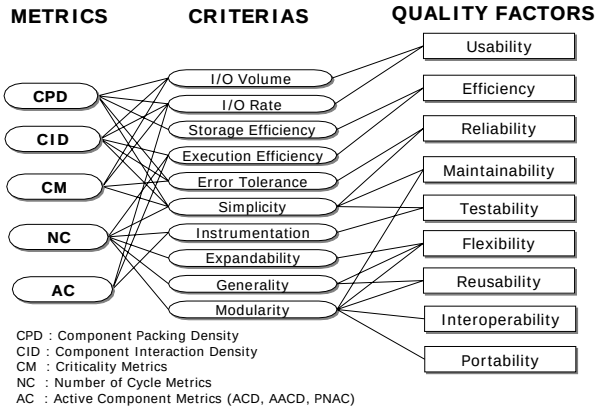


Fig. 4 The Suite of Metrics v Quality Factors

The suite of metrics should be applicable for some stages in software development life cycle, such as design, implementation and testing phase. At design phase, a software designer focuses on development of software structure. Knowing the complexity and criticality of the component structure in advance will help the designer builds better design with better risk analysis. At implementation phase, programmers can use information about complexity and criticality from design stage to accomplish their task carefully at some particular items considered complex and/or critical. At testing phase, software tester can instrument the code and run dynamic metrics suite to get information on how extensive components has been used. Information on the number of cycle can point to a Super-component, which is most useful for software designer.

## VI. Conclusions

This paper proposes a set of static and dynamic metrics. The static metrics characterize the complexity and criticality of component integration and would help a developer in reasoning how complex a system is and locating critical areas in a component assembly. The *Triangular metric* has been generated by combining the complexity and criticality metrics and it is useful in classifying the type of application from a given component assembly. Dynamic metrics use information from the number of cycles in the executed application in order to identify new super-components, which offer better and more functionality. In particular, the a*ctive-component* metrics show components that have high extent of use; a more frequently used component has a higher reusability. This paper also shows that the metrics are based on measurement theory and they have been validated using Weyuker's properties. Most metrics fulfill the Weyuker's property criteria, while a few do not. The impact of our metrics in the context of *McCall's Quality Model* has also been explained in this paper. We therefore contend that these metrics help component-based developers (and integrators) to identify complexity and criticality in an integrated system.

The parsing of metrics from the assembly description makes it possible to visualize various components in the system and their associations; a special tool is being developed to visualize and display the metrics. A project manager can view the component relationships so as to have a better knowledge on their complexity and criticality values. Complex and/or critical components assembly would potentially take longer time to develop and test than a simple one. A better prediction can be established as a consequence of the use of the proposed metrics.

The metrics suite can also be incorporated in a CASE (Computer Aided Software Engineering) tool. Object Constraint Language (OCL) [30] is another related standard, which describes constraints in the analysis and design phases through the Unified Modelling Language (UML) based description. OCL can be embedded on the component model as an added constraint to system building. Adding the constraint in the proposed metrics could yield another method of measuring CBSE software development. The incorporation of metrics into a CASE tool will aid the software project manager in understanding the component based development cycle better. We intend to gather field data for validating the metrics empirically. A further study of complexity and criticality on software component metric would help provide a basis for significant future progress in this area.

## References

[1] A. Arsanjani, Developing and Integrating Enterprise Components and Services, Communication of the ACM, Vol. 45, No. 10, October 2002, pp. 31-34.
[2] F. Berzal, I. Blanco, J. Cubero, N. Marin, Component-based Data Mining Frameworks, Communications of the ACM, Vol. 45, No. 12, December 2002, pp. 97-100.
[3] L. D. Blak, A. Kedia, PPT: A COTS Integration Case Study, Proceeding of 22th International Conference on Software Engineering (ICSE), Orlando, 2002, pp. 41-48.
[4] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer and B.Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.
[5] A.W. Brown, Large-Scale, Component-Based Development, Prentice Hall PTR, 2000.
[6] L. Brownsword, T. Obendorf, C.A. Sledge, Developing New Processes for COTS-Based Systems, IEEE Software, July/August 2000, pp. 48-55.
[7] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-oriented Design, IEEE Transaction on Software Engineering, Vol. 20 No. 6, June 1994, pp. 476-493.
[8] E. S. Cho, M.S. Kim, S.D. Kim, Component Metrics to Measure Component Quality, The 8th Asia-Pacific Software Engineering Conference (APSEC), Macau, 2001, pp. 419-426.
[9] J. J. Dolado, A Validation of the Component-Based Method for Software Size Estimation, IEEE Transactions on Software Engineering, Vol. 26, No. 10, October 2000, pp. 1006-1021.
[10] R.R. Dumke, A.S. Winkler, Managing the Component-Based Software Engineering with Metrics, Proceeding of the 5th International Symposium on Assessment of Software Tools, Pittsburgh, June 1997, pp. 104-110.

*Ref [11-30] can be obtained from the authors.*