Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 3159-0040
Vol. 3 Issue 1, January - 2016

A Proposed Hybrid Component Complexity Metrics For Component Based Software Development

Akwukwuma Veronica Viola[1]
Computer Science Department
University of Benin, Benin City
Edo State.
Nigeria
vvakwukwuma@yahoo.com

Sogbaike Oluwasegun Charles
Computer Engineering Department
Delta State Polytechnic, Otefe-Oghara, Delta State.
Nigeria.
ssogbaike@yahoo.co.nz
+2348166799456

Okene David Ese[3]
Electrical & Electronics Engineering Dept.
Federal University of Petroleum Resources, Effurun
Delta State.
Nigeria.
ese4all2002@yahoo.com

***Abstract*—Component-based software development has become a highly widespread approach for application development instead of developing software from scratch every time; it involves developing software system from existing software components. Software component are black-box in nature where their source codes are not available to the users. Measuring software complexity is an important aspect during software development. Component complexity is also an important issue in component based software development. Component complexity is the effort required to understand and compose a software component to form a software system. From literature, component complexity of black box component is defined by four factors. Software metrics is a system of meas-urement that represents software quality characteristics quantitatively. Conventional complexity metrics cannot measure complexity that exists in software components that are black box. Numerous complexity metrics exits in literature that measure black box component complexity but none of these complexity metrics considered all the four factors that define component complexity. The objective of the work therefore is to propose a hybrid component complexity metrics that considers all the four factors (parameters) that define black box component complexity and apply the proposed complexity metrics to measure component complexity of a university student information system.**

Keywords—Complexity Metrics, Coupling, Cohesion, Component-based Software Development, Software Components

I. INTRODUCTION

An increasing number of software projects miss schedules, exceed budgets, and deliver defective products, and this has turned industry experts to component-based solutions to overcome this software crisis (Vitharana et.al 2003). Component based software development (CBSD) involves composing software system from existing software rather than building from the scratch (Jasmine & Vasantha 2008; Vitharana et.al 2003). This principle embodies an element of "buy, don't build" that shifts the emphasis from programming software to composing software systems (Pressman 2001). CBSD provides many advantages like reduced development time, effort, and increased quality. The advantages of component based development lead to its numerous usages. A standard survey conducted on component based software development from 118 companies around the world indicated that around 53% of the organizations are using component-based approach in its software development (Sharma et.al 2008). CBSD involves understanding the design and implementation of components that is been used to compose the software system. An incomplete understanding of software components has a ripple effect on the entire software system therefore increasing the complexity of the software system (Chhillar & Kajla 2012).

Components are black-box entities that encapsulate services behind well-defined interfaces (Sharma et.al 2007). A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Software component can be deployed independently and is subject to composition by third parties (Chaudhary & Chhillar 2013; Khimta et.al 2008). A generally accepted view of a

software component is that it is a software unit with provided services and required services (Lua & Wang 2007). The provided services are operations performed by the component. The required services are the services needed by the component to produce the provided services. The interface of a component consists of the specifications of its provided and required services

In component based software development, qualifying a software component for composition is a foundation for software system development. In component based software engineering (CBSE) activities, issues related to component integration play a more pivotal role than others, and the process of selecting an ideal component for composition is the challenge of component integrators (Somerville 2011). A component assembler starts with application requirements, searches component repositories for selecting appropriate components, and assembles them by providing the required glue. From a component assembler perspective, being able to assess the complexity of candidate alternative component assemblies is crucial (Kaur and Singh 2013b).

Software complexity means measurement of the resources expended in developing, testing, debugging, maintenance, user training, operation and correction of software products (Latika & Rathore 2011). Complexity in component based software engineering is a measure of the resources expended by a system while integrating with a piece of software to perform a given task and determining the level of component complexity requires the application of software metrics.

Software metrics seek to represent software characteristics and qualities quantitatively and can be used to guide decision as to whether a component can be selected for composition or not. Several metrics has been deployed to measure and qualify software component for composition but theses metrics cannot effectively represent complexity quality characteristics of a software component.

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

From literature, four parameters are used to define software component complexity (Narasimhan and Hendradjaya 2004, Salman 2006, Sharma et.al 2007, Chen et.al 2011, Latika & Rathore 2011, Kaur & Singh 2013a and Kaur & Singh 2013b) and they are:

- Software Component constituent and interaction: A software component is constituted with several methods, variables, interfaces. Software component complexity metrics should consider these parameters as inputs to the metrics.
- Data type of component's value returned or passed and incompatibility: Software component can receive a data type that is passed different from the data type it can handle this result to data-type incompatibility. Data type incompatibility is a factor that causes component complexity
- Coupling: Two objects are coupled if and only if at least one of them acts upon the other. Composing a software system requires some measure of coupling but the degree of coupling translates into the level of dependency of one software component on the other. Coupling is an important parameter when composing software system so as to determine the level dependence of one component on another. This determines the level of complexity of the software component.
- Cohesion: Software component that are viewed as architectural units consist of methods. Cohesion of methods in the software component is important when determining a component complexity

Various types of complexity metrics have been proposed such as Lines of Code (LOC), McCabe's Cyclomatic Complexity Metric, Halstead's Complexity Metric, and Henry's and Kafura's Metric (Kaur and Singh 2013b). These conventional metrics are suitable for software component with available source code; they cannot be applied to black box component where the source codes are not available to the users. It is difficult to use conventional metrics in Component-based Development as these metrics require analysis of source codes.

There are several black box metrics that measure complexity of a black box component proposed in literature such as:

- Complexity metrics for software component (Narasimhan and Hendradjaya 2004)
- Software Component Complexity metrics (Salman 2006)
- Component Complexity (CC) (Sharma et.al 2007),

Complexity metrics for component based software system (Chen et.al 2011)

- Component complexity metrics through integration metrics (Latika & Rathore 2011)
- Component Complexity metrics for black box component (CCMBB) (Kaur and Singh 2013a),
- Interface complexity metrics (Kaur and Singh 2013b)

None of the aforementioned black box software component complexity metrics considered all the four parameters that define software component complexity. Hence there is the need for a hybrid software component complexity metric that takes into cognizance all the four complexity parameters. This is the focus of this research paper. Table 1 gives a summary of the existing complexity metrics and the complexity parameters considered.

Table 1. Complexity Metrics and the Complexity Parameters Considered

	Component Complexity Parameters					
Metrics	Component constituent and interactions(links)	Data type of return value, argument and no of incompatibility	Coupling	Cohesion	System level metrics	Component level metrics
Software Component Complexity metrics (Salman 2006)	Yes	No	No	No	Yes	No
Component Complexity (CC) (Sharma et.al 2007)	Yes	Yes but incompatibility was not considered	No	No	No	Yes
Complexity metrics for software component (Narasimhan and Hendradjaya 2004)	Yes	No	No	No	Yes	No
Complexity metrics for component based software system (Chen et.al 2011)	Yes	No	Yes	Yes	No	Yes
Component complexity metrics through integration metrics (Latika & Rathore 2011)	Yes	No	No	No	Yes	No
Interface complexity metrics (Kaur and Singh 2013) b	Yes	Yes	No	No	No	Yes
Component Complexity metrics for black box component (CCMBB) (Kaur and Singh 2013) a	Yes	Yes	Yes	No	No	Yes

II. METHODOLOGY FOR THE PROPOSED HYBRID COMPONENT COMPLEXITY METRICS

The hybrid component complexity metrics for software component is a combination of existing compo-

nent complexity metrics. Sharma et.al (2007) complexity metrics considered component complexity parameters which are number of component constituent and interactions, Data type of return value, and parameter but did not consider the incompatibility that exist when component of different data types configuration interact. In composing the proposed hybrid component complexity metrics, the Interface complexity metrics (Kaur and Singh 2013b) (eq, 1) was considered for the first two complexity parameters in table 1, this is because it took cognizance of the incompatibility factor in addition to Data type of return value, the number of component constituent and interactions. For the last two component complexity parameter (Coupling and cohesion) of table 1, Complexity metrics for component based software system (Chen et.al 2011) (eq. 6&7) was considered, although Component Complexity metrics for black box component (CCMBB) (Kaur and Singh 2013a) measured coupling also but did not consider cohesion and combined coupling with number of incompatibility that exist when components of different data types interact.

Mathematical Illustration of component complexity metrics of Kaur and Singh (2013b) and Chen et.al (2011)

The interface complexity metrics (Kaur and Singh 2013b)

IC = $\sum_{i=1}^{m} IMCM_i$ ..... (1)

Interface Method Complexity Metric (IMCM)
IMCM = Wr + PCM (M) + Number of parameter incompatibilities ..... (2), Where Wr is the weight assigned to return type, PCM (M) is the Parameter Complexity Metric for method.

PCM (M) = $\sum_{i=1}^{n} Wp(Pi)$ ..... (3)

Where Wp(Pi) is the weight assigned to the ith parameter of the method on the basis of its data type , n represents the number of parameters in a method in table 2. The methods having no return value and no parameters have been considered as simple methods and their weight value has been assumed .025.

Table 2 : Represents weight values assigned to different categories of data types for parameters and return values (Kaur and Singh 2013)

Parameters/ Return values →	Very Simple	Simple	Medium	Complex	Very Complex
Weight assigned ↓	0.10	0.20	0.30	0.40	0.50

Source: Component Complexity Metrics (Chen et.al, 2011)

Sole component complexity metric (SCCM) is combination of the above three component metrics with different weights for each metric.

$$SCCM_j = \alpha * MV_j + \beta * COM_j + \gamma * AIM_j$$

..... (4)

Actual Interactions Metric (AIM)

$AIM_j = \frac{II + OI}{II_{jmax} + OII_{jmax}}$ ..... (5)

II_{jmax} and OI_{jmax} are maximum numbers of input and output interactions in a component j. (OI) and (II) are the

available number of incoming and outgoing interactions

Coupling Metrics

$MV_j = \frac{\bigcup_{1 \le i \le m \cap i \ne j} MV_{ji}}{M_j + V_j}$ ..... (6)

where M_j and V_j is the sets of methods and instance variables of component C_j. $MV_{j,i}$ is the set of methods and instance variables in component C_i invoked by component C_j. MV_j, the set of all methods and instance variables in other components, $1 \le i \le m$ and $i \ne j$, that are invoked by component C_j

Cohesion Metrics

$COM_j(C) = \frac{E_j(C)}{V_j(C) * M_j(C)}$ ..... (7)

Set of method members $M_j(C) \equiv \{ m_{j1}, m_{j2}, \ldots m_{jm}\}$ and a set of instance variables $V_j(C) \equiv \{v_{j1}, v_{j2}, \ldots v_{jn}\}$. $E_j(C)$ is the set of pairs (v_j, m_j) for each instance variable v in V(C) that is used by method m in $M_j(C)$

The hybrid component complexity metrics is defined as follows

$$\mathbf{HCCM} = \mathbf{IC} + \mathbf{MVj} + \mathbf{COMj} \quad \text{..... (8)}$$

Where IC is the interface complexity metrics of (Kaur and Singh 2013b), MV_j the coupling metrics (Chen et.al 2011); COM_j is the Cohesion Metrics (Chen et.al 2011). Where j is the component being measured

III. IMPLEMENTATION

The proposed hybrid complexity metrics was implemented by applying to a component based system which is a university student information system. The university student information system is a software system from which the students of different departments can receive information about their marks details, fee details and course details. This system has been developed by integrating the components. Figure 1 and figure 2 are the class diagram and activity diagram respectively of the university student information system.

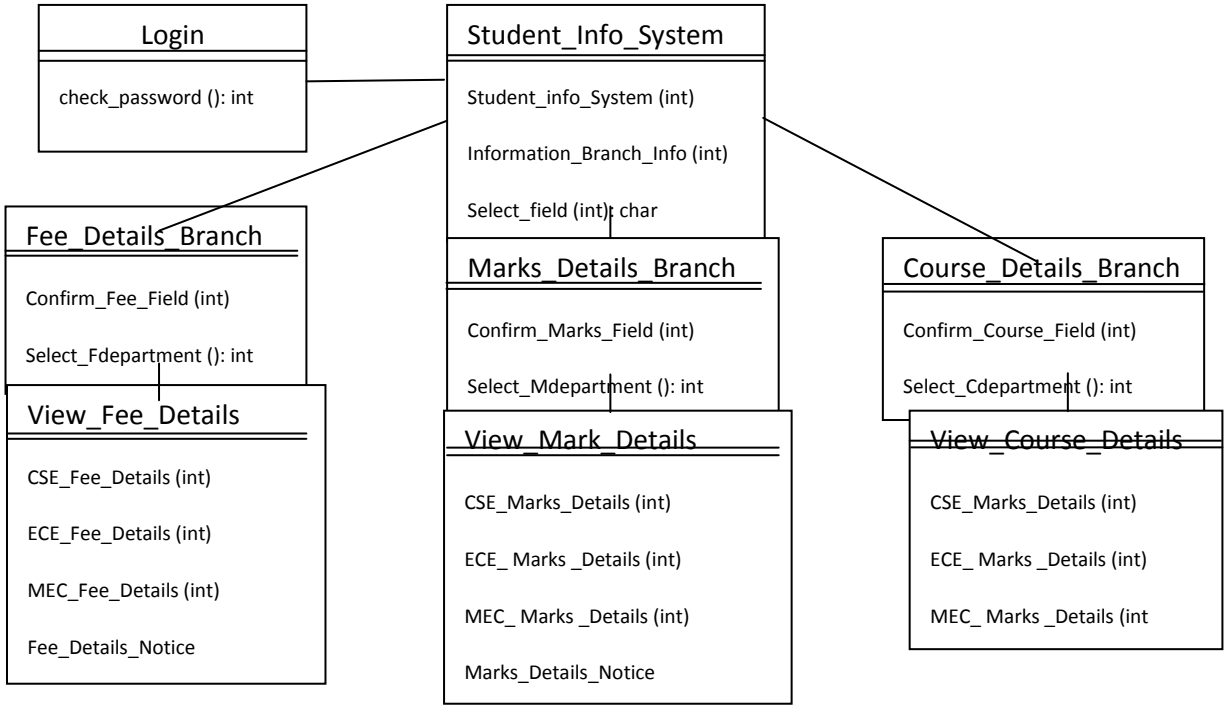

Figure 1 Class Diagram of University Student Information System

TOTAL NO OF LINKS = 10
TOTAL NO OF INTERFACE= 20
TOTAL NO OF COMPONENT = 8
TOTAL NO OF METHODS = 21

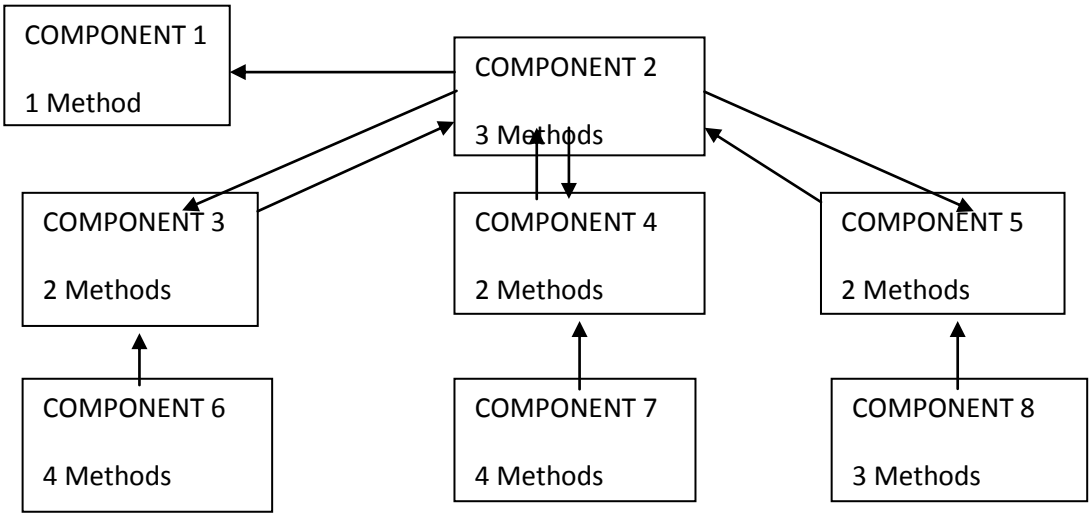

Figure 2 Activity Diagram of University Student Information System

The system is composed of eight (8) components as shown in the class diagram fig 1; and the operations of each of the components are as follow:

Login Component: This component checks the password entered by the user in order to authenticate the user. If the password is correct then it will return value 1 otherwise it will return value 0. The return value will be passed to the second and third method of component named Student_Info_System

Student_Info_System Component: This component provides the information about the system's working, information about the various information branches .The third method of this component provides the options to the user to select the branch from which the user want to get information. The second and third method will perform their actions only if the password is correct. This component passes F, M, C if user wants to get information about fee details, marks details and course details respectively, for the confirmation of the selected branch. But the components Fee_Details_Branch, Marks_Details_Branch and Course_Details

and Course_Details_Branch accept the integer value for the confirmation of the selected branch. Thus when this component is used three parameter incompatibilities are caused which will make it difficult to use the method. It will also reduce the composability

Fee_Details_Branch Component: The first method of this component confirms the selected branch from which the users want to get information. If user has selected fee details branch to get information then it will return 1 other wise 0. Second method of this component displays list of departments from which the user can select any department for which the user want to check fee_details. This component will pass 1, 2, 3, for CSE, ECE, and MEC departments respectively, to the View_Fee_Details Component. When this component is used it will create one parameter incompatibility because component Student_Info_ System passes the char parameter but the Fee_Details_Branch component takes the integer parameter to confirm the selected branch.

Mark_Details_Branch Component: The first method of this component confirms the selected branch from which the users want to get information. If user has selected mark details branch to get information then it will return 1 other wise 0. Second method of this component displays list of departments from which the user can select any department for which the user want to check mark_details. This component will pass 1, 2, 3, for CSE, ECE, and MEC departments respectively, to the View_Mark_Details Component. When this component is used it will create one parameter incompatibility because component Student_Info_ System passes the char parameter but the Mark_Details_Branch component takes the integer parameter to confirm the selected branch.

Course_Details_Branch Component: The first method of this component confirms the selected branch from which the users want to get information. If user has selected course details branch to get information then it will return 1 other wise 0. Second method of this component displays list of departments from which the user can select any department for which the user want to check course_details. This component will pass 1, 2, 3, for CSE, ECE, and MEC departments respectively, to the View_Course_Details Component. When this component is used it will create one parameter incompatibility because component Student_Info_System passes the char parame-

ter but the Course_Details_Branch component takes the integer parameter to confirm the selected branch.

View_Fee_Details Component: 1, 2, 3 values are passed to this component when the user want to view fee details of CSE, ECE and MEC departments respectively. The Fee_Details_Branch component passes the integer value for the selected department to the View_Fee_Details component and this component also accept the integer value to confirm the selected department. Because the passed parameter and received parameter data types are same so this component does not create any incompatibility problem.

View_Mark _Details Component: 1, 2, 3 values are passed to this component when the user want to view mark details of CSE, ECE and MEC departments respectively. The Mark_Details_Branch component passes the integer value for the selected department to the View_Mark_Details component and this component also accept the integer value to confirm the selected department. Because the passed parameter and received parameter data types are same so this component does not create any incompatibility problem.

View_Course_Details Component: 1, 2, 3 values are passed to this component when the user want to view course details of CSE, ECE and MEC departments respectively. The Course_Details_Branch component passes the integer value for the selected department to the View_Course_Details component and this component also accept the integer value to confirm the selected department. Because the passed parameter and received parameter data types are same so this component does not create any incompatibility problem.

Table 3 is summary of the operations of the students' information system from the class diagram of figure 1.

Table 3 Activity Diagram of University Student Information System

Component	J		1	2	3	4	5	6	7	8	II_{MAX}	OI_{MAX}	$E_J(C)$
I	M_j	V_j	MV_{ji}	MV_{ji}	MV_{ji}	MV_{ji}	MV_{ji}	MV_{ji}	MV_{ji}	MV_{ji}			
1	1	1		0,0							-	1	1
2	3	3	1,1		2,2	2,2	2,2				4	3	2
3	2	2		2,2				1,1			1	2	2
4	2	2		2,2					1,1		1	2	2
5	2	2		2,2						1,1	1	2	2
6	4	3			1,1						1	-	3
7	4	3				1,1					1	-	3
8	3	3					1,1				1	-	3

Equation 1, 2, and 3 was used to compute complexity parameters one and two in table 1 which in the hybrid metrics is represented as IC and Table 3 was used as input parameters to compute the coupling (MV_j) and cohesion (COM_j (C)) aspects of the proposed hybrid metrics and the obtained component complexity result for the student university information system is as shown in table 4.

Table 4 Complexity Result of Hybrid Complexity Metrics for Black Box Component

Component	IC	COM_j (C)	MV_j	HCCM
1	0.6	1	0	1.6
2	3.925	0.33	2.33	6.585
3	1.70	0.50	1.50	3.70
4	1.70	0.50	1.50	3.70
5	1.70	0.50	1.50	3.70
6	0.325	0.25	0.28	0.855
7	0.325	0.25	0.28	0.855
8	0.30	0.35	0.33	0.98

IV. VALIDATION OF HYBRID COMPONENT COMPLEXITY METRICS (HCCM)

Another important aspect in software metrics is the validation of a metrics. A software complexity metric is valid if it succeeds in satisfying defined properties. Several researchers have tried to describe a set of properties that a good software complexity metric must satisfy but none has been totally accepted or totally rejected by the software development community (Salman 2006; Sharma et.al 2007). On this premise, six validation properties were proposed in Salman (2006) and these properties can be applied to all types of complexity metrics.

Property 1: Non-negativity: A complexity metric value cannot be a negative number. From table 4, the (HCCM) all the metric values are non-negative this satisfies the property of non-negativity however the zero in component one coupling result (MV_j) mean the component requires a low coupling effort.

Property 2: A software complexity metric must provide a scale of values. Comparison between different alternatives must be possible. For any two software artefacts (component) it must be possible to compare and then make managerial decisions according to the metrics values. Our proposed metrics result in table 4 provided a scale of value indicating that component two is of the highest complexity based on the value and thus managerial decision will require more time, effort and cost to compose the component.

Property 3: The complexity of a single software unit S composed of two software components cannot be less than the sum of the complexities of the individual components. Complexity(S) >= Complexity (C1) + Complexity (C2).

The complexity of a component based software system is a function of the complexities of individual components that make it up, and an added complexity will appear as a result of new interactions that may exist between the components. In the best case, when a system is composed of two components and no new added interactions between the components are available, the system's complexity will be equal to the sum of the individual component complexities.

We use our proposed metrics to obtain the complexity of the entire software system by adding all the component complexity value of each individual component. The sum of all the complexity value of each component equals the complexity value of the component.

Property 4: If a component C is decomposed into two or more components C_1, C_2... C_n then the sum of complexities of the resulting components is no more than the overall complexity of the original component.

Complexity (C_1) + Complexity (C_2) + … + Complexity (C_n) <= Complexity(C)

There is usually an added complexity whenever two components are composed. The new complexity usually results from the interactions between these components. So, when the component is decomposed these links will disappear and only the component's intrinsic complexity will remain.

The hybrid component complexity metrics (HCCM) result in table 4 is a metric result for components of the software system and summing these results yields the complexity of the entire system and the complexity value of Student University information system cannot be less than the sum of the entire individual component it is composed of. This satisfies the fourth property.

Property 5: The complexity value of one component does not have a direct relationship to its methods, i.e. for any two components C1 and C2, if Complexity (C1) > Complexity (C2) then it is not necessary that C1 has more methods than C2.

From table 4, Complexity value of component 8 (0.98) is considerably lower than that of component one (1.6) but component eight contains more methods compared with component one which has only one see figure 2. This satisfies the fifth property.

Property 6: The complexity value is directly influenced by a component structure or organisation.

From the university student information system complexity result, component two has the highest complexity value because of its organisation satisfying the sixth property.

V. CONCLUSION

To produce a software system with minimal development cost, effort and within the shortest possible time, component based software development is needed. With component based software engineering, software systems are developed by composing existing software components.

Qualifying a software component for composition requires the use of software metrics and the most important software quality is complexity which is defined by four parameters. Several metrics have been proposed to measure black box component complexity but these metrics did not consider all the four parameters that define complexity.

We then proposed a hybrid complexity metrics that considered all the four parameters that defines component complexity and implemented the metrics on a university student information system. We also validated the hybrid complexity metrics by verifying that the metrics satisfies some six properties.

To effectively measure component complexity, the hybrid complexity metrics is ideal to represent the level complexity of black box software component.

VI. REFERENCES

[1] Chen J, Wang H, Zhou Y, and Bruda S (2011): Complexity Metrics for Component-based Software Systems. International Journal of Digital Content Technology and its Applications, Volume 5.number 3 pp 235 - 244

[2] Jasmine K.S, and Vasantha. R. (2008) Cost Estimation Model for Reuse Based Software Products Proceedings of the International Multi-Conference of Engineers and Computer Scientists (IMECS) Hong Kong Vol. I

[3] Kaur N. and Singh A. (2013b): Component Complexity Metrics: A Survey. International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 6 pp 1056 -1061

[4] Kaur N. and Singh A. (2013a) : A Metric for Accessing Black Box Component Reusability. International Journal of Scientific & Engineering Research, Volume 4, Issue 7, pp 1114 – 1121

[5] Khimta S, Sandhu P. S, and Brar A. S (2008): A Complexity Measure for JavaBean based Software Components: World Academy of Science, Engineering and Technology pp 449-452

[6] Latika M. and Rathore V.S (2011): Software Component Complexity Measurement through Proposed Integration Metrics: Journal of Global Research in Computer Science vol 2 no. 6 pp 13 - 15

[8] Lau K. and Wang Z. (2007): Software Component Models; IEEE Transactions on Software Engineering, Vol 33, no 10 pp 709 – 724

[9] Narasimhan, V. L., & Hendradjaya, B. (2004): A new suite of metrics for the integration of software components. In Proceedings of the First International Workshop on Object Systems and Software Architectures (WOSSA'2004).

[10] Pressman R (2001) Software Engineering: A Practitioner's Approach: Roger Pressman fifth edition, Mcgraw Page 1- 888, Toronto

[11] Salman N. (2006): Complexity Metrics as predictors of maintainability and integrability of software components. Journal of Arts and Sciences pp 39 -50

[12] Sharma A, Kumar R. and Grover P.S (2007): Empirical Evaluation and Critical Review of Complexity Metrics for Software Components. Proceedings of the 6th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems pp 24-29, Greece

[13] Sharma A, Kumar R. and Grover P.S (2008): Managing Component-Based Systems with Reusable Components. International Journal of Computer Science and Security, Volume 1: Issue (2) pp 52 – 57

[14] Somerville I (2011): software engineering Addison Wesley publication ninth editions

[15] Vitharana P, Zahedi F.M, and Jain H. (2003): Design, Retrieval and Assembly in Component Based Software Development Communications of the ACM, Vol. 46, No. 11 pp 97 – 102

[7] Chhillar R.S and Kajla P. (2012): New component composition metrics for component based software development international journal of computer applications volume 60 no 15.pp 17-20