EVALUATION OF COMPLEXITY FOR COMPONENTS IN COMPONENT BASED SOFTWARE ENGINEERING

Puneet Goswami*

Pradeep Kumar**

Keshav Nand***

ABSTRACT

Component Based Software Engineering (CBSE) is focused on assembling existing components to build a software system, with a potential benefit of delivering quality systems by using quality components.. In the present paper, complexity metric is proposed for component based on the different constituents of the component like, Component Dynamic Complexity, methods and properties with different weights assigned to them. This metric is applied on various JavaBeans components for empirical evaluation. Further, correlation study has been conducted for this metric with quality characteristics, like, customizability, readability and Throughput rate. The study conducted shows negative correlation between them, which confirms the assumption that high complexity of the components leads to the high cost of maintainability.

*Associate Professor, Department of Computer Science & Engineering, Galaxy Global Group of Institutions, Ambala.

**Associate Professor, Department of Computer Science & Engineering, G.J.U.S&T, Hisar.

***Lecturer, SNRL Girls College, Lohar Majra.

1. INTRODUCTION

Components are black box in nature. The source code of these components is not available.. Complex interfaces will lead to the high efforts for understanding and customizing the components. Therefore for better reusability, interface complexity should be as low as possible. We use the methodology for measuring their interface complexity of software components. This Paper proposes an interface complexity metric for software components, which is based on complexity involved in the interface methods and properties, Component Dynamic Complexity (CDC) used in the interface. These interface methods may have parameters and return values, which are the only source of information available to us for that interface method. Depending on the nature and the number of these parameters and return values and properties, weight values may be assigned to them, which can be used to measure the complexity of the target interface method. Finally, the sum of complexities of these methods will give the overall interface complexity of the target component.

1.1 Review Literature:

It consists of a literature survey of various metrics for CBS. It covers complexity, reusability, customizability, maintainability, and other aspects of these systems. Some of the metrics proposed in the literature are derived from that of object-oriented systems with no or minor changes, others are exclusively specific for CBS.

1.2 Complexity Metrics for Component-based systems

Some of the metrics proposed in the literature are derived from that of object-oriented systems with no or minor changes, others are exclusively specific for CBS.

Cho *et al.* (2001) proposed a suite of complexity metrics for software components. The approach considers the classes and their methods of each component and captures the dynamic complexity, based on analysis of source code of the component. However, it cannot be used to measure the complexity for black-box components, as the source code of these components is not available.

Gill and Grover (2004) proposed interface complexity metric, based on interface signatures, constraints on the interfaces and the packaging for different context of use. For each of these aspects, a definition is also proposed. However, work still lacks of any empirical evaluation and validation of the proposed metric.

Sharma *et al*. (2008a) proposed interface complexity metric for software components by considering interface methods and their associated properties, argument types and return types. Authors evaluated this metric on several Java Beans components and finally validated it

against execution time, readability and customizability. Results concluded that complex components take much time to execute and these are very difficult to maintain. [1] However, this works lacks the approach of assigning weight values to the methods and properties for proposed metric. This metric also lack the approach to calculate the dynamic complexity of components. Moreover author for performance, time taken by these components (with default values of parameters) to execute in seconds is measured which is based on assumption.

2. PROPOSED COMPLEXITY METRIC

We extended the approaches described in (Rotaru *et al.*, 2005; Gill and Grover, 2004; Boxall and Araban, 2004) Sharma *et al.* (2008a). Proposed metric uses the behaviour of the component through its interface methods and properties, which are available even without going into the internals of the component. For the proposed metric, we consider the events and their listeners similar to the methods. We propose that the interface complexity metrics for the component will be due to the complexities involved in its interface methods and properties described above and define Interface Complexity Metric (*ICM*) for Component *C* as:

$$\mathbf{ICM(C) = a\sum_{i=1}^{n} CIM_i + b\sum_{j=1}^{m} CP_j + \sum_{k=1}^{p} CDC_k}$$

CIM_i= ith Component Interface Method, CP_j= jth Component Property,

CDC_k= kth Component Dynamic Complexity

where CIM_i is the complexity of i^{th} interface method and CP_j is the complexity of j^{th} property. *A* and *b* are weight values for methods and properties respectively, as complexity of a interface method may have different weight value than the complexity of a property.

The third approach used in order to measure the Complexity of a component is CDC. CDC focuses on how many message passing is occurred in a component. CDC is a metric that measures the complexity of internal message passing in a component with a dynamic view. Therefore, the dynamic complexity of each component is calculated by counting messages passed between classes contained in a component. We define the CDC as following formula: $\sum_{k=1}^{p} CDC_k$, the kth Component Dynamic Complexity of each interface method.

Complexity of interface methods may be me assured based on its return type and arguments passed to it. We assign different weight values to these methods based on the nature (data type) of arguments/return values, used in the method. Our assumption in assigning weight values is that a method having no argument (e.g. constructor) may be considered as simplest method and we assign the weight value to these methods 0.025. All other interface methods

are assigned weight values depending on the type and total number of arguments and return types.

3. EMPIRICAL EVALUATION OF PROPOSED COMPLEXITY METRIC USING WEIGHTED ASSIGNMENT TECHNIQUE:

To obtain the values of the proposed complexity metric, an experiment is conducted on various JavaBeans components (these are available at various websites ww.elegantjbeans.com, www.java.sun.com, www.java2s.com). These JavaBeans components vary from very simple and small to complex and large. These have different number of attributes and methods. We assign different weight values to these methods based on the data type of arguments or return values, used in the method.

Data types are categorized as:

- Very Simple include integer, double, Boolean, float type.
- Simple include structured data type.
- Medium include Class type and Object type.
- Complex includes pointers, built-in data type.
- And highly complex includes User-defined data types.

Method having no argument (e.g. constructor) may be considered as simplest method and we assign the weight value to these methods 0.025. All other interface methods are assigned weight values depending on the type and total number of arguments and return types. The following table shows these weight values:

Data Type No. ↓ →	VerySimple	Simple	Medium	Complex	Highly Complex
1-3	0.025	0.05	0.15	0.17	0.25
4-6	0.050	0.10	0.30	0.34	0.50
7-9	0.075	0.15	0.45	0.51	0.75
>=10	0.98	0.20	0.60	0.68	1.00

Table1. Weight values for Interface Methods

Case-I:= In the openAccount the number of method is 20 and the number of properties is 30. Suppose the method are integer type and the properties are also integer type then the both interface and properties are very low. Then the weight value for a and b from the table is 0.050 and 0.050.

Now calculate the component dynamic complexity for the openAccount…

As depicted in Figure**,** there are several message flows among interface method contained in a component such as Open Account component. It is difficult to measure the complexity of component with only class diagrams or component diagrams. Therefore, it is enable to measure the dynamic complexity of each component with interaction diagrams. Applied CDC into this example diagram, the measurement result is given by:

OpenAccount() = l+1+1+2+1+3+4=13.

As shown in Figure1.1, there are seven message flows in the 'Open Account' interface method. Therefore, we obtained the value of OpenAccount by applying the formula $\sum_{k=1}^{p} CDC_k$

Here the values obtained by formula **n+1** as even the argument for an interface method is not present then default value 0+1 where n =0 represents there is no argument present in message and by default value is 1.

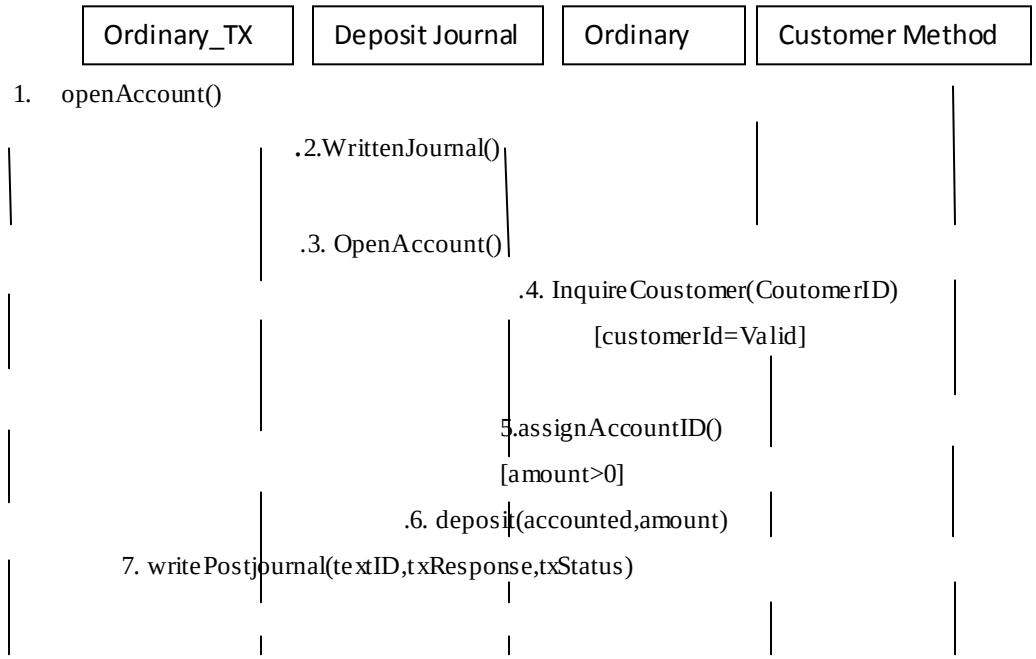


Fig 1.1 OpenAccount Sequence Diagram

In the above sequence diagram the CDC for the openAccount is 1+1+1+2+1+3+4 = 13

So the Interface component complexity for the openAccount is

ICM(C) = 0.050*20 + 0.050*30 +13

= 1+1.5 +13

= 15.5

Similarly, we calculated the CDC for remaining components and values are shown in Table1.1

JavaBeans	Interface Method	"a"	Properties	"b"	CDC	Interface Complexity
OpenAccount	20	0.050	30	0.050	13	15.5
CloseAccount	5	0.34	8	0.075	5	7.3
InquireAccount	50	0.45	75	0.025	4	28.38
InquireTransactionHistory	25	0.51	38	0.20	8	28.35
InquireCustomerAccount	40	0.75	60	0.51	30	97.6
Deposit	11	0.050	17	0.050	10	11.35
WithDraw	12	0.98	18	0.075	9	22.11
Transfer	44	0.10	66	0.60	12	56
InquireBookKeeping	30	0.05	45	0.34	18	34.8
AssignAccountId	22	1	33	0.17	20	47.61
CallInterest	48	0.25	62	0.75	15	73.5

Table 1.1 Complexity Metrics Values

Table 1.1 shows that the Java Beans considered for the experimentation vary from very simple (only 5 methods and 8 properties) to highly complex (around50 methods and 75 properties). The complexity of these components varies from 7.3 to 97.6.

Evaluation of Customizability : Component customization is the process that involves 1) modifying the component for the specific requirement; 2) doing necessary changes to run the component on special platform; 3) upgrading the specific component to get a better performance or a higher quality. The objectives of component customization are to make necessary changes for a developed component so that it can be used in a specific environment or cooperate with other components well.. The following formula is used to evaluate this metric:

$$\text{Customizability} = \frac{\text{Number of Set Methods}}{\text{Total Number of Properties}}$$

JavaBeans	Set Methods	Properties	Customizability
OpenAccount	12	13	0.92
CloseAccount	3	3	1
InquireAccount	9	12	0.75
InquireTransactionHistory	3	4	0.75
InquireCustomerAccount	1	11	0.09

Deposit	17	17	1
WithDraw	7	8	0.88
Transfer	4	6	0.67
InquireBookKeeping	4	5	0.80
AssignAccountId	7	10	0.7
CallInterest	1	10	0.1

Table1.2 Values of Customizability for various Java Beans Components

Evaluation of Readability:

Readability can be measured by getting the observable properties from the component. Readability will help an application developer to understand the component. If a component is understandable, it will be easier to use it and maintain it.. The following formula is used to evaluate this metric:

$$\text{Readability} = \frac{\text{Number of Get Methods}}{\text{Total Number of Properties}}$$

JavaBeans	Get Methods	Properties	Readibility
OpenAccount	4	5	0.8
CloseAccount	8	8	1
InquireAccount	12	17	0.7
InquireTransactionHistory	7	10	0.7
InquireCustomerAccount	1	10	0.1
Deposit	12	13	0.92
WithDraw	7	8	0.88
Transfer	5	10	0.5
InquireBookKeeping	8	10	0.8
AssignAccountId	4	6	0.67
CallInterest	3	10	0.3

Table 1.3 Values of Readibility for various Java Beans Components

Karl Pearson's Correlation Coefficient used to get the correlation coefficient between interface complexity and quality characteristics, Customizability and Readability.

Validation of the Proposed Metric:

Data Analysis is done by using **Minitab 16 English software** and Graphical representation on impact of Complexity on customizability, Readability & Transaction Throughput Rate is shown using the same software.

Correlation analysis is performed for the proposed metric with several other quality characteristics, like, performance, customizability and readability on same Java Bean components used for proposed complexity metric.

Calculate Correlation Coefficient between C*omplexity* and *customizability* using Karl Pearson Method. X is Interface Complexity and Y is Customizability.

Karl Pearson's Formula is

$$r = \frac{\sum xy}{\sqrt{\sum x^2} * \sqrt{\sum y^2}}$$

Where $x = X - \overline{X}$ and $\overline{X} = \frac{\sum X}{N}$

$y = Y - \overline{Y}$ and $\overline{Y} = \frac{\sum Y}{N}$

Correlation between complexity and customizability

$$r = \frac{-84.478}{1.004 * 88.511}$$

$$= -0.950$$

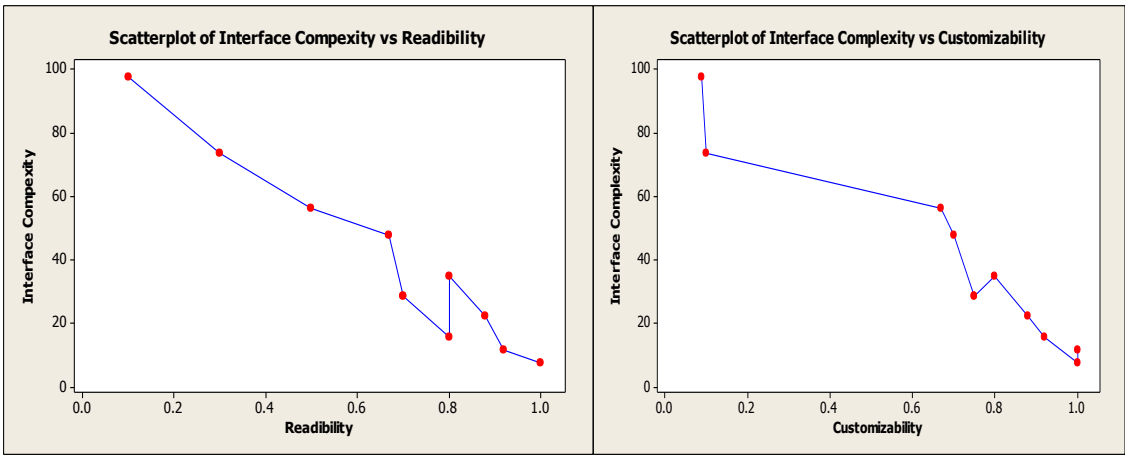
The graph clearly indicates that with the increase in Interface Complexity of Components the customizability and Readability of component decreases.

Now, Calculate Correlation Coefficient between complexity and Readability using Karl Pearson Method. X is Interface Complexity and Y is Readability Karl Pearson's Formula is

Correlation between Interface Complexity and Readability

$$r = \frac{-74.2165}{76.233}$$

$$= -0.974$$



As there are negative correlation between Complexity and Customizability, and Complexity and Readability, which confirms that highly complex components are hard to customize and to understand which leads towards poor reusability and maintainability. These correlations Coefficients and their interpretation validate the proposed complexity metric for components.

Component Transaction Throughput Metrics:

The *transaction throughput rate* of a component *Ck* is the ratio of the total number of successfully processed transaction requests to the total number of received transaction requests during a given performance evaluation period *Tp*. The formal metric is given here:

Throughput Rate (C_k , T_p , TR) = m/n

Where *m* is the total number of successfully processed *TR* transaction requests during *Tp*, and *n* is the total number of received *TR* transaction requests during *Tp*. When the throughput rate reaches 1, component C_k has the highest successful rate in processing transaction requests in *TR* category.

JavaBeans	m	n	Transaction Throughput rate of Component
OpenAccount	7	8	0.88
CloseAccount	9	9	1.00
InquireAccount	3	4	0.75
InquireTransactionHistory	3	4	0.75
InquireCustomerAccount	1	10	0.10
Deposit	8	8	1.00
WithDraw	7	8	0.88
Transfer	3	6	0.50

InquireBookKeeping	12	18	0.66
AssignAccountId	12	18	0.66
CallInterest	1	5	0.20

Table 1.4 Values of Throughput Rate of components for various Java Beans

Here, in this table we have evaluated the Transaction Throughput Rate of component on Set of Java Beans where *m* is the total number of successfully processed *TR* transaction requests during *Tp*, and *n* is the total number of received *TR* transaction requests during *Tp*.

Now, Correlation between the Interface complexity of Component & Transaction Throughput rate of component is calculated. The results are shown in Table1.8

Characteristics	Pearson Correlation Coefficient
Complexity vs. Transaction Throughput rate	-0.984
Complexity vs. Customizability	-0.950
Complexity vs. Readability	-0.974

Table1.5 Correlation Coefficients among proposed and other metrics

Table 1.8 shows that there is a strong negative correlation between complexity and Transaction throughput rate of components which clearly indicates that complexity of components increases than throughput rate decreases .We know that as throughput rate of component is inversely proportional to Execution Time therefore Complex components take more time to execute, which is self evident and is proved by our proposed metric. Also, there are negative correlation between complexity and customizability and complexity and readability, which confirms that highly complex components are hard to customize and to understand which leads towards poor reusability and maintainability. These correlation coefficients and their interpretation validate the proposed complexity metric for components.

4. CONCLUSION

The paper discusses various complexity metrics proposed by researchers especially for component -based systems. 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, the proposed metrics provide more accurate results than the interface metrics proposed by Sharma *et al.* (2008a) the work lacks the approach of assigning weight values to the methods and properties for metric, metric also lack the approach to calculate the dynamic complexity of components(CDC).The metrics

proposed in this paper also conducts validation through other metrics, namely, customizability and readability by using Karl Pearson's Coefficient method and verifies the facts that complex components take more time to execute and are hard to maintain and reuse.

REFERENCES

1. Arun Sharma, Rajesh Kumar, P. S. Grover, Evaluation of Complexity for Software Components, International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol. 19, Issue 5, November 2008, pp: 919-931.
2. Bertoa, M., Troya, J. M., Vallecillo, A., 2006. Measuring the Usability of Software Components, Journal of Sys tems and Software, Vol. 79, Issue 3, pp: 427-439.
3. Boxall, M. A. S., Araban, S., 2004. Interface Metrics for Reusability Analysis of Components, Proceedings of. Australian So ftware Engineering Conference (ASWEC'2004), Melbourne, Australia, pp: 40 -46.
4. Cho ES., Min Sun Kim MS., Kim SD. Component metrics to measure component quality. Software Engineering Conference on, 2001. APSEC 2001. Eighth Asia-pacific software engineering conference. pages: 419- 426, 2001.
5. George T. Heineman and William T. Councill "Component Based Software Engineering"Publisher: Addison Wesley Longman, 2001.
6. Gill, N. S., 2003. Reusability Issues in Component-based Development, ACM SIGSOFT Software Engineering Notes, Vol. 28, Issue 4, pp: 1-5.
7. Gill, N. S., Grover, P. S., 2003. Component-Based Measurement: Few Useful Guidelines, ACM SIGSOFT Software Engineering Notes, Vol. 28, Issue 6, pp: 1-4.
8. Gill, N. S., Grover, P. S., 2004. Few Important Considerations for Deriving Interface Complexity Metric for Component-Based Systems, ACM SIGSOFT Software Engineering Notes, Vol. 29 Issue 2, pp: 1-6.
9. Kharb, L., Singh, R., 2008. Complexity Metrics for Component -Oriented Software Systems, ACM SIGSOFT Software Engineering Notes, Vol. 33, Issue 2, pp: 1-3.
10. Nael, S., 2006. Complexity Metrics AS Predictors of Maintainability and Integrability of Software components, Journal of Arts and Sciences, Issue 5, pp: 39-50.
11. Puneet Goswami, P. K. Bhatia and Vijender Hooda, "Effort estimation in Component Based Software Engineering", International Journal of Information Technology and Knowledge Management, Volume 2, issue 2, pp. 437-440, 2009.