CHAPTER 4

QUANTIFICATION AND EVALUATION OF INTERFACE COMPLEXITY IN COTS BASED SYSTEMS

4.1 Introduction
4.2 Scope and Goal of the Chapter
4.3 Related Work
  4.3.1 Complexity Metrics for Basic Software System
  4.3.2 Complexity Metrics for Component Based System
4.4 Proposed Complexity Metrics: Aim and Approach
4.5 Identification and Classification of Factors for Interface Complexity at Integration level
4.6 Assign Weight to each factor and subfactors using AHP
4.7 The Proposed Metric
  4.7.1 The OCI metric
  4.7.2 The ICI metric
4.8 Theoretical evaluation of proposed Metrics using Weyunker’s Properties
4.9 Experimentation on Proposed Metrics
4.10 Result Analysis
4.11 Summary

The major findings of the Chapter have been published in International Journal of Computer Applications vol. 73, Issue 02, p.p.:21-28, July 2013. Published by Foundation of Computer Science, New York, USA.
4.1 INTRODUCTION

A COTS-Based System (CBS) is an assembling approach to build up the application, with attention on integrating each COTS component. In CBS, as the source code of every COTS component is not provided with them, we have to specify some important properties for each COTS component like constraints related with its use, customizability properties and interactions with other components. Customization can be carried out via its interface properties and operations. The different COTS components might be built up having different platforms and using various programming languages. It might execute on distributed machines. Several components may be developed in-house, while some others might be developed by a third party and simply known as commercial off-the-shelf components (COTS). A key to the success in the development of COTS-Based System is integrators or developers capability to use these COTS components which are often purchased from and developed by the third parties. Hence, the integration and selection of a COTS component is main importance for helping the developer to select and integrate the finest COTS software component. One of the major attribute for selection and integration the finest component could be complexity. It might also be a major factor for evaluating maintainability, reusability (as discussed in previous chapter) and overall efforts for developing the application. If the designer and integrator know the way to assess the complexity of the COTS component, he/she may compare a number of available COTS components, thus select the best one.

As the source code for the COTS component is not provided with them, they are considered to be black-box in nature. Application software, in which they are integrated, can interact with these COTS components only through their well specified interfaces. Interface specification may act as a primary and major source for understanding, implementing and using the COTS component. Interface predict out each and every element of a COTS component in some systematic and logical manner. Hence, the complexity of these interfaces plays a major role in calculating the overall complexity of these COTS components. Here, one important point that should be considered is that while calculating the overall complexity of a component
we might also consider integration complexity and not only the interface complexity (Narasimhan and Hendradjaya, 2004). Integration complexity is also known as interface coupling complexity (coupling complexity in short).

This chapter includes 11 major sections. In the next section, scope and the goal of the chapter is presented. Section 4.3 provides the complexity concepts and metrics provided by various researchers for the basic systems as well as for component based systems. Section 4.4 discusses the aim of this work and the solution strategy. Section 4.5 identifies the various factors and subfactors related to interface complexity while integrating the two or more COTS components. Section 4.6 describes the Analytical Hierarchy Process (AHP) through which the weights are assigned to these factors and subfactors. Section 4.7 presents the metrics system that supports the measurement by considering individual factors separately. Section 4.8 provides the theoretical validation of proposed metrics by considering weyunker’s property as base. Section 4.9 shows the experimentation work applied to these metrics. Section 4.10 is about the analysis of result and finally, section 4.11 summarizes with its limitations.

4.2 SCOPE AND GOAL OF THE CHAPTER

This chapter provides quantification and evaluation work of interface complexity in COTS Based Systems at the COTS integration time. We quantify interface complexity by proposing two coupling complexity metrics: Operational Coupling complexity Index (OCI) and Instance Coupling complexity Index (ICI) for the COTS software components, which are based upon the complexity involved in various interface operations and instance variables created between the interfaces. The interface operations can include various types of parameters and return values. This is the only kind of information that is available to us for that component’s interface method. The quantification is performed by considering the number of input, output parameters and their types. Based upon these factors of operations and instance variables, we used analytical hierarchy approach (AHP) to assign weights to these factors and outcomes OCI and ICI. These coupling complexity metrics can be used to find the strength of functional dependency between the components which are shown
in chapter 5 of the thesis. This chapter also shows the experimentation and theoretical validation of the proposed metrics.

4.3 RELATED WORK

IEEE defines software complexity as “the degree to which a system or component has a design or implementation that is difficult to understand and verify” (IEEE, 1990). Basili (1980) defines complexity as “a measure of the resources expended by a system, while interacting with a piece of software to perform a given task”. In case of the interacting system is a program, the complexity may be defined by the complexity of performing tasks such as coding, debugging and testing or the modifications in the software. There is no well defined method to calculate the software complexity. Hence, the term complexity measurement is difficult to define. The proper meaning of the term software complexity may be defined by zuse (1991) as the complicatedness to change, maintain and understand the software. The above defined definitions mainly relate software complexity with the complicatedness of performing a task on to the software. Here, some implied assumptions are that the software complexity connects in well with reusability and maintainability or the effort for the development of the application.

In the legacy software systems, complexity may be defined as the complicatedness to investigate the source code, to modify it, and to maintain its various modules. Here, in COTS-Based System, COTS components can only be added, customized or may be removed. Customization can be performed through its interface operations or properties.

4.3.1 Complexity Metrics for Basic Software System

Complexity is the main factor for estimating the reliability, cost and functionality of any software system. Various types of metrics have been proposed by various researchers for measuring different different characteristic of complexity such as data structures, size, inter-module structure and control flow. One of the most widely acceptable and famous complexity metric is Cyclomatic Complexity (CC)
metric, which was proposed by McCabe in 1976. This metric is based upon the program graph and may be defined as follows:

\[ V(G) = E - N + 2P \quad \text{-------------------------- (4.1)} \]

Here, \( N \) is the no. of nodes in the graph, \( E \) is the no. of edges and \( P \) is the no. of connected components. Author also showed that \( V(G) \) may be used for the measurement of procedural complexity of a program. Halstead (1977) developed a method, with importance on computational complexity to measure a program’s module complexity, which can be directly computed from the source code. The measurement is based upon the four scalar numbers, which can be derived directly from the program’s source code. The four scalar numbers are:

- \( n1 \): the no. of different operators,
- \( n2 \): the no. of different operands,
- \( N1 \): the total no. of operators,
- \( N2 \): the total no. of operands.

By using these four numbers, we can derive five basic measures, as shown in Table 4.1.

**Table 4.1: Halstead Complexity Metric (Halstead, 1977)**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Symbol</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program length</td>
<td>( N )</td>
<td>( N = N1 + N2 )</td>
</tr>
<tr>
<td>Program vocabulary volume</td>
<td>( n )</td>
<td>( n = n1 + n2 )</td>
</tr>
<tr>
<td></td>
<td>( V )</td>
<td>( V = N \times (\log_2 n) )</td>
</tr>
<tr>
<td>Difficulty</td>
<td>( D )</td>
<td>( D = (n1/2) \times (N2/n2) )</td>
</tr>
<tr>
<td>Effort</td>
<td>( E )</td>
<td>( E = D \times V )</td>
</tr>
</tbody>
</table>
These measurements served as strong indicators for the source code complexity in the legacy software systems. As these metrics are applied to the source code, they can be applied only after the coding stage and usually considered as maintenance metrics (Edmond, 2007).

Kafura and Henry (1981) also proposed the complexity metric which was based upon the no. of local data information flow enter (fan-in) and exit (fan-out) in every procedure or function. The metric can be formulated by eq. 4.2.

\[
\text{Complexity} = (\text{Proc. Length}) \times (\text{fan-in} \times \text{fan-out})^2
\]  

However from the COTS component’s point of view, these metrics could not be applied to determine the complexity of whole COTS component due to their measurement of the complexity for the procedure only. Thus, other characteristic of the components like attributes, classes and interface operations could not be considered, which are considered to be the main contributors for evaluating the complexity of the COTS components (Sharma et al., 2007).

Li (1993) also proposed some size based measurement on no. of methods and attributes metrics for complexity. Chidamber and Kemerer (1994) proposed a set of quality measures for class complexity in an object-oriented environment. They proposed Number of Children (NOC), Depth of Inheritance Tree (DIT), Coupling Between Objects (CBO), Weighted Methods Complexity (WMC), Lack of Cohesion in Methods (LCOM), Response for a Class (RFC). Out of these metrics NOC, DIT, RFC, CBO can be used for evaluating the external complexity of the relationships between various classes and it does not depend on the source code’s complexity of the procedures. WMC measures the complexity of the methods as shown by eq. 4.3:

\[
WMC(C) = \sum_{i=1}^{n} C_i
\]  

Here, \(C_i\) represents the static complexity for the corresponding method \(M_i\). Thus, it measures the complexity of methods for a class.
4.3.2 Complexity Metrics for Component Based System

First complexity metrics were proposed for Software components by Vernazza et al. (2000) by extending the metrics proposed by Li (1993). They considered that a component comprises group of classes, thus it is sensible to suppose that the complexity of the different classes may manipulate the overall complexity of the resultant component. They proposed weighted class per component (WCC) metric, which may be defined as:

\[ WCC = \sum_{i=1}^{n} NOM(C_i) \quad (4.4) \]

Where NOM is the no. of methods presented in the component. The other extended metrics are:

- NOCC: Number of Children for a Component, which counts the no. of children of all the classes in the component.
- EXTCBO: External Coupling between Objects counts the no. of external classes, coupled to it.
- RFCOM: Response Set for a Component is the no. of all the methods in the member classes and the methods called by those classes.
- Maximum of the DIT (MAXDIT) and Mean DIT of unrelated trees (MUT).

This paper also performed some validation tasks on the proposed metrics by taking the properties defined by Briand et al. (1999). However, the validation is based upon several theoretical properties only and not any empirical validation has been done on them.

In 2001, Cho et al. proposed several metrics for evaluating the customizability, complexity and reusability for CBS. They also classified the metrics into design time and run time metrics. Authors also considered interfaces, classes and relationships between various classes for measuring the complexity metrics. These metrics merged Cyclomatic Complexity (CC) with interfaces and sum of classes, hence obtained some static, plain, and some dynamic complex metrics. These metrics
can ultimately help the evaluator to estimate the various size estimates for the component. However, these metrics considered only the no. of constituents (classes, methods and interface methods etc), and did not considered the technical complexities for their constituents.

Nael (2006) proposed the structural complexity metrics for CBSS based upon the no. of components, connectors between components, interface of each component and composition tree. For each of these categories, author considered several metrics. For interfaces, he proposed total no. of interfaces and average no. of interfaces per component metrics. Though, the proposed metrics are very fundamental in character and are based upon just the numbers yet these did not consider any complexity of individual interface.

We can’t use most of the component metrics discussed above for the COTS based system level metrics as they are based upon the source code. Hence, we require designing several complexities metric for black-box commercial off -the-shelf components (COTS), that could be used by the application designer or integrators to select and integrate the best COTS component and help him/her to construct the better quality CBS. In COTS components, the complexity mainly exits due to the interface, their integration and semantics. Thus, in order to determine the whole complexity of the COTS component, we have to consider and determine all these factors.

Interface complexity metrics are the measures of the complexity of interfaces. Interface defines COTS components in terms of provided or required services and act as a base for its use and implementation. It is the first and may be the only source of information to understand the COTS component. It consists of a set of operations or methods which function as access point for their interaction with the other components. Integration metrics helps to measure the efforts required in the integration procedure of COTS components. Finally, semantic metrics helps to measure the complexity of the relationship between the component and the application. It was Mahmood and Lai (2005) who considered all these three factors for estimating the complexity from the UML component-based system specification. This type of early stage estimation of complexity of components from their
specifications helps to direct the effort required in maintaining and testing these components and hence producing reliable component based systems.

Boxall and Araban (2004) defined Interface Textual Complexity Metrics using a set of mathematical expressions. The metrics are based upon several measuring features of component interface, such as interface size, no. of various types of arguments in operations, level of repetition of these arguments, the synchronization in the identifiers, identifier length and the density and its level in the reference arguments. The proposed metrics provides a different and new way of measuring the complexity by analyzing the arguments used, still they considered all the arguments similar in nature hence, did not distinguished them on the basis of their data type and complexity occupied in them.

Gill and Grover (2004) also projected a new metric called Component Interface Complexity Metric (CICM) for measuring the interface complexity. The work is based upon the interface classification of software components. It proposes that the main cause of complexity of a component is due to its interface data signature, interface constraint, interface packaging and its configurations. An interface constraint entails the individual element and the relationship between these elements. An interface signature entails the functionality of components and comprises of attributes, their operations and events. An interface packaging and configuration, entails about a component’s usage in an application or in some other component. Interface signature and interface constraints deals with the internal functioning of the component therefore, depends upon the coding involved while the interface packaging and configuration deals with the use of the component after the development. Even though, the proposed metrics are very strong candidate for measuring interface complexity, yet empirical validation is still required.

Rotaru, Dobre, and Petrescu (2005) proposed the interfaces method based approach to measure the composability degree of software component. They considered various types of parameters and their return values of interface methods for measuring the complexity. They also accepted that the software component interfaces consisting only methods having no parameters and no return value will
possess the highest composability degree due to absence of any external dependencies. Similarly, the interface methods having both, parameters and return values will possess the lowest composability degree.

Sharma, Kumar, and Grover (2008) also projected some interface complexity metrics for software component. They considered interface methods, their related properties, argument types and return types. Although they evaluated the metrics using various java beans components and validated them against execution time, readability and customizability properties yet, an important factor of integration complexity did not taken into account.

Bhasin et al. (2011) had attempted to design a composite complexity measure to quantify important aspects of complexity of a component based system. The proposed measure takes into account two major complexities of a component based system: one due to individual component and the other due to its interaction with other components. Individual component complexity may arise due to size of a component, type and nesting level of control structures present in code component. Component’s interaction complexity may be due to its interface with other components. Graph theoretic notions and concept of weights were used to illustrate interaction among software components and to compute complexity. The proposed measures were applied to four cases chosen for the study but still the empirical validation of proposed work required. Further, as these metrics can’t be applied to COTS based systems in which coding of components are not available.

4.4 PROPOSED COMPLEXITY METRICS: AIM AND APPROACH

Here, we are extending the approach described in (Rotaru et al., 2005; Sharma et al., 2008; Bhasin et al. 2011) for proposing the new interface complexity metric for COTS components. Proposed metric uses the behavior or the signature of the COTS component by using their interface operations, which are available for the COTS component. Two components interact to each other when one component provides an interface and another component uses it.
For measuring the degree of interface integration complexity through operations and instances we propose OCI and ICI by normalizing it to 0-1. They are theoretical validated and applied through a small case study. The overall approach can have following steps:-

1) Identify and classify the various factors and subfactors for measuring operational complexity index and instance variable complexity index.
2) Assign weight to each factor and subfactors using Analytical Hierarchy Process (AHP) approach.
3) Propose OCI and ICI metrics.
4) Experimentation and theoretical Validation of the proposed metrics.

4.5 IDENTIFICATION AND CLASSIFICATION OF FACTORS FOR INTERFACE COMPLEXITY AT INTEGRATION LEVEL

To identify the various factors and subfactors first, we have to understand the basic structure of Component based software system. Based upon various literature reviews (Monge et al., 2000, Szyperski, 1998), A CBSS can be made up of many components. Each component consists of many Interfaces. The interface can be categorized as providing interface or receiving interface. Further each interface provides or receives functionality through operations or methods and instance variables. As components are black box in nature (especially COTS components), the interface operations provides information about only the number and type of Input and Output parameters. Similarly the instance variables will have types.

It can be depicted by figure 4.1.
According to chiller et al. (2012), Interface complexity defined in terms of the number of methods and instance variables invoked by component from other components. Weighted values were assigned to constituents of interfaces (methods and instance variables). Interface methods were classified according to data type of arguments and return values. Basic assumption in computing data type of instance variables, arguments and return type (interface methods) were: Primitive data types such as integer taken simple, Structured data type such as string, array, list, and vector taken medium, Complex data types includes class type, user defined components, pointers, references and others. If arguments were different for same interface method then higher data type taken. Similarly they compute data type of return types. Their result shows that complexity and dependency increases with increase in number of
invoked methods and instance variables during an interface in a component based system.

Similar approach of strengthening the various types of operations/methods in an interface using the weighted assignment technique was conducted by Kaur et al. (2010), and showed that the coupling increases with numbers and complexity of input and output parameters. Although, these methods considered operational/method level complexity by considering the number and type of arguments and instance variables in the interface as a major factor for calculating the complexity of the interfaces, yet we want to draw some important points which may be useful while considering these complexities as to quantify the functional dependency between the components through these interfaces.

These researchers (Chiller et al., 2012; Kaur et al., 2010) considered return value i.e. IN parameters as a part of OUT parameter and not considered it separately. We argue here that the IN parameter should be given equal weightage to the total number of OUT parameters because the methods with return values (IN parameters) will have more complexity than the methods without the return values but total number of arguments/parameters is equal in both the operations. For example a method with 2 simple and 3 complex OUT parameters and without any IN parameter should not be considered equal to the method with 2 simple and 2 complex OUT parameters but with one complex return value.

Another point that should be considered, that the number of parameters is not only the sole criteria in deciding the complexity of the operations. It is the combination of many factors and sub factors. Accordingly, we are classifying interface operations into four categories. Each return value is considered as IN parameter and arguments passed as OUT parameters.

- Interface operations without IN Parameter and without OUT Parameters
- Interface operations with IN Parameter but without OUT parameters
- Interface operations without IN Parameter but with OUT parameters
- Interface operations with IN Parameter and with OUT parameters
Further both IN and OUT parameters complexity can vary according to data types of these parameters (TOP). So we categorized them into three types as simple, medium and complex in the same way as in (Chillar et al., 2012). Primitive data types such as integer, Boolean, double etc. will be considered as simple (S), Structured data type such as string, array, list, and vector will be considered as medium (M), and class type, user defined components, pointers and references will be considered as complex(C).

We will take another factor into consideration that the overall weight for each data type will be taken different according to number of these data types present in these operations. We have assigned equal weightage to IN parameters and OUT parameters i.e. 0.5 to each. As return value either maybe or not. So IN parameters can have four sub factors:- No return value (NR), Return Simple (RS), Return Medium (RM), Return Complex (RC). OUT parameters numbers (NOP) may further be divided into no out parameters (NO), out parameters from 1 to 4 (1-4), out parameters from 5 to 8 (5-8), and out parameters greater than 8 (>8).

According to various combinations, the OUT parameters subfactors considered are No OUT parameters (NO), 1-4 simple type parameters (1-4S), 1-4 medium type parameters (1-4M), 1-4 complex type parameters (1-4C), 5-8 simple type parameters (5-8S), 5-8 medium type parameters (5-8M), 5-8 complex type parameters (5-8C),>8 simple type parameters (>8S), >8 medium type parameters (>8M), >8 complex type parameters (>8C). In the similar way to OUT parameters instance variables can have same factors.

4.6 ASSIGN WEIGHT TO EACH FACTOR AND SUBFACTORS USING ANALYTICAL HIERARCHY PROCESS (AHP) APPROACH.

Researchers (chiller et al., 2012; kaur et al., 2010) used weighted assignment technique to assign weights to these factors. Here we will use Analytical Hierarchy Process (AHP) approach to assign weights to these factors. AHP approach enables the decision maker to structure a decision making problem into a hierarchy, thus, helping him/her to simplify and understand the exact problem. Although it is considered to be
a time consuming technique due to the mathematical calculations and a lot of pair wise comparisons which increases as number of alternatives and criteria increases or changes yet it is preferred MCDM method due to its flexibility and accuracy in the result. It provides a widespread and rational decision making framework that proves a powerful methodology for determining relative strength among a set of elements (Jayswal et al., 2007). Another famous old method that can be applied to develop such type of framework is Weighted Scoring Method (WSM). WSM technique is easy to understand and use but weights to the attribute are assigned subjectively and it is very difficult to assign when the number of criteria becomes high. Another problem with weighted scoring method is that some common numerical scaling may be required to obtain the final score. In AHP approach decision makers can compare each alternative that improves decision making process by compliant the ambiguity of human decision making. AHP found to be most appropriate for the complex decision which involves the comparison of decision elements that are usually difficult to quantify. AHP have been applied effectively in various engineering and non-engineering areas, like corporate planning, transportation planning, portfolio selection, marketing etc. (Satty, 1980). It consists of mainly:

a) The development of comparative importance among the factors and sub factors using expert’s opinion or throughout comprehensive paired comparison analysis,
b) Assigning a weightage for each of the factor and sub factor using priority vector,
c) Performing analogous analysis for each of the alternative solution strategy and for each of their attributes, and
d) Developing a single overall score for each of the alternative solution strategy.

The total score for all alternates will be 1.

Various researchers (Chen et al., 2004, Sharma et al., 2008) have successfully applied this method to assign weight to various factors in the field of computer engineering.
The priority for each alternative is assigned as follows:-

IN parameters (IN) and OUT parameters (OUT) have been assigned equal priority in deciding the coupling complexity between the components, that is, IN=OUT

For IN parameters NR < RS < RM < RC.

For OUT parameters Number of out parameters (NOP) has equal weightage to type of parameter (TOP), that is, NOP=TOP

Further for NOP the priority assigned will be as follows:-

\[0 < 1-4 < 5-8 < (>8)\]

TOP priority is assigned as: - Simple (S) < Medium (M) < Complex (C)

We have used an open source tool Open Decision Maker (ODM) v 1.0.1, (described in Appendix A) to make sensitivity analysis between the alternative factors and to calculate the overall weights using AHP. The detailed AHP analysis report for the IN parameters and OUT parameters for OCI has been presented in Appendixes C, and D respectively. The critical consistency ratio is under 0.1 in all the cases. The results of this analysis have been shown in the Table 4.2.

**Table 4.2:** Weighted values of IN parameters and OUT parameters of operations*

<table>
<thead>
<tr>
<th>Factors</th>
<th>Sub-Factors</th>
<th>Weight Value</th>
<th>Sum</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>No Return Value (NR)</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return</td>
<td>Simple (RS)</td>
<td>0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium (RM)</td>
<td>0.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complex (RC)</td>
<td>0.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>No OUT Parameter (NO)</td>
<td>0.008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-4</td>
<td>Simple (1-4 S)</td>
<td>0.012</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Medium (1-4 M)</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complex (1-4 C)</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TABLE 4.3: Weighted values of Instance variables

<table>
<thead>
<tr>
<th>Factors</th>
<th>Weight Value</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple (1-4 S)</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>Medium (1-4 M)</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>Complex (1-4 C)</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>5-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple (5-8 S)</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>Medium (5-8 M)</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>Complex (5-8 C)</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>&gt;8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple (&gt;8 S)</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>Medium (&gt;8 M)</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>Complex (&gt;8 C)</td>
<td>0.34</td>
<td>1</td>
</tr>
</tbody>
</table>

#Based upon analysis report in appendix E
4.7 THE PROPOSED METRICS

The proposed Operational Coupling Complexity Index (OCI) and Instance Coupling Complexity Index (ICI) metrics can be defined using the weights assigned to them in the previous section.

4.7.1 The OCI metric

Operational Coupling complexity Index (OCI) can be calculated as follows:

\[
OCI = \frac{\text{Weighted value of IN parameter factor} + \text{Sum of weighted value of OUT parameter factors}}{\text{IN Parameter value set}}
\]  

Where IN Parameter value set can be IN= {0.02, 0.06, 0.13, 0.29} depending upon the above sub factors and OUT Parameter value is the combination of the weights of all the sub factors present in the operation according to type of parameter (TOP) and number of parameters (NOP).

4.7.2 The ICI metric

Instance Variable Coupling complexity Index (ICI) can be calculated as

\[
ICI = \text{Sum of weighted value of instance variable factors}
\]

4.8 THEORETICAL EVALUATION OF PROPOSED METRICS USING WEYUNKER’S PROPERTIES

Weyuker (1988) proposed properties to theoretical validate software complexity measures. Several researchers have suggested various properties that software metrics should posses to improve their usability. Although Weyuker’s properties are also criticized by various researchers (Salman, 2006; Saleh, 2004), yet it proves to be an important base to classify a complexity measure and hence coupling complexity index measure. Weyuker’s properties state that (Weyunker, 1988).

Property 1: There may be some programs/functions, P and Q for which M(P)
≠ M(Q). Here, M is the metric value.

**Property 2:** If C is any non-negative number, then there might be finitely many programs/functions P for which M(P) = C.

**Property 3:** There may be distinct functions/programs P and Q for which M(P) = M(Q).

**Property 4:** There may be functionally equivalent functions/programs P and Q for which M(P) ≠ M(Q).

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

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

**Property 7:** There exists program/function 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 renaming of Q, then M(P) = M(Q).

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

We have evaluated these properties for the proposed interface metrics as described below:

I. There may be two different functions with different interface integration complexities, therefore satisfying the property 1.

II. As each function will possess at least one operation with some functionality, thus its complexity will always have some positive value. Hence, validating the property 2.

III. For two different functions having different functionality, the proposed integration complexity metric value may be same, as these operations may
comprise same type and number of parameters but with different functionality. It satisfies Property 3.

IV. Even if the functionality of the two different functions are same, both may possess different interface integration complexity because these functions may have been designed by using different programming concepts and contains different types and number of parameters. It validates property number 4.

V. If a function is assembled with other function for the integrated function to enhance functionality, the interface complexity of these two independent functions shall be lower than the interface complexity of the integrated function, hence satisfying the property 5.

VI. Two operations with the same complexity mean that both operations will possess the same number and type of IN and OUT parameters. Still it is possible that they may be developed by using different programming methodologies thus when integrating them in the function, both may have different combination with other functions and instance variables. Hence possessing different interface complexities of the different integrations in every case. It confirms property number 6.

VII. If the ordering of operations in a function is changed, then it will not change the overall interface complexity of the modified function. So, the property no. 7 does not satisfied by our proposed functional dependency metric.

VIII. It is understandable that renaming an operation or a function will not influence the interface complexity of that function, thus satisfying the property no 8.

IX. When any two functions are assembled, then we may have to write some more operations related with the integration in addition to the existing operations. It will increase the overall complexity of the assemble function. It validates the property no. 9.

The graphical results shown in the next section proves these statements. In this way, eight out of nine Weyuker’s properties are satisfied by the proposed interface complexity metrics.
4.9 EXPERIMENTATION ON PROPOSED METRICS

As discussed in the fifth section, the various types of operations can be present in a function. Each type of function will possess different value for OCI, and hence will influence the integration complexity. The various possible minimum and maximum value cases for these different types of operations can be shown as follows:

**CASE I** Operations with No IN Parameter (NR) and No OUT Parameters (NO)

\[ \text{OCI} = 0.02 + 0.008 = 0.028 \]

**CASE II** Operations with IN parameter but No OUT Parameters

*Subcase 2.1 (Minimum value)* Return Simple (RS) value but No OUT (NO) parameters

\[ \text{OCI} = 0.06 + 0.008 = 0.068 \]

*Subcase 2.2 (Maximum Value)* Return Complex (RC) value but No OUT (NO) parameters

\[ \text{OCI} = 0.29 + 0.008 = 0.298 \]

**CASE III** Operations with No IN Parameter (NR) but with OUT parameters

*Subcase 3.1 (Minimum Value)* No return Value (NR) and 1 simple (1-4 S) OUT parameter

\[ \text{OCI} = 0.02 + 0.012 = 0.032 \]

*Subcase 3.2 (Maximum Value)* No return (NR) value and 10 simple (>8S), 10 complex (>8C) and 10 medium (>8M) type OUT Parameters

\[ \text{OCI} = 0.02 + 0.012 = 0.032 \]
OCI = 0.02 + 0.04 + 0.08 + 0.16 = 0.30

**CASE IV Operations with IN parameters and OUT Parameters**

*Subcase 4.1 (Minimum Value)* Return Simple (RS) value and 1 simple (1-4 S) OUT parameter

\[
OCI = 0.06 + 0.012 = 0.072
\]

*Subcase 4.2 (Maximum Value)* Return Complex (RC) value and 10 simple (>8S), 10 complex (>8C) and 10 medium (>8M) type OUT Parameters

\[
OCI = 0.29 + 0.04 + 0.08 + 0.16 = 0.57
\]

Similarly, the ICI values for the minimum and maximum cases can be depicted as follows.

*CASE 5.1 (Minimum ICI value):* 1 Simple type instance variable.

ICI = 0.02

*CASE 5.2 (Maximum ICI value):* 10 Simple type, 10 medium type and 10 complex type instance variables.

ICI = 0.08 + 0.17 + 0.34 = 0.59

4.10 RESULT ANALYSIS

Graphically, the result values for OCI and ICI for all the above cases can be analyzed by graphical figures 4.2, 4.3 and 4.4.
Thus the minimum value of OCI among all the cases will be in case I, that is
0.028 and the maximum value among all the cases will be in case IV, that is 0.57 (see figure 4.2 and 4.3). The minimum value for ICI will be for 1-4 S i.e. 0.02 and maximum value for ICI will be 0.59 (See figure 4.4). Thus the graph shows that the ICI and OCI is not the sole measure of one factor. It is not a simple linear scale because it is the combination of many factors and subfactors.

4.11 SUMMARY

The chapter summarizes several complexity metrics projected by various researchers particularly for COTS based systems. Most of the proposed metrics are based upon the source code of the component, hence cannot be applied on the COTS components by the designers or integrators of these components during their application development.

We provides quantification and evaluation work of interface complexity in COTS Based Systems at the COTS integration time. We quantify interface complexity by proposing two coupling complexity metrics: Operational Coupling complexity Index (OCI) and Instance Coupling complexity Index (ICI) for COTS software components that are based upon the complexity involved in the interface operations and instance variables created between the interfaces. The interface operations could contain parameters and their associated return value that is the only type of information which will be available to us related to that interface function. The quantification is performed by considering the number of input, output parameters and their types. Based upon these factors of operations and instance variables, we used analytical hierarchy approach to allocate weights to these factors which outcomes OCI and ICI.

This work evaluated the proposed metric theoretically by using Weyuker’s properties and then experimentation work is shown on these metrics through the cases of minimum and maximum values for all the possible combinations of methods. The result graph provides how these metrics behave in all the cases. These coupling complexity metrics can be used to find the strength of functional dependency between the components which are shown in chapter 5 of the thesis.