CHAPTER 4
INTERACTION AND INTEGRATION COMPLEXITY METRICS FOR
COMPONENT-BASED SOFTWARE

4.1 Introduction

In Component-Based Software development, developer’s emphasis is on the assembling and integration of pre-constructed, pre-examined, customizable, and easily deployable software components, rather than developing software from scratch. Integration and Interaction among the components follow a well-defined architectural design and should be according to the user’s specification of requirements. Components interact with each other for two basic reasons:

a) To access the services and functionalities of other components, and

b) To provide the services and their own functionalities to other components.

These interactions among various components lead to the generation of complexity in Component-Based Software. In general, the term complexity is defined as the assessment and calculation (sometimes prediction) of resources required resolving a problem. In the domain of software development, complexity is an attribute which cannot be measured directly, that is, complexity is an indirect measure. Software complexity is a term that encompasses numerous properties of a piece of software, all of which affect internal as well as external interactions. Complexity may be characterized as Computational, Algorithmic, or Information processing [71]. Computational complexity focuses on the amounts of resources required for the execution of algorithms like space and time, whereas the information processing complexity is a measure of the total number of units of information transmitted by an object. In software engineering, complexity is a measure of the interactions of the various elements or components of the system. As the number of entities increases, the number of interactions between them would increase as per the software requirements.

This chapter aims to develop and suggest competent and proficient interaction complexity estimation measurements and metrics. In this chapter, two complexity computation techniques are suggested:

i) In-Out Interaction complexity, and

ii) Cyclomatic complexity for Component-Based Software.
4.2 In-Out Interaction Complexity Graph

This section describes the proposed In-Out interaction complexity metrics to capture and compute the complexity produced due to the interactions among different components.

4.2.1 Notations

To represent the interactions among components graph theory notations are used. When two or more components are integrated then they must share interactions [38]. To represent these interactions, two types of edges have been used: _In-Interactions and Out-Interactions_.

i) **In-Interaction**: In-Interactions are the edges incoming to a component, which come from some other components. These incoming edges are the responses of requests made by the calling component. In the Figure 4.1, the edge coming from component C2 to component C1 is the In-Interaction for C1 and denoted as \( I_{in} \).

ii) **Out-Interaction**: Out-Interactions are the edges outgoing from a component to some other components. These outgoing edges are the request edges to some other called components. In the Figure 4.1, the edge going from component C1 to component C2 is the Out-Interaction for C1 and denoted as \( I_{out} \).

A _request edge (Out-Interaction)_ is drawn from the calling component to the called component when a component makes a request for some service to another component. Further, the called component responds either in the form of some service or through some message. This scenario is shown by the _response edge (In-Interaction)_ from the called component to the calling component [8]. In the coding, these interactions take place through some parameters or operands. The calling component passes some parameters to the called component and in response the called component will return some parameters to the calling component, as shown in the Figure 4.1.

During the computation of In-Out Interaction complexity both the edges are taken into account. In Figure 4.1 Component C1 has one Out-Interaction \( I_{out} \) as a request edge from C1 to C2 and one In-Interaction \( I_{in} \) as a response edge coming from C2. Since components C1 and C2 both are dependent on each other for request and response, to calculate the overall interaction complexity of this scenario, the complexities generated due to both components C1 and C2 are considered. So component C2 has one Out-Interaction in the form of a response edge from C2 to C1 and one In-Interaction in the form of a request edge coming from C1.
Similarly, such notations can be used for scenarios where more than two components interact with each other.

In the present work, four components C1, C2, C3, and C4 are considered and shown in Figure 4.2. It is observed that the component C1 has 3 In-Interactions and 3 Out-Interactions, component C2 has 1 In-Interaction and 1 Out-Interaction, component C3 has 4 In-Interactions and 4 Out-Interactions, and component C4 has 4 In-Interactions and 4 Out-Interactions.
4.2.2 Total-Interactions of a Component (TI_C)
Total Interactions of a component is computed as the sum of In-Interactions (I_in) to, and Out-Interactions (I_out) from, that component and defined as:

\[ TI_C = I_{out} + I_{in} \]  

(4.1)

where, I_in is In-Interaction and I_out is the Out-Interaction.

4.2.3 Total-Interactions of Component-Based Software (TI_CBS)
Total Interactions of Component-Based Software is computed as the sum of the total number of In-Interactions (I_in) and total number of Out-Interactions (I_out) of all participating components, as:

\[ TI_{CBS} = \sum_{i=1}^{n} I_{out} + \sum_{i=1}^{n} I_{in} \]  

(4.2)

where ‘n’ is the number of components in the Component-Based Software.

4.2.4 Interaction-Ratio of a Component (IR_C)
The value of Interaction-Ratio metric shows the degree of dependency of a component. The Interaction ratio of a component is computed as the total number of Out-Interactions (I_out) divided by the total number of In-Interactions (I_in) of a component. It is defined as:

\[ IR_C = \frac{I_{out}}{I_{in}} = \begin{cases} 
< 1 & I_{out} < I_{in}, \text{ high dependency.} \\
= 1 & I_{in} = I_{out}, \text{ equal dependency.} \\
> 1 & I_{out} > I_{in}, \text{ high dependency.} 
\end{cases} \]  

(4.3)

We can verify that, if IR_C = 1 when I_in = I_out, which results interacting components are equally dependent on each other, IR_C < 1 when I_out < I_in, which results other interacting components are highly dependent on this component, and IR_C > 1 when I_out > I_in, implies this component is highly dependent on other components.

4.2.5 Interaction-Ratio of Component-Based Software (IR_CBS)
The Interaction-Ratio of Component-Based Software is computed as the sum of the total number of Out-Interactions (I_out) divided by the sum of the total number of In-Interactions (I_in) of all participating components. It is given as:
\[ \text{IR}_{CBS} = \frac{\sum_{i=1}^{n} I_{\text{out}}}{\sum_{i=1}^{n} I_{\text{in}}} \]  

(4.4)

where ‘n’ is the number of components in the Component-Based Software.

### 4.2.6 Average-Interactions among Components (AI_{Cn})

The Average-Interaction metric shows the degree of connectivity among the components. The average Interaction between components can be calculated as the ratio of the sum of In-Interactions \((I_{in})\) and Out-Interactions \((I_{out})\) to the number of components participating in the integration \(C_n\). It is defined as:

\[
\text{AI}_{Cn} = \frac{(I_{in} + I_{out})}{I_{max}}
\]

(4.5)

We can verify that, if \(\text{AI}_{Cn} < \frac{1}{2}\), it implies that at least half of the interacting components are disjoint, if \(\text{AI}_{Cn} = 1\), it shows there is at least one interaction among components, and if \(\text{AI}_{Cn} > 1\), then components are highly coupled.

### 4.2.7 Interaction-Percentage of Components (IP_{Cn})

The Interaction-Percentage metric shows the level of underflow or overflow of interaction paths among components. The percentage of components \(\text{IP}_{Cn}\) is the ratio of the summation of In-Interactions \((I_{in})\) and Out-Interactions \((I_{out})\) to the Maximum possible interactions \((I_{max})\) among the components. It is defined as:

\[
\text{IP}_{Cn} = \left(\frac{I_{in} + I_{out}}{I_{max}}\right) \times 100
\]

(4.6)

We can verify that, if \(\text{IP}_{Cn} < 1\), it implies the underflow condition, that is, more interactions are possible among the components, if \(\text{IP}_{Cn} = 1\), it shows the balanced condition, and if \(\text{IP}_{Cn} > 1\), then it shows overflow condition, that is, components are sharing heavy interaction, which will increase the complexity.
4.3 Proposed Cyclomatic Complexity for Component-Based Software

According to graph theory, the Cyclomatic number of a graph $G$ can be defined as $(e - n + p)$, where $e$ is the number of edges, $n$ is the number of nodes and $p$ denotes the number of strongly connected components. In the graph theory, Cyclomatic complexity represents the number of fundamental cycles of a graph [17]. In the context of computer programs, Cyclomatic complexity reflects the total number of independent paths in a program. Ultimately, it identifies the independent logics used in the program and provides a quantitative measure of the logical complexity of a program.

4.3.1 Flow Graph Notation for Component Integration in CBSE

The design documents are very crucial documents of the software development. These documents are used to verify the coupling of components at the time of integration. When a component makes a request for some service to another component, a request edge is drawn from the calling component to the called component. Further, the called component responds either in the form of some service or through some message. This scenario is shown by the response edge from the called component to the calling component as shown in Figure 4.1. These request and response edges create a closed region between two interacting components. The closed region contributes to the inter-component Cyclomatic complexity of the software. Individual complexities of components also contribute to the overall complexity of the Software. Higher interaction among components increases the overall complexity because of more coupling among components [82].

![Figure 4.3 Interaction Scenario with Two Components](image-url)
4.3.2 Cyclomatic complexity Metric

For a control flow graph $G$, drawn from the sequence of statements used in component development, the Cyclomatic complexity $V(G)$ for component based software is suggested as:

$$V(G) = |E| - |V| + 2 + |P|$$  \hspace{1cm} (4.7)

where $|E|$ is the cardinality of the set of edges in a control flow graph, $|V|$ is the cardinality of the vertex set and $|P|$ is the cardinality of interacting components. The constant 2 is used to indicate that the node $V$ contributes to the complexity if its out-degree is 2.

To validate the Cyclomatic complexity proposed in Equation (4.7), we suggest the metric as:

$$V(G) = \sum_{i=1}^{n} (IC)_i + \sum_{j=1}^{m} (CR)_j + OR$$  \hspace{1cm} (4.8)

where, $(IC)_i = (IC_1, IC_2, IC_3, \ldots, IC_n)$ is the Cyclomatic complexity of ‘$n$’ interacting components, $(CR)_j = (CR_1, CR_2, CR_3, \ldots, CR_m)$ is the number of closed regions, and OR is the open region.

**Minimum Level of Cyclomatic Complexity:** In a control flow graph, as the number of decision nodes and edges increases the value of the Cyclomatic complexity also increases. To define the minimum level of the Cyclomatic complexity of a CBS system we have assumed that each component has unit complexity, i.e. 1. For the minimum level of interaction, each component has one ReqEd (Request Edge) and one ResEd (Response Edge). Now the minimum level of the Cyclomatic complexity can be defined as:

$$V(G) = 2 \times C_n$$  \hspace{1cm} (4.9)

where, $C_n$ is the total number of components.

Scenario based estimation is taken into account to assess the minimum level of Cyclomatic complexity of interacting components ($P$) in the CBSE. We have considered following three scenarios:

**Scenario 1: When the cardinality of the set of components is ‘2’**

Let us consider the component interacting scenario shown in Figure 4.4, there are two components $C1$ and $C2$ and their cardinality is $|P| = 2$. There are 2 connecting edges between these components, i.e. request edge (ReqEd) and response edge (ResEd). This scenario derives two regions, one is the open region (OR) and the other is the closed region (CR).

From Equation (4.7),
\[ V(G) = |E| - |V| + 2 + |P| \]

If we have \( E = 2 \), \( V = 2 \), and \( P = 2 \), Complexity of \( C_1 \) = 1, Complexity of \( C_2 \) = 1, then,
\[ V(G) = 2 - 2 + 2 + 2 = 2 + 2 = 4. \]

**Scenario 2: When the cardinality of the set of components is ‘n’**

If we have ‘n’ components (denoted by nodes) in a system, we need at least \((n - 1)\) edges to connect them. Since components interact through requests and responses, we will require at least \((n - 1)\) ReqEd and corresponding \((n - 1)\) ResEd.

From Equation (4.7),
\[ V(G) = |E| - |V| + 2 + |P| \]

If we have \( P = n \), then there are \( 2 \ast (n - 1) \) edges in total, therefore,
\[ V(G) = 2 \ast (n - 1) - n + 2 + n \\
= 2 \ast n. \]

**Scenario 3: When the cardinality of the set of components is ‘n+1’**

If we have \( n + 1 \) components in a system, we need at least \((n - 1) + 1\), i.e. \( n \) edges to connect them. In this case we will require at least \((n - 1) + 1\) ReqEd, and \((n - 1) + 1\) ResEd to connect these components.

From Equation (4.7),
\[ V(G) = |E| - |V| + 2 + |P| \]

If we have \( P = n + 1 \), then there are \( 2 \ast (n - 1 + 1) \) edges in total. Therefore,
\[ V(G) = 2 \ast (n - 1 + 1) - (n + 1) + 2 + n + 1 \\
= 2 \ast (n + 1). \]
4.4 Case Studies
To compute the Cyclomatic complexity of a component based software system using the proposed method, first we have to draw the control flow graph of the components. In this work, six cases are analysed for proposed metric. In Case 1, the directed graph consists of two components and they derive one closed region through interaction edges. Case 2 involves three components and two closed regions. Case 3 involves three components and four closed regions. Case 4 consists of four components and three closed regions. Case 5 consists of four components and seven closed regions. Case 6 has five components and ten closed regions. Other than these closed regions, each case has an open region. Every component that interacts with another derives a closed region. Every region outside the component that is not closed is an open region. All closed and open regions contribute to the complexity of the software.

Case 1: Interaction Between Two Components (One closed region and open region)
Figure 4.4 depicts the interaction between two components. Components interact through one request edge and one response edge, and derive only one closed region CR. Components C1 and C2 have their own Cyclomatic complexities, C1 has complexity 1 and C2 has complexity 2; they contribute to the overall complexity of the integrated system. There is an open region (OR). If there are 9 nodes (V), 10 edges (E), and 2 interacting components (P), then,
- Cyclomatic complexity using McCabe method:
  \[ V(G) = e - n + 2p = 10 - 9 + 2 \times 2 = 5 \]
- Cyclomatic complexity using the proposed metric:
  \[ V(G) = |E| - |V| + 2 + |P| = 10 - 9 + 2 + 2 = 5 \]

Verification is carried out as:
The obtained result from the proposed metric is verified with the help of Equation (4.8) as:
If i = 2, j = 1, IC1 = 1, IC2 = 2, CR = 1, then
\[ V(G) = (1 + 2) + 1 + 1 = 5 \]

Case 2: Interaction Between Three Components (Two closed regions and an open region)
In this case, three components C1, C2 and C3 are considered as shown in Figure 4.5. Component C1 interacts with C2 and C3; therefore they derive two closed regions CR1 and CR2 respectively. There is no interaction between C2 and C3; leads to no closed region between these components. All these components have their individual complexities, i.e. C1 has complexity 2, C2 has complexity 2 and C3 contributes a complexity value 4. There is a single open region represented
by OR. From Figure 4.5, it is observed that the number of edges (E) is 22, number of nodes (V) is 16, and the numbers of interacting components (P) are 3.

- Cyclomatic complexity using McCabe method:
  \[ V(G) = e - n + 2p = 22 - 16 + 2 \times 3 = 12 \]

- Cyclomatic complexity using the proposed metric:
  \[ V(G) = |E| - |V| + 2 + |P| = 22 - 16 + 2 + 3 = 11 \]

**Verification is carried out as:**

The obtained result from the proposed metric is verified with the help of Equation (4.8) as:

If \( i = 1 \) to 3, \( j = 1 \) to 2, \( IC_1 = 2, IC_2 = 2, IC_3 = 2 \), \( CR = 1 \), then

\[ V(G) = (2 + 2 + 4) + 2 + 1 = 11 \]

![Figure 4.5 Interaction Scenario with Two Closed regions and an Open region](image)

**Case 3: Interaction Between Three Components (Four closed regions and an open region)**

The depiction of Case 3 is shown in Figure 4.6. This case is same as the case 2, where there are three components C1, C2 and C3, however there are more interactions than the case 2. Component
C1 interacts with C2 and C3, so they derive two closed regions CR1 and CR3 respectively. An interaction between C2 and C3 leads to the other closed region CR4. A new closed region CR2 is created by the combined mutual interactions between C1, C2 and C3. There is a single open region represented by OR. The individual Cyclomatic complexities of components C1, C2 and C3 are 2, 2 and 4 respectively. From Figure 4.6, it is observed that the number of nodes (V) is 16, the number of edges (E) is 24, and the number of interacting components (P) is 3.

![Figure 4.6 Interaction Scenario with Four Closed regions and an Open region](image)

- Cyclomatic complexity using McCabe method:
  \[ V(G) = e - n + 2p = 24 - 16 + 2 \times 3 = 14 \]
- Cyclomatic complexity using the proposed metric:
  \[ V(G) = |E| - |V| + 2 + |P| = 24 - 16 + 2 + 3 = 13 \]

**Verification is carried out as:**

The obtained result from the proposed metric is verified with the help of Equation (4.8) as:

If \( i = 1 \) to 3, \( j = 1 \) to 4, \( IC_1 = 2, IC_2 = 2, IC_3 = 4, CR = 1 \), then
\[ V(G) = (2 + 2 + 4) + 4 + 1 = 13 \]
Case 4: Interaction Between Four Components (Three closed regions and an open region)

This scenario is depicted in Figure 4.7. In this case there are four components C1, C2, C3 and C4. Component C1 interacts with C2 and C3 so they derive two closed regions CR1 and CR3 respectively. Component C2 interacts with component C4 and derives a closed region CR2.

From Figure 4.7, it is observed that there is an open region (OR), three closed regions CR1, CR2, and CR3. The Cyclomatic complexities of C1, C2, C3 and C4 are 2, 2, 4, and 1 respectively. Number of nodes (V) is 21, number of edges (E) is 28, and the numbers of interacting components (P) are 4.

- Cyclomatic complexity using McCabe method:
  \[ V(G) = e - n + 2p = 28 - 21 + 2 \times 4 = 15 \]
- Cyclomatic complexity using the proposed metric:
  \[ V(G) = |E| - |V| + 2 + |P| = 28 - 21 + 2 + 4 = 13 \]

Verification is carried out as:

The obtained result from the proposed metric is verified with the help of Equation (4.8) as:
If $i = 1$ to $4$, $j = 1$ to $3$, $IC_1 = 2$, $IC_2 = 2$, $IC_3 = 2$, $IC_4 = 1$, $CR = 1$, then

$$V(G) = (2 + 2 + 4 + 1) + 3 + 1 = 13$$

**Case 5: Interaction Between Four Components (Seven closed regions and an open region)**

This scenario is depicted in Figure 4.8. This case is same as the case 5, where there are four interacting components $C_1$, $C_2$, $C_3$ and $C_4$, however there are more interactions than the case 4. Component $C_1$ interacts with $C_2$, $C_3$ and $C_4$ deriving the closed regions $CR_1$, $CR_3$, and $CR_4$. Component $C_2$ interacts with $C_4$ and deriving a closed region $CR_6$. Component $C_3$ is interacting with $C_4$ yielding a closed region $CR_7$. Interaction among components $C_1$, $C_2$, and $C_5$ creates a new closed region $CR_2$. Interaction among components $C_1$, $C_3$ and $C_5$ yields a closed region $CR_5$. There is an open region $OR$. All these closed regions and an open region contributes to the total complexity of the integrated software. From Figure 4.8, it is observed that the Cyclomatic complexity of $C_1$ is 2, $C_2$ is 2, $C_3$ is 4, and $C_4$ is 1, number of nodes ($V$) is 21, number of edges ($E$) is 32, and the number of interacting components ($P$) is 4.

![Figure 4.8 Interaction Scenario with Seven Closed regions and an Open region](image-url)
• Cyclomatic complexity using McCabe method:
  \[ V(G) = e - n + 2p = 32 - 21 + 2 \times 4 = 19 \]

• Cyclomatic complexity using the proposed metric:
  \[ V(G) = |E| - |V| + 2 + |P| = 32 - 21 + 2 + 4 = 17 \]

**Verification is carried out as:**

The obtained result from the proposed metric is verified with the help of Equation (4.8) as:

If \( i = 1 \) to \( 4 \), \( j = 1 \) to \( 7 \), \( IC_1 = 2 \), \( IC_2 = 2 \), \( IC_3 = 4 \), \( IC_4 = 1 \), \( CR = 1 \), then

\[ V(G) = (2 + 2 + 4 + 1) + 7 + 1 = 17 \]

**Case 6: Interaction Between Five Components (Ten closed regions and an open region)**

This scenario is depicted in Figure 4.9. In this case C1, C2, C3, C4 and C5 are five interacting components. Here component C1 is interacting with C2, C3 and C5 and derives closed regions CR1, CR3, and CR4 respectively. Component C2 is interacting with C5 forming closed regions CR2 and CR6. Component C3 is interacting with C4 and C5 deriving closed regions CR5, CR7 and CR8. Component C4 is interacting with C5 forming closed region CR10.

![Figure 4.9 Interaction Scenario with Ten Closed regions and an Open region](image-url)
Since components C3, C4 and C5 are interacting with each other they derive a closed region CR9. There is an open region OR.

From Figure 4.9, it may be noted that the Cyclomatic complexities of C1, C2, C3, C4 and C5 are 2, 2, 4, 1 and 4, number of nodes (V) is 28, number of edges (E) is 44, and the number of interacting components (P) is 5.

- Cyclomatic complexity using McCabe method:
  \[ V(G) = e - n + 2p = 44 - 27 + 2 * 5 = 27 \]
- Cyclomatic complexity using the proposed metric:
  \[ V(G) = |E| - |V| + 2 + |P| = 44 - 27 + 2 + 5 = 24 \]

**Verification is carried out as:**

The obtained result from the proposed metric is verified with the help of Equation (4.8) as:

If \( i = 1 \) to \( 10 \), \( IC_1 = 2 \), \( IC_2 = 2 \), \( IC_3 = 4 \), \( IC_4 = 1 \), \( IC_5 = 4 \), \( CR = 1 \), then

\[ V(G) = (2 + 2 + 4 + 1 + 4) + 10 + 1 = 24 \]

**Performance Result**

From the above defined case studies it is observed that using the presented method complexity of the components interactions can be reduced. Performance of the presented metric increases as the number of components increases. Comparisons are made with the McCabe’s method and achieved better results in terms of reduced complexity.

<table>
<thead>
<tr>
<th>Number of Components</th>
<th>McCabe’s Cyclomatic Complexity</th>
<th>Proposed Cyclomatic Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (One Closed Region)</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3 (Two Closed Regions)</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>3 (Four Closed Regions)</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>4 (Three Closed Regions)</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>4 (Seven Closed Regions)</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>5 (Ten Closed Regions)</td>
<td>27</td>
<td>24</td>
</tr>
</tbody>
</table>
4.5 Summary

Interaction and Integration metrics are useful to compute the complexity of those third-party components for which code is not available, since we have included only the interaction attributes of the components and we have not attempted to analyze the logic or code. Computations related to these metrics included the complexities generated due to incoming as well as outgoing edges of a component. Every component in the Component-Based Software has some incoming edges and has outgoing edges to provide and access services. Through these metrics we can calculate and analyze not only the interaction attributes of the components but also the dependencies on each other.

The purpose of this work is to develop a method for finding the Cyclomatic complexity when integrated software system is characterized by various interacting links between the components. Some cases are used to illustrate the computation of Cyclomatic complexity for interacting modules. McCabe’s Cyclomatic complexity model given for a single component is used as the base. The proposed technique logically computes the number of independent paths (Cyclomatic complexity) for an integrated software system where multiple components interact with each other. We have proved the Cyclomatic complexity method for base case, n, and ‘n+1’ components.

Later in the chapter 5, Cyclomatic complexity is also used to count the number of test cases in the context of White-Box testing.