CHAPTER 6
ESTIMATING EXECUTION TIME AND RELIABILITY OF COMPONENT-BASED SOFTWARE

6.1 Introduction
In Software Engineering, reliability plays the role of the foundation stones of the software’s quality. Reliability can be defined as “the probability of failure-free software operation in a specified environment for a specified period of time” [150], or it is defined as “the probability of failure-free operation of a computer program for a specified time in a specified environment” [70]. Musa defined the reliability as “the likelihood of execution without failure for some definite interval of natural units or time” [113]. As per IEEE “Software reliability is defined as the ability of a system or a component to perform its required functions under stated conditions for a specified period of time” [6]. Reliability of COTS Component-Based Software systems can be resulting from reliabilities of individual COTS components which are in the system [151].

We propose a reliability estimation method using reusability feature of the components lying on an execution-history. In this work, have also defined and used the interaction aspects of the components to compute the reliability and the actual execution time of components. Function-points [65] are used as the base metric to estimate the reusability-ratio of the different categories of components. In the absence of function-points we can use lines of code to define the reusability-ratio.

6.2 Basic Definitions and Terminologies
The terms defined here are used to develop the reliability estimation metric. In this section, these terminologies and their definition are given:

a) Component Interaction Graph (CIG): Component Interaction Graph is defined as a tuple consisting of four elements \( \{C_N, E_N, s, t\} \), where \( (C_N, E_N) \) is a directed graph. ‘\( C_N \)’ represents the set of nodes or components and ‘\( E_N \)’ denotes the set of directed edges from one component to the other components. ‘\( s \)’ is the starting node, that can be any component of CIG and ‘\( t \)’ is the last or termination node, where multiple termination nodes are possible. In CIG, starting and termination nodes may change according to the architecture design of the CBS software.

b) Node or Component (\( C_i \)): In Component Interaction Graph, every component is denoted as
Ci, where i = 1, 2…, n. Each component in the CIG contains 5 tuple information, C_i (ETC_i, ReC_i, InvC_i, RRC_i, IRC_i). Where ETC_i is the execution time of the component, ReC_i is the reliability of the component, InvC_i is the invocation counts of the component, RRC_i is the Reliability-ratio of the component, and IRC_i is the Interaction-ratio of the component.

c) Reliability of a Component (ReC_i): Reliability of a Component (ReC_i) represents the probability of failure free execution of component C_i, where i = 1, 2, 3…, n. In this work, we assume that the reliability of components is known to us.

d) Reusability-Ratio of a Component (RRC_i): The reusability ratio of component C_i, is shown as RRC_i, where i = 1, 2, 3…, n. Reusability-ratio of a component depends on its level of reusability. It may be Off-the-shelf, Fully-qualified, Partially-qualified or new, as discussed in Chapter 3.

e) Number of Invocations (InvC_i): Number of Invocations (InvC_i) is the number of times a component is executed or invoked, either to access or to provide services.

f) Execution Time of Component (ETC_i): Every component takes a time period to get executed. This execution time of the component is denoted as ETC_i, the time taken by the component to perform its desired and defined services.

g) Average Execution Time of a Component (ATC_i): Components are invoked to access or to provide services. In a particular execution path, a component can be invoked as many times as its services are required by the system.

Average execution time of a component is defined as:

\[
ATC_i = ETC_i \times InvC_i
\]

where, (ETC_i) is the execution time of a component, and InvC_i is the number of invocations.

h) Execution-history of a Path (EH): Execution-history represents the way components interact with each other. Execution-history is the sequence of execution of components in a particular path from source to termination. If the path has ‘k’ components from start to end and we have ‘n’ paths then, Execution-history is defined as:

\[
EH_1 = C_{11} \rightarrow C_{12} \rightarrow C_{13} \rightarrow \ldots \rightarrow C_{1k}
\]

\[
EH_2 = C_{21} \rightarrow C_{22} \rightarrow C_{23} \rightarrow \ldots \rightarrow C_{2k}
\]

\[
EH_3 = C_{31} \rightarrow C_{32} \rightarrow C_{33} \rightarrow \ldots \rightarrow C_{3k}
\]
\[ E_{Hn} = C_{n1} \rightarrow C_{n2} \rightarrow C_{n3} \rightarrow \ldots \rightarrow C_{nk} \]

Where, \( k = 1, 2, 3\ldots \), t. ‘t’ is the termination node. Here, it is important to note that the number of components in different paths may vary. Therefore, the execution time of each execution-history is different.

**i) Execution Time of an Execution-history (TE_{Hi}):** Execution time of an execution-history is the sum of execution time of all its participating components. We calculate the execution time of an execution-history of a path (TE_{Hi}) as:

\[
TE_{Hi} = \sum_{i=1}^{[t]} (ETC_i \ast I_{nvC_i})
\]

(6.2)

where, \( (ETC_i) \) is the execution time of a component, and \( I_{nvC_i} \) is the number of invocations. Since the execution-history may contain backward and forward loops, we have taken the number of invocations of a component into account. The minimum value of \( I_{nvC_i} \) is 1, since a component lying on a particular path must be invoked at least once.

**j) Probability of an Execution-history (PE_{Hi}):** There are more than one path possibilities in CBS application. The selection probability of a particular execution-history is considered from a number of probable execution-histories. Execution of an execution-history requires the invocation of components residing on that path, from start to end, one after the other, or using backward and forward loops [116].

**k) In-Interaction:** In-Interactions are shown as directed edges pointing towards the calling component. They represent the incoming services to the component or invocations made by some other components. In-Interaction is denoted as \( I_i \), and defined as:

\[
I_i = |\text{Incoming edges to } C_i|
\]

**l) Out-Interaction:** Out-Interactions are the directed edges outgoing from a component. They show the return of request or invocation made by the calling component to some other components. Out-Interaction is denoted as \( O_i \), and defined as:

\[
O_i = |\text{Outgoing edges from } C_i|
\]

**m) Interaction-Ratio of a Component (IRC_i):** The Interaction-ratio of a component can be calculated by taking the ratio of total number of Out-Interactions and the sum of In-Interactions and Out-Interactions of a component. Interaction-Ratio of a component is
defined as:

\[ \text{IRC}_i = \frac{|O_i|}{|O_i| + |I_i|} \]  

(6.3)

n) **Probability of Interaction between two components (PIC\(_{ij}\)):** Suppose, there are two components \(C_i\) and \(C_j\) then the interaction probability defines the probability of selection of Component \(C_j\) for execution, after the component \(C_i\) has been executed. In other words, Interaction probability defines the probability that the component \(C_i\) invoked component \(C_j\) to perform its services. Sum of total probability of interactions invoked by a component to other components is always 1. When we have predefined execution-history, the probability of interaction among components is predetermined which is unity.

Now we define the Component-Integration Graph (CIG) as a 4 tuple set:

\[
\text{CIG} \quad (C_N, E_N, s, t)
\]

- \(C_N\) represents the number of components in the CIG, and
- \(C_i\) represents the particular component ‘i’,
- \(E_N\) represents the number of edges in the CIG,
- ‘s’ shows the starting component of the CIG,
- \(s = C_1\), that is the first component of each path,
- ‘t’ denotes the last component of the CIG,
- \(t = C_n\), that is the termination component of each path.

\[
\text{CIG} \quad (C_N, E_N, s, t)
\]

\(\forall\) Component \(C_i \in \text{CIG}\), we have,

\(C_i (\text{ETC}_i, \text{ReC}_i, I_{in}, \text{RRC}_i, \text{IRC}_i)\)

\(\forall\) Directed edge ‘e’, we have,

\((C_i, C_j, \text{PIC}_{ij})\)

\(I_i = C_i \leftarrow C_j\)

\(O_i = C_i \rightarrow C_j\)
These conventions are followed by every component residing in the Component Interaction Graph, as shown in Figure 6.1.

![Component Interaction Graph](image)

**Figure 6.1 Component Interaction Graph**

### 6.3 Proposed Reliability Metrics

In this reliability estimation method, we have included the individual complexities of components generated due to its interaction with other components. We have also considered the reusability properties of the components during estimation of reliability of the whole software. Using CIG, we define the estimation technique for reliability and actual execution time of component-based software.

The proposed reliability metric has the following considerations:

1. **Reusability-Ratio of Components.**
ii. Reliabilities of individual components.

iii. Actual execution time of an execution-history.

iv. The probability of selection of a particular path’s execution-history.

v. The execution time of each component lying under the selected execution-history.

vi. Actual execution time of the CBS application.

vii. The invocation counts of the components, considering that the components are called by other components to access and provide services.

viii. Reliability of execution-history.

ix. Reliability of CBS application.

6.3.1 Reusability-Ratio.
Component-based software is the mixture of newly developed, reused with modifications and off-the-shelf components. Ratio of reusability depends on the trade-offs between the newly developed components and the reused components.

At Component level, we define the reusability in the terms of Function-points [63]. In absence of function-points we can use lines of code. Reusability-Ratio is the ratio of total number of function-points of a component to the number of reused function-points.

\[
RRC_i = \frac{|\text{Number of reused function-points of the component}|}{|\text{Total function-points of the component}|}
\]

\[
\begin{align*}
RRC_i &= \begin{cases} 
1, & \text{only off-the-shelf components}, \\
\frac{1}{2}, & \text{only reused components}, \\
> \frac{1}{2}, & \text{reused FP > new FP}, \\
< \frac{1}{2}, & \text{low reusability}.
\end{cases}
\end{align*}
\] (6.4)

RRCi = 1, shows that, there is no new function-point involved in the component, and all the components are off-the-shelf. All the function-points have been reused and will reduce the overall cost and development time of the software.

RRCi = \frac{1}{2}, shows that half of the functional units and function-points involved in the components are reused. It counts in the category of partially-qualified components.

RRCi > \frac{1}{2}, represents that the number of reused function-points are greater than the new function-points. Such components come under the category of fully-qualified.

RRCi < \frac{1}{2}, represents that less number of reused function-points are involved in the component.
While calculating the reusability-ratio of components, we consider the following three contexts:

a. When all the components involved in development of CBS application are new,
b. When we have mixture of new and reusable components, and
c. When we have used only reusable components in the CBS application.

a. *When all the components are new:* This is the case, when we have no off-the-shelf or adaptable component that fits into our requirement. Software is made of only newly developed components. In this case the value of $C_{Reused}$ is 0.

\[ T_{CBS} = |C_{New}| \]

b. *When new and reused components are involved:* In this case component-based software is composed of newly developed as well as reused components.

\[ T_{CBS} = |C_{New}| + |C_{Reused}| \]

c. *When only reused components are involved:* This is the case, when we have not used any new component. All the components involved in the CBS development are either off-the-shelf or adaptable. In this case the value of $C_{New}$ is 0.

\[ T_{CBS} = |C_{Reused}| \]

where $T_{CBS}$ is the total number of components required to constitute the component-based software application, $|C_{New}|$ denotes the cardinality that is, number of components which are developed from the scratch and $|C_{Reused}|$ denotes the cardinality of components that are selected from the repository. $C_{Reused}$ is composed of two types of components, off-the-shelf components ($C_{Off-the-shelf}$), and/or adaptable components ($C_{Adaptable}$). Adaptable components can further be divided into two categories: fully-qualified components ($C_{Full-qualified}$) and partially-qualified components ($C_{Part-qualified}$).

Now we define metrics to calculate the ratio and the degree of reusability of components in Component-Based Software.

A. **Fully-Qualified Adaptable Reusability-Ratio at Component Level (RRC_{i-Full-qualified}):**

At component level, Fully-Qualified Adaptable Reusability-Ratio has three different scenarios:

**Scenario 1:** Reusability-Ratio of whole component,

**Scenario 2:** Reusability-Ratio of Reused parts of the Component only, and

**Scenario 3:** Reusability-Ratio of adaptable parts of the Component only.
**Scenario 1: Reusability-Ratio of whole component**

In this scenario, ratio is taken as the total number of function-points of the component. Where, total function-points include reused function-points, off-the-shelf and new function-points. Therefore, Reusability-Ratio for this scenario is defined as:

\[
RRC_{\text{Full-qualified}} = \frac{|\text{Fully-Qualified Function Points of the component}|}{|\text{Total Function Points of the Component}|} \tag{6.5}
\]

**Scenario 2: Reusability-Ratio of Reused parts of the Component only**

In this scenario, ratio is taken in the context of only reused function-points of the component. In this case, the total number of fully-qualified function-points is divided by the total number of reused, that is, adaptable as well as off the shelf function-points. Therefore, is defined as:

\[
RRC_{\text{Full-qualified}} = \frac{|\text{Fully-Qualified Function Points of the component}|}{|\text{Total Reused Function Points of the Component}|} \tag{6.6}
\]

**Scenario 3: Reusability-Ratio of adaptable parts of the Component only**

The ratio is taken in the context of adaptable function-points only. In this case, the total number of fully-qualified function-points is divided by the total number of adaptable, that is, fully-qualified and partially-qualified function-points only. Therefore, Reusability-Ratio is defined as:

\[
RR_{\text{Full-qualified}} = \frac{|\text{Fully-Qualified Function Points of the component}|}{|\text{Total Adaptable Function Points of the component}|} \tag{6.7}
\]

**B. Partially-Qualified Adaptable Reusability-Ratio at Component Level (RRC\textsubscript{i-Part-qualified})**

At component level, Partially-Qualified Adaptable Reusability-Ratio is defined for three different scenarios:

**Scenario 1:** Reusability-Ratio of whole component,

**Scenario 2:** Reusability-Ratio of Reused parts of the Component only, and

**Scenario 3:** Reusability-Ratio of adaptable parts of the Component only.
We calculate the reusability-ratio in the context of a particular component as the total number of partially-qualified reused function-points is divided by the total number of function-points of the components. In the context of whole component Reusability-Ratio is defined as:

\[
RRC_{i-\text{Part-qualified}} = \frac{|\text{Partially-Qualified Function-Points of the component}|}{|\text{Total Function-Points of the Component}|} \tag{6.8}
\]

**Scenario 2: Reusability-Ratio of Reused parts of the Component only:**

In this scenario, ratio is taken in the context of only reused function-points of the component. Here, the total numbers of partially-qualified function-points are divided by the total numbers of reused function-points. Reusability-Ratio is defined as:

\[
RRC_{i-\text{Part-qualified}} = \frac{|\text{Partially-Qualified Function-Points of the component}|}{|\text{Total Reused Function-Points of the Component}|} \tag{6.9}
\]

**Scenario 3: Reusability-Ratio of adaptable parts of the Component only:**

In this scenario, ratio is taken in the context of adaptable function-points only, as the total numbers of fully-qualified function-points are divided by the total numbers of adaptable function-points only. We define reusability-ratio as:

\[
\text{RR}_{\text{Full-qualified}} = \frac{|\text{Partially-Qualified Function-Points of the component}|}{|\text{Total Adaptable Function-Points of the component}|} \tag{6.10}
\]

**C. Off-the-shelf Reusability-Ratio at Component Level (RRC_{i-\text{Off-the-shelf}}):**

At component level, Off-the-shelf Reusability is defined for two scenarios:

**Scenario 1:** Reusability-Ratio of whole component,

**Scenario 2:** Reusability-Ratio of Reused parts of the Component only.

**Scenario 1: Reusability-Ratio of whole component:**

In this scenario, we calculate the reusability-ratio in the context of a particular component. Here, the total number of off-the-shelf reused function-points is divided by the total number of function-points of the components. Reusability-Ratio is defined as:

\[
RRC_{i-\text{Off-the-shelf}} = \frac{|\text{Off-the-shelf Function-Points of the component}|}{|\text{Total Function-Points of the Component}|} \tag{6.11}
\]
Scenario 2: Reusability-Ratio of Reused parts of the Component only:
Ratio is taken in the context of only reused function-points of the component as the total number of off-the-shelf function-points is divided by the total number of reused function-points.

\[ \text{RRC}_{\text{i-Off-the-shelf}} = \frac{|\text{Off-the-shelf Function-Points of the component}|}{|\text{Total Reused Function-Points of the Component}|} \] (6.12)

6.3.2 Reliabilities of Individual Components
One of the major components involved in the reliability of software components is the sequence of inputs and the sequence of execution of components defined in the component-based software architecture. In the CBS, components interact with each other to access and provide services. Any faulty component, when interact with any other component, it cumulates the consequences of faults. This work focuses on integration and interaction issues and reusability ratio of components during the computation of the reliability of component-based software. Our assumption is that the reliability of components is known, and the reliability of each component is denoted by \( R_{\text{c}C_i} \).

6.3.3 Actual Execution Time of an Execution-history (ATE_{Hi})
From equation (6.2) we have, the execution time of an execution-history of a path (TE_{Hi}). Here it is important to note that during the execution of an execution-history, components interact with each other through some interfaces and parameters. Therefore, while calculating the execution time, their interaction ratio plays an important role. We have taken the interaction-ratio as an additional factor which includes the time and overhead of the interaction between two components. Now, we define the Actual Execution Time of an Execution-history as:

\[ \text{ATE}_{Hi} = \sum_{i=1}^{[t]} (\text{ETC}_i \ast \text{I}_{nvi}C_i + \text{IRC}_i) \] (6.13)

where, ETC\(_i\) is the execution time of the component, I\(_{nvi}C_i\) is the count of invocations and IRC\(_i\) is the interaction-ratio of components ‘i and ‘j’. We can compute Interaction-ratio from Equation (6.3).

6.3.4 Actual Execution Time of CBS (ATE_{CBS})
From equation (6.13), we have the Actual Execution time of an execution-history among all the execution histories. In a CBS application, we may have more than one path. Therefore, the
Average Execution path is defined as the sum of the execution of all the execution histories, that is:

$$ATE_{CBS} = \sum_{Hi=1}^{n} ATE_{Hi} \quad (6.14)$$

where, $Hi = 1$ to number of Execution-histories ‘$n$’.

6.3.5 Reliability of an Execution-history ($RH_i$)

On a particular path, we have more than one component, and every component has its reliability. Components cooperate with each other; hence there is some degree of interaction among them. In addition, when components interact more than once, their reusability is also an important issue regarding their reliability. During the computation of reliability of an execution-history, we introduce these two factors, that is, Interaction-ratio and the Reusability-ratio. Reliability of an Execution-history of a particular path is calculated as:

$$RH_i = \prod_{j=1}^{|t_i|} \prod_{i=1}^{j} \{ PIC_{ij} \times (R_{eC_i} + (1 - R_{eC_i} \times RRC_{ij} \times IRC_i)) \} \quad (6.15)$$

where PIC$_{ij}$ is the probability of invocation of component $C_i$, $R_{eC_i}$ is the reliability of the component $C_i$, $RRC_{ij}$ is the reusability-ratio of the component $C_i$, and IRC$_i$ is the interaction-ratio of the components ‘$i$’, which may be the ‘New’, ‘Adaptable’, or ‘Off-the-shelf’. Without the loss of generality, we assume that the reusability-ratio of new components is 0.01.

6.3.6 Reliability of Component-Based Software ($R_{CBS}$)

In the assessment of the reliability of component-based software it can be concluded that the reliability of CBS is a function of reliabilities of individual component [150, 152, 153].

When we have more than one execution-history in a CBS application, each history has its own reliability estimated from the starting component to the ending component. We define the reliability of the Component-Based Software as the probability of each path and the reliability of that path’s execution-history, that is:

$$R_{CBS} = \sum_{Hi=1}^{n} PE_{Hi} \times RH_i \quad (6.16)$$

where, $R_{CBS}$ is the reliability of the component-based software, and $PE_{Hi}$ is the probability of the execution-history of a path, $RRC_i$ is the Reusability-ratio of the component.
6.4 Case Studies

To illustrate the proposed method we use a case study defined in the Figure 6.1. This case study has five components. Component Interaction Graph (CIG) of the case study has shown in Figure 6.1. To elaborate the role of reusability and interaction issues, we have used all the three types of components in CIG. We have assumed that CIG defined in Figure 6.1, composed of new, reused adaptable (partially-qualified and fully-qualified both) and off-the-shelf components. Where component C3 is the newly developed component, hence its reusability-ratio is 0.01. Components C1 and C5 are partially-qualified components having reusability of 0.18 and 0.2 respectively. Component C2 is fully-qualified having reusability-ratio 0.8 and component C4 is off-the-shelf component having reusability-ratio 1. When a component invoked some other component through an out-interaction edge, the calling component will get services in return. We show this type of requests through Out-interaction edges, and the services in return are shown as In-interaction edges, as in Figure 6.1. All the interaction edges among components are either in-interaction or out-interaction edges. These interaction edges play an important role in the computation of interaction-ratio of these interacting components.

From the given CIG in Figure 6.1, it is observed that there are three paths of execution from the starting component C1 to the ending component C2.

Now based on Execution-histories of these paths, we have three cases:

Case 1: H1 → C1 → C3 → C5
Case 2: H2 → C1 → C4 → C5
Case 3: H3 → C1 → C2 → C4 → C5.

Each component, its interaction edge and execution-history have corresponding values for parameters execution time of C_i, reliability of C_i, number of invocations of C_i for execution, reusability-ratio of C_i. On the basis of these parameters, we calculate the ATC_i of component ‘i’. These values and computations are shown in Table 6.1. We have taken these calculations considering the execution-history of the CBS application.

Case 1: Values and computations of Execution-history H1:

<table>
<thead>
<tr>
<th>C_i</th>
<th>ETC_i</th>
<th>R_iC_i</th>
<th>I_nvC_i</th>
<th>RRC_i</th>
<th>IRC_i</th>
<th>ATC_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
<td>4</td>
<td>0.6</td>
<td>2</td>
<td>0.18</td>
<td>0.6</td>
<td>8</td>
</tr>
<tr>
<td>C_3</td>
<td>3</td>
<td>0.7</td>
<td>2</td>
<td>0.01</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>C_5</td>
<td>6</td>
<td>0.9</td>
<td>2</td>
<td>0.2</td>
<td>0.33</td>
<td>12</td>
</tr>
</tbody>
</table>
Table 6.2 Probability of Components Interaction in Path Execution-History \(H_1\)

<table>
<thead>
<tr>
<th>(C_i \Rightarrow C_j)</th>
<th>(PIC_{ij})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_1 \Rightarrow C_1)</td>
<td>1</td>
</tr>
<tr>
<td>(C_1 \Rightarrow C_3)</td>
<td>0.33</td>
</tr>
<tr>
<td>(C_3 \Rightarrow C_5)</td>
<td>1</td>
</tr>
</tbody>
</table>

**Reliability Computation of Execution-history \(H_1\)**

First we calculate the reliability growth (RG\(i\)) in each component due to the Reusability-ratio.

Reliability Growth in component \(C_1\):

\[
RG_{C_1} = (R_e C_i + (1 - R_e C_i RRC_i \times IRC_i))
\]

\[
= (0.6 + (1 - 0.6 \times 0.18 \times 0.6))
\]

\[
RG_{C_1} = 0.65
\]

Reliability Growth in component \(C_2\):

\[
RG_{C_2} = (R_e C_i + (1 - R_e C_i RRC_i \times IRC_i))
\]

\[
= (0.8 + (1 - 0.8 \times 0.8 \times 0.5))
\]

\[
RG_{C_2} = 0.89
\]

Reliability Growth in component \(C_3\):

\[
RG_{C_3} = (R_e C_i + (1 - R_e C_i RRC_i \times IRC_i))
\]

\[
= (0.7 + (1 - 0.7 \times 0.01 \times 0.01))
\]

\[
RG_{C_3} = 0.7
\]

Reliability Growth in component \(C_4\):

\[
RG_{C_4} = (R_e C_i + (1 - R_e C_i RRC_i \times IRC_i))
\]

\[
= (0.7 + (1 - 0.7 \times 0.5))
\]

\[
RG_{C_4} = 0.86
\]

Reliability Growth in component \(C_5\):

\[
RG_{C_5} = (R_e C_i + (1 - R_e C_i RRC_i \times IRC_i))
\]

\[
= (0.9 + (1 - 0.9 \times 0.33 \times 0.33))
\]

\[
RG_{C_5} = 0.91
\]

Applying these values on Equation (6.15) we calculate the Reliability of Execution-history \(H_1\)

\[
EH_1 = (0.65 \times 1) + (0.65 \times 0.7 \times 0.33) + (0.65 \times 0.7 \times 0.33) \times 0.91 \times 1
\]

\[
EH_1 = 0.94
\]

Using Sherif-Yacoub’s method:

\[
EH_1 = (0.6 \times 1 \times 1) + ((0.6 \times 1 \times 1) \times 0.7 \times 0.33) + (((0.6 \times 1 \times 1) \times 0.7 \times 0.33)) \times 0.9 \times 1
\]
Case 2: Values and computations of Execution-history \( H_2 \):

Table 6.3 Inputs of Path Execution-History \( H_2 \)

<table>
<thead>
<tr>
<th>( C_i )</th>
<th>( \text{ETC}_i )</th>
<th>( R_e C_i )</th>
<th>( I_{nv} C_i )</th>
<th>( \text{RRC}_i )</th>
<th>( \text{IRC}_i )</th>
<th>( \text{ATC}_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>4</td>
<td>0.6</td>
<td>2</td>
<td>0.18</td>
<td>0.6</td>
<td>8</td>
</tr>
<tr>
<td>( C_4 )</td>
<td>8</td>
<td>0.7</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
<td>16</td>
</tr>
<tr>
<td>( C_5 )</td>
<td>6</td>
<td>0.9</td>
<td>2</td>
<td>0.2</td>
<td>0.33</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 6.4 Probability of Components Interaction of Path Execution-History \( H_2 \)

<table>
<thead>
<tr>
<th>( C_i \rightarrow C_j )</th>
<th>( \text{PIC}_{ij} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 \rightarrow C_1 )</td>
<td>1</td>
</tr>
<tr>
<td>( C_1 \rightarrow C_4 )</td>
<td>0.33</td>
</tr>
<tr>
<td>( C_4 \rightarrow C_5 )</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Applying the values of Table 6.3 and Table 6.4 on Equation (6.15) we calculate the Reliability of Execution-history \( H_2 \)

\[ \begin{align*}
\text{EH}_2 &= (0.65 \times 1) + (0.65 \times 0.8 \times 0.33) + ((0.65 \times 0.8 \times 0.33) \times 0.91 \times 0.5) \\
\text{EH}_2 &= 0.90 
\end{align*} \]

Using Sherif-Yacoub’s method:

\[ \begin{align*}
\text{EH}_2 &= (0.6 \times 1 \times 1) + (0.6 \times 0.7 \times 0.33) + ((0.6 \times 0.7 \times 0.33) \times 0.9 \times 0.5) \\
\text{EH}_2 &= 0.80 
\end{align*} \]

Case 3: Values and computations of Execution-history \( H_3 \):

Table 6.5 Inputs of Path Execution-history \( H_3 \)

<table>
<thead>
<tr>
<th>( C_i )</th>
<th>( \text{ETC}_i )</th>
<th>( \text{ReC}_i )</th>
<th>( I_{nv} C_i )</th>
<th>( \text{RRC}_i )</th>
<th>( \text{IRC}_i )</th>
<th>( \text{ATC}_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>4</td>
<td>0.6</td>
<td>2</td>
<td>0.18</td>
<td>0.6</td>
<td>8</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>6</td>
<td>0.8</td>
<td>2</td>
<td>0.8</td>
<td>0.5</td>
<td>12</td>
</tr>
<tr>
<td>( C_4 )</td>
<td>8</td>
<td>0.7</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
<td>16</td>
</tr>
<tr>
<td>( C_5 )</td>
<td>6</td>
<td>0.9</td>
<td>2</td>
<td>0.2</td>
<td>0.33</td>
<td>12</td>
</tr>
</tbody>
</table>
Applying these values of Table 6.5 and Table 6.6 on Equation (6.15) we calculate the Reliability of Execution-history $H_1$:

$$EH_3 = (0.65 \times 1) + (0.65 \times 0.8 \times 0.33) + ((0.65 \times 0.8 \times 0.33) \times 0.89 \times 0.5) + ((0.65 \times 0.8 \times 0.33) \times 0.89 \times 0.5) \times 0.91 \times 0.5$$

$$EH_3 = 0.94$$

Using Sherif-Yacoub’s method:

$$EH_3 = (0.6 \times 1 \times 1) + (0.6 \times 0.8 \times 0.33) + ((0.6 \times 0.8 \times 0.33) \times 0.7 \times 0.5) + ((0.65 \times 0.8 \times 0.33) \times 0.7 \times 0.5) \times 0.9 \times 1$$

$$EH_3 = 0.87$$

**Actual execution time and reliabilities of execution-histories:**

Based on these values of corresponding execution histories, we can compute the execution time, probability of execution, actual execution time and the reliability of each execution-history of the CBS application. These calculations are shown in Table 6.7.

Table 6.7 Execution Time and Reliabilities of Execution-Histories

<table>
<thead>
<tr>
<th>$H_i$</th>
<th>$TE_{HI}$</th>
<th>$PE_{HI}$</th>
<th>$ATE_{HI}$</th>
<th>$R_{HI}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_1$</td>
<td>26</td>
<td>0.33</td>
<td>27.93</td>
<td>0.94</td>
</tr>
<tr>
<td>$H_2$</td>
<td>36</td>
<td>0.33</td>
<td>37.43</td>
<td>0.90</td>
</tr>
<tr>
<td>$H_3$</td>
<td>48</td>
<td>0.33</td>
<td>49.93</td>
<td>0.94</td>
</tr>
</tbody>
</table>

**Actual execution time and reliability of CBS application:**

Now we can calculate the actual execution time and the reliability of the CBS application using the values of execution histories through equation (6.14) and equation (6.16).
Table 6.8 Actual Execution Time and Reliability of CBS Application

<table>
<thead>
<tr>
<th>Application</th>
<th>$\text{ATE}_{\text{CBS}}$</th>
<th>$R_{\text{CBS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBS</td>
<td>115.29</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Figure 6.2 Execution Times, Probability of Execution and Actual Execution Times of Execution-Histories

Comparative Reliability analysis of two methods:

Table 6.9 Comparative Reliabilities of each Path through both the Methods

<table>
<thead>
<tr>
<th>$H_i$</th>
<th>$R_{Hi}$ (Sherif-Yacoub)</th>
<th>$R_{Hi}$ (Proposed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_1$</td>
<td>0.87</td>
<td>0.94</td>
</tr>
<tr>
<td>$H_2$</td>
<td>0.80</td>
<td>0.90</td>
</tr>
<tr>
<td>$H_3$</td>
<td>0.87</td>
<td>0.94</td>
</tr>
</tbody>
</table>
This chapter proposes an estimation method for the actual execution time and the reliability for the execution-history of a path. Further, this work defines the estimation method of the execution time and the reliability for the component-based application. We have introduced an interaction model named Component Interaction Graph (CIG). The component interaction graph contains not only the interaction information of components but holds some useful attributes of each coordinating component. The CIG is used to base the execution time and the reliability estimations. These computations are based on the fundamental properties of component-based software, that is, the feature of reusability and the nature of interaction. We have defined metrics to explore properties as reusability-ratios and interaction-ratios. The results conclude that the reusability-ratio of component plays vital role in the reliability of execution-history and ultimately in the reliability of the CBS application. The obtained results are liable to conclude that the increase in the reusability of components help for growth in the reliability of the application because they are pre-tested and qualified components as compared to those that are not.
to new components. Since interaction among components through well-defined interfaces
contribute in execution time, we have taken interaction-ratio in account while calculating the
time for execution-histories and CBS application.

Though, there is a nominal growth in the execution time due to these interactions, but the
Interaction-ratio gives the complete privilege for execution of the component. These ratio
metrics are helpful not only in the computations of the execution and reliabilities of the CBS
applications but can be used as an attribute to store it in repository along with the component
for future reuse.