CHAPTER 5

PROPOSED ADAPTABILITY METRICS

Software maintenance is one of the crucial activities in the software development, which requires 60% of the total efforts expended towards other activities. Adaptability is an important quality attribute that plays a vital role in the maintenance. Providing adaptability feature at the architecture level will reduce the effort expended towards the maintenance. Software architecture for adaptive systems should be flexible to allow the components to change their behavior. The separation of concerns (SOC) principle stated in aspect-oriented approach is used for implementing adaptability in a software system. An adaptable middleware framework proposed in this work uses the SOC principles stated in aspect-oriented approach to provide a solution for adaptability at the architecture level. This framework modularizes the dynamic changes representing them as aspects. The adaptable efficiency of this framework is to be measured to evaluate the framework.

Evaluating the adaptability at the architectural level is performed using the following approaches:

- developing adaptability scenario profile for the architecture based on the system adaptability goals
- performing an impact analysis under the scenario profile
- applying the metric and calculating the value of adaptability degree (Tarvainen 2008).
Among these approaches, the impact analysis is used to define the metric for evaluating the adaptability efficiency of the adaptable middleware framework. The adaptable efficiency of the framework is defined by measuring the ability of the component to adapt the dynamic changes in their functions. Measuring the adaptability of the component is realized by its structure such as provider and required interfaces. It also includes evaluating the complexity of the classes defined within the component. The complexity of a class is evaluated with respect to the number of public methods in a class, the number of external services requested from other classes and the number of attributes. It is a difficult task to consider all these factors. Also, the use of design metrics to test the performance issues of the software system was proposed in (Yogesh and Anju 2012), which initiates the use of design metric for measuring the adaptability efficiency of the system. Hence, in the present work for measuring the adaptability of the software at the architecture level, the class complexity evaluation strategy is refined by measuring the impact of the changes on the component functions. Here, the change impact is specified using the number of classes/components accessing the service associated with the change, which is termed as coupling between the classes/components. It implies that the coupling metrics are appropriate for measuring the adaptability.

The authors of (Poshyvanyk and Marcus 2006) have considered coupling as a primary property that influences the maintenance task of a software system. This principle induces the coupling as a measure to evaluate the adaptability of a system. Hence, in this work a conceptual level coupling metrics have been proposed in a new dimension for evaluating the adaptability of object-oriented system and adaptable aspect-oriented framework. In general, coupling metric is used to measure the level of interdependency between modules/components/classes in a system (Rachel et al 2010). This work redefines the coupling metric stated by (Chidamber and
Kemerer (1991) and introduces the new metric conceptual binding between classes (CBC) for measuring the adaptability efficiency of an object-oriented system. The conceptual binding between classes (CBC) is measured as the sum of the number of methods in each class associated with other classes in a system and the adaptability of a system is inversely proportional to this measure. Based on the dynamic quantification of the system behavior feature provided by the aspect-oriented approach (Filman 2000) and aspect-oriented design principles stated in (Wampler 2007), the coupling principle is used to evaluate the adaptability of the aspect-oriented framework. Based on these principles, a new metric namely the conceptual binding between aspect and classes (CBAC) is introduced to measure the adaptability efficiency of a system, which realizes the changes using aspects. CBAC represents the total number of classes/components in a system associated with the aspects that are representing the dynamic changes. Hence, the CBAC metric is used to specify the adaptability of the proposed aspect-oriented adaptable framework. The CBC and CBAC are the metrics to evaluate the adaptability of a system at design level. The adaptability of a system can also be evaluated at the execution level through measuring the time taken for reconfiguring the components and the time taken for executing the functions of the components along with the changes.

5.1 CONCEPTUAL BINDING BETWEEN CLASSES METRIC

5.1.1 Metric Formulation

The coupling between classes is viewed in a new dimension as a conceptual coupling of a class with other classes in a system. The conceptual Coupling Of a Class (COC) is measured as the degree to which the methods of a class are semantically associated with other classes. The Conceptual Binding between the Classes (CBC) of a software system is measured as the
average of COC in a system. Hence, the adaptability efficiency of an object-oriented system is specified with CBC value of a system.

Let the set of classes be denoted as \( C = \{ c_1, c_2, \ldots, c_n \} \), where `n` is the number of classes in a software system and the set of methods in each class \( c_i \in C \) be represented as \( M(c_i) = \{ m_{i1}, m_{i2}, \ldots, m_{ik} \} \), where \( 1 \leq i \leq n \), \( k \) is the number of methods in a class \( c_i \). The number of methods in a class \( c_i \) is specified as the cardinality of a set \( M(c_i) \), which is denoted as \(|M(c_i)|\).

The coupling of the methods \( M(c_i) \) in a class \( c_i \in C \) with other classes \( c_j \in C(j\neq i) \) is the sum of the coupling of each method \( m_{ik} \) in \( c_i \) with \( c_j \) \((j\neq i)\), which is represented as \( \text{COC}(c_i) \) and is expressed in Equation (5.1). The coupling of each method in a class with other classes is defined as the total number of the usage and re-usage or sharing of a method by other classes.

\[
\text{COC}(c_i) = \sum_{k=1}^{M(c_i)} \text{Coupling}((m_{ik}(c_i), c_j)) \quad (5.1)
\]

The conceptual binding between classes (CBC) is the average of coupling of each class with other classes of the system and it is expressed as shown in Equation (5.2).

\[
\text{CBC} = \frac{\sum_{i=1}^{n} \text{COC}(c_i)}{|C|} \quad (5.2)
\]

When the value of CBC is high, it implies that the classes in the system are tightly coupled. This specifies that it is difficult to make the changes in the system. Hence it is derived that the Adaptability Of the System (AOS) is increased by reducing the coupling between the classes, which is expressed as in Equation (5.3).

\[
\text{AOS} = 1 / \text{CBC} \quad (5.3)
\]
The data specifying the dependency between classes is represented using Class Dependency Matrix (CDM) as shown in Table 5.1. In this matrix, each row represents the association of a class $c_i$ with the methods of the classes $c_j (j \neq i, 1 \leq i \leq n \text{ and } 1 \leq j \leq n)$; each column represents the association of the method $m_{ik}$ of the class $c_i$ with other classes $c_j (j \neq i)$; and each cell represents the number of usage and re-usage or sharing of the method $m_{ik}$ of a class $c_i$ in a class $c_j (j \neq i)$.

$$\text{CDM}(i, j) \quad (1 \leq i \leq n \text{ and } 1 \leq j \leq h, \text{ where } h = \sum_{i=1}^{n} |M(c_i)|)$$ takes value ‘0’ when the method of class $c_i$ is associated with methods of the same class; otherwise it takes value ‘t’ where $0 \leq t \leq \text{ number of methods in } c_j \text{ associated with } c_i$. Using CDM, COC($c_i$) is defined as the sum of $\text{CDM}(i, j)$ associated with $m_{ik}(c_i)$.

<table>
<thead>
<tr>
<th></th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>...</th>
<th>$c_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>m$_{11}$</td>
<td>m$_{12}$</td>
<td>..</td>
<td>m$_{1p}$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>m$_{21}$</td>
<td>m$_{22}$</td>
<td>..</td>
<td>m$_{2q}$</td>
</tr>
<tr>
<td>..</td>
<td>t</td>
<td>t</td>
<td>..</td>
<td>t</td>
</tr>
<tr>
<td>$c_n$</td>
<td>t</td>
<td>t</td>
<td>..</td>
<td>t</td>
</tr>
</tbody>
</table>

5.1.2 Demonstration of Conceptual Binding Between Classes Metric

The banking transaction system discussed in the Section 3.5 is designed with the classes specified in the class set $C$.

$C = \{\text{Authentication, SB-Acc Transaction, Current-Acc Transaction, Account, UI Management}\}$. 
The number of classes in a set C is denoted as \(|C|\) that takes the value 5. The methods defined in a class ‘c’ are denoted as \(M(c)\). The methods, which are defined in the classes of a system, are as follows:

\[ M(\text{Authentication}) = \{\text{userVerification, accountVerification}\} \]

**|\(M(\text{Authentication})|\)** denotes the number of methods in Authentication class, which is equal to 2.

\[ M(\text{SB-Acc Transaction}) = \{\text{deposit, withdrawal, fundTransfer, billPayment, EMIPayment, principalPayment, viewStatus, openAccount, closeAccount}\} \]

The number of methods in this class is represented as \(|M(\text{SB-Acc Transaction})|\), and the value is 9.

The class Current-Acc Transaction is defined with the methods as specified in the SB-Acc Transaction class.

\[ M(\text{Account}) = \{\text{getAccount, setAccount}\} \]

**|\(M(\text{Account})|\)** represents the number of methods in Account class that is equal to 2.

\[ M(\text{UIManagement}) = \{\text{displayContentController,displayFormatController}\} \]

The number of methods in UI Management class is represented as \(|M(\text{UI Management})|\), which takes the value 2.

According to the class design specification as shown in Figure 4.10, for performing all the transaction operations specified in the SB-Acc Transaction and Current-Acc Transaction classes require validating user and account details and also have to access the account information using ‘set and get’ operations defined in Account class. Similarly, displaying the result of these transactions needs to use display content and display format controls specified in the UI Management class. These details are captured using Class Dependency Matrix (CDC) of a banking transaction system.
Table 5.2 Class dependency matrix for banking transaction system

<table>
<thead>
<tr>
<th></th>
<th>Authentication</th>
<th>SB-Acc Transaction</th>
<th>Current-Acc Transaction</th>
<th>Account</th>
<th>UI Management</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aV</td>
<td>uV</td>
<td>d</td>
<td>w</td>
<td>..</td>
</tr>
<tr>
<td>Authentication</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>SB-Acc Transaction</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>Current-Acc Transaction</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>..</td>
</tr>
<tr>
<td>Account</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>UI Management</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
</tbody>
</table>

**Notation Description:** aV represents accountVerification(); uV represents userVerification(); d denotes deposit(); w denotes withdrawal(); vS represents viewStatus(); gA represents getAccount(); sA denotes setAccount(); dCC represents displayContentController; dFC denotes displayFormatController.
The efficiency of adapting the changes specified in the authentication, account transaction and user interface management scenario as described in the Section 3.5 on an object-oriented banking system is measured using CBC. The impact of incorporating changes in the authentication scenario is represented using the coupling between the methods in Authentication class with other classes as specified in Equation (5.1). The conceptual coupling of each method in each class is represented in the Class Dependency Matrix (CDM) of a banking system as shown in Table 5.2. In that table, COC(Authentication) is represented as association of Authentication with SB-Acc Transaction and Current-Acc Transaction classes and it is measured as COC(Authentication) that takes value 4; CBC(Authentication), its values is \(36/5 = 7.2\).

Similarly, the changes in transaction scenario are measured by determining the association of SB-Acc Transaction and Current-Acc Transaction classes with other classes. Through CDM, the association value for SB-Acc Transaction and Current-Acc Transaction classes is identified as 9 each. Hence, the value of COC(SB-Acc Transaction) is 9; the value of COC(Current-Acc Transaction) is 9 and the corresponding CBC values are: CBC(SB-Acc Transaction) is \(9/5(1.8)\); CBC(Current-Acc Transaction) is \(9/5(1.8)\).

The impact of the changes in the user interface management is determined as the association of methods in UI Management class with other classes, which is specified as 4 using CDM Table. The value of CBC(UI Management) is \(4/5(0.8)\).

The adaptability of the system to incorporate each type of the changes is represented as \(1/CBC\). For incorporating the changes in the various scenarios, the coupling of the corresponding classes are measured as stated above and the generated results have been consolidated in Table 5.3. The
inference of the generated coupling measures is stated as ,if the coupling value of a class is high, which implies that the efficiency of adapting the changes associated with that class is low.

Table 5.3 Adaptability values for banking system using CBC

<table>
<thead>
<tr>
<th>Changes</th>
<th>COC</th>
<th>CBC</th>
<th>AOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication Changes</td>
<td>36</td>
<td>7.2</td>
<td>0.14</td>
</tr>
<tr>
<td>Transaction Changes</td>
<td>18</td>
<td>3.6</td>
<td>0.28</td>
</tr>
<tr>
<td>User Interface Management Changes</td>
<td>4</td>
<td>0.8</td>
<td>1.25</td>
</tr>
</tbody>
</table>

5.2 CONCEPTUAL BINDING BETWEEN ASPECT AND CLASSES METRIC

5.2.1 Metric Formulation

Using the aspect-oriented paradigm, the coupling between classes can be reduced by separating the cross-cutting concerns, which improves the adaptability of the system. The work described in this paper proposes aspect as a unit to represent the dynamic changes in order to make the system adaptable to those dynamic changes. Since aspect-oriented approach is an extension of object-oriented approach, object-oriented metrics can be used to measure aspect-oriented systems (Zakaria and Hosny 2003). A new dimension for measuring the adaptability of the system through redefining COC metric has been proposed here. The adaptability of the system is measured with the distribution of aspects among the classes. Here, this adaptability measure specifies the number of classes that are associated (affected) with the dynamic changes in the requirements, which is referred to as the conceptual binding between aspect and classes (CBAC). Initially, the association between the methods of a class with dynamic changes in the
requirements is to be measured, which is referred as the conceptual binding between aspect and methods in a class (CBAM) for determining CBAC. The association between the requirements and the methods in a class is used to derive the association between the aspect and the methods.

The association between the requirements and the classes is represented using Requirements Class Association Matrix (RCAM) as shown in Table 5.4. RCAM (p,q), (1 ≤ p ≤ n and 1 ≤ q ≤ h, where h=∑_{i=1}^{n}|M(c_i)|) takes value 1, if Requirement#p is defined in the method of a set M(c_i); otherwise it takes value 0. In Equation (5.4) the expression of CBAM using RCAM is shown with denoting the aspect defined for representing dynamic changes in the requirement as ‘a’, associating aspect ‘a’ with the Requirement#p.

\[
\text{CBAM} (a, M(c_i)) = \sum_{k=1}^{\mid M(c_i) \mid} w(a, m_{i,k}(c_i)), \quad \text{(5.4)}
\]

where \(w(a, m_{i,k}(c_i)) = \text{RCAM} (p,q)\).

### Table 5.4 Requirements class association matrix

<table>
<thead>
<tr>
<th></th>
<th>c_1</th>
<th>c_2</th>
<th>…..</th>
<th>c_1</th>
<th>c_2</th>
<th>…..</th>
<th>c_1</th>
<th>c_2</th>
<th>…..</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m_{11}</td>
<td>m_{12}</td>
<td>…..</td>
<td>m_{1r}</td>
<td>m_{21}</td>
<td>m_{22}</td>
<td>…..</td>
<td>m_{2s}</td>
<td>m_{ni}</td>
</tr>
<tr>
<td>Requirement#1</td>
<td>0/1</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
</tr>
<tr>
<td>Requirement#2</td>
<td>0/1</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
</tr>
<tr>
<td>Requirement#n</td>
<td>0/1</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
<td>…..</td>
<td>0/1</td>
</tr>
</tbody>
</table>

CBAC is specified as the average of CBAM of each class in the system, which is expressed in Equation (5.5). The set of classes in the system is represented as ‘C’. m_{ij} ∈ M(c_i) and an aspect implementing the dynamic changes is denoted as ‘a’ in the equation stated below.
\[
\text{CBAC} (a, C) = \frac{\sum_{i=1}^{n} (\text{CBAM} (a, M(c_i)))}{|A|},
\]

(5.5)

Let the set of ‘n’ aspects be denoted as \(A = \{a_1, a_2, \ldots, a_n\}\) and the distribution of these ‘n’ aspects over the system (DOF(A)) is defined as the average of CBAC(a, C) that is shown in Equation (5.6).

\[
\text{DOF} (A) = \frac{\sum_{i=1}^{n} (\text{CBAC}(a_i, C))}{|A|}
\]

(5.6)

From the above discussion, it is clear that the adaptability efficiency of an aspect-oriented system or AOS is measured with the number of functions into which the aspects are to be weaved, which is expressed in the Equation (5.7). It leads to derive a lemma stated below:

\[
\text{AOS}_{\text{dynamic requirement changes}} = \text{DOF(Aspects)}
\]

(5.7)

**Lemma:** The AOS is increased with increasing the CBAC factor.

The proposed adaptable middleware model provides the facility to adapt to the changes by defining those changes as aspects and weaving them into the corresponding methods in the classes. While changing the requirements, it is sufficient to define an aspect through which the changes will get reflected on all the functions associated with that change. This implies that all classes associated with a dynamic change can adapt it through a single weaving without altering the existing structure of the system, which automatically reduces the effort to be expended on adapting to the changes. Hence, if the number of classes associated with the aspects is more, then the possibility of adapting the changes implemented in those aspects will be high.
This discussion concludes that increasing the conceptual coupling between aspect and classes (CBAC) will increase the adaptability of the system.

5.2.2 Metric Evaluation

The validity of the proposed metric is to be proved by assessing it towards the general properties of the metric. In this work the properties proposed to validate the object-oriented metrics are used for validating the proposed CBAC metric. Also the CBAC metric was used to measure the adaptability efficiency of the banking transaction system. The work done on metric validation and its applicability for measuring the adaptability efficiency is described in the following sections. The Conceptual binding between the aspect and classes (CBAC) is evaluated based on the properties stated in the (Chidamber and Kemerer 1991). The property list includes Non-coarseness, Non-uniqueness, Permutation Significance, Functional Implementation importance, Monotonicity, Non-equivalence interaction and Interaction complexity.

**Non-Coarseness:** The CBAC metric of two different aspects are not same. 

\( \text{CBAC}(a_1, C) \neq \text{CBAC}(a_2, C) \), where \( a_1, a_2 \) are two different aspects. Hence CBAC satisfies this property.

**Non-Uniqueness:** CBAC possesses non-uniqueness based on the conceptual closeness of the aspects, 

\( \text{CBAC}(a_1, C) = \text{CBAC}(a_2, C) \).

**Permutation Significance:** If the aspect \( a_1 \) is the permutation of \( a_2 \), then the coupling between the aspect \( a_1 \) with the classes and the coupling between the aspect \( a_2 \) with the classes will take the same value - 

\( \text{CBAC} (a_1, C) = \text{CBAC} (a_2, C) \). Hence it does not satisfy this property.
**Function Implementation Importance:** If the functions of aspects $a_1$ and $a_2$ are similar and have different implementations, then $\text{CBAC}(a_1, C) \neq \text{CBAC}(a_2, C)$. This implies that coupling is determined based on the implementation of the aspects, and not on the type of the operation.

**Monotonicity:** Let $a_1$ and $a_2$ are the aspects and $\text{CBAC}(a_1, C) = n_1$, $\text{CBAC}(a_2, C) = n_2$. $\text{CBAC}((a_1+a_2), C) = n_1+n_2-\alpha$, where $\alpha$ is the number of reduction in the coupling after combining $a_1$ and $a_2$. $n_1-\alpha \geq 0$ and $n_2-\alpha \geq 0$. Hence, $\text{CBAC}(a_1+a_2) \geq \text{CBAC}(a_1)$ and $\text{CBAC}(a_1+a_2) \geq \text{CBAC}(a_2)$, which implies CBAC metric satisfies monotonicity.

**Non-equivalence interaction and Interaction complexity:** Let $a_1$, $a_2$ and $a_3$ are the aspects and $\text{CBAC}(a_1, C) = n_1$, $\text{CBAC}(a_2, C) = n_2$. $\text{CBAC}(a_3, C) = n_3$. $\text{CBAC}((a_1+a_3), C) = n_1+n_3-\alpha$, where $\alpha$ is the number of reduction in the coupling after combining $a_1$ and $a_3$. $n_1-\alpha \geq 0$ and $n_3-\alpha \geq 0$.

$\text{CBAC}((a_2+a_3), C) = n_2+n_3-\lambda$, where $\lambda$ is the number of reduction in the coupling after combining $a_2$ and $a_3$. $n_2-\lambda \geq 0$ and $n_3-\lambda \geq 0$. Since $\alpha, \lambda$ are not equal, $\text{CBAC}((a_1+a_3), C)$ is not equal to $\text{CBAC}((a_2+a_3), C)$. It implies that the coupling between the aspects $(a_1+a_3)$ with the system is not equal to the coupling between the aspects $(a_2+a_3)$ with the system.

According to the above discussion, it is concluded that the proposed CBAC metric satisfies all the properties except permutation significant and non-equivalence of interactions. This shows the validity of the metric.
5.2.3 Demonstration of Conceptual Binding Aspect and Classes Metric

In this work, the conceptual binding between the set of aspects “A” with the classes $c_i \in C$ (CBAC (A,C)) represents the adaptability of the system that is designed using the aspect-oriented solution proposed in an adaptable middleware framework. The CBAC metric is also considered as the measure which shows the association between the requirements that are to be changed dynamically and the methods associated with those requirements. Here the proposed adaptability metric is demonstrated to show the adaptability efficiency of the Banking Transaction and Sales Processing systems.

**Case Study: 1 – Banking Transaction System**

According to the banking transaction system described in Section 3.5, the following are considered as the requirements that are to be realized in the software.

- RequirementBT1: Perform the user authentication.
- RequirementBT2: Perform the account validation.
- RequirementBT3: Allow the user to perform deposit.
- RequirementBT4: Perform withdrawal transaction.
- RequirementBT5: Perform fund transfer transaction.
- RequirementBT6: Perform bill payment.
- RequirementBT7: Allow the user to open or close the account.
- RequirementBT8: Perform account status view.
- RequirementBT9: Maintain the account details.
- RequirementBT10: Allow the customer to access the service through desktop, laptop and mobile devices.

These requirements are realized in the system through the methods defined in the classes as shown in Figure 4.4. The detailed descriptions regarding the class design is given below:

The class set \( C \) is defined as \( C = \{ \text{Authentication, SB-Acc Transaction, Current-Acc Transaction, Account, UI Management} \} \)

- User authentication and account validation processes are implemented in the respective userVerification and accountVerification methods of the Authentication class.

- Deposit, withdrawal, fund transfer, bill payment, viewing the account status, account opening and closing the account functions are implemented in the deposit, withdrawal, fundTransfer, billPayment, viewStatus, openAccount and closeAccount methods of SB-Acc Transaction and Current-Acc Transaction classes.

- Account maintenance related operations are implemented in Account class through setAccountdetails and getAccountdetails methods.

- The controllers for performing the transaction through various types of devices are defined in displayContentViewerController and displayFormatController methods of UI Management class.

The above mentioned information is specified in the Requirements Class Association Matrix for Banking System as shown in Table 5.5.
Table 5.5 Requirements class association matrix for banking system

<table>
<thead>
<tr>
<th>RequirementBT1</th>
<th>Authentication</th>
<th>SB-Acc Transaction</th>
<th>Current-Acc Transaction</th>
<th>Account</th>
<th>UI Management</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aV</td>
<td>uV</td>
<td>d</td>
<td>w</td>
<td>..</td>
</tr>
<tr>
<td>RequirementBT1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>RequirementBT2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>RequirementBT3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>RequirementBT4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>..</td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RequirementBT8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>RequirementBT9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
<tr>
<td>RequirementBT10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
</tr>
</tbody>
</table>

**Notation Description:** aV represents accountVerification(); uV represents userVerification(); d denotes deposit(); w denotes withdrawal(); vS represents viewStatus(); gA represents getAccount(); sA denotes setAccount(); dCC represents displayContentController; dFC denotes displayFormatController.
The changes in the authentication strategy, procedure for depositing the money and account maintenance procedure are posted dynamically to the banking transaction system. Adapting these changes by the system is measured using the CBAC metric, which is calculated using the data specified in Table 5.5.

The adaptability of user authentication strategy change and account validation procedure changes is represented with the number of classes associated with authentication requirements, which is determined as 1 from Table 5.5 and the corresponding CBAC is 1/5 for each change. Similarly, CBAC value for changes in the deposit, withdrawal, fund transfer and bill payment policies are observed as 2/5 for each, where 2 represents the number of classes associated with each process and CBAC value for changes in account maintenance process is derived as 2/5. The CBAC value for adapting changes in the device controller is 2/5. The dynamic changes proposed in the banking system and the corresponding CBAC values have been shown in Table 5.6.

<table>
<thead>
<tr>
<th>Dynamic Changes</th>
<th>CBAC/AOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication Strategy</td>
<td>0.2</td>
</tr>
<tr>
<td>Banking Transaction Strategy</td>
<td>0.4</td>
</tr>
<tr>
<td>Account Maintenance Strategy</td>
<td>0.4</td>
</tr>
<tr>
<td>User Interface Management Strategy</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 5.6 Adaptability values for banking system using CBAC
The above discussion shows the procedure of measuring the adaptability of changes in the requirements using the number of methods or classes in the system involved in the realization of those requirements. Also it is derived that the value of CBAC associated with the changes in the requirement is high when more number of methods and classes realize the requirement. Changes in the requirements are classified into two types. They are Single Reflection, which implies the changes to be reflected on one method or class in a system and Multiple Reflection, which represents the association of changes with multiple methods or classes.

The CBAC values observed for adapting these categories of changes are shown in Table 5.6. From this observation, it is concluded that the CBAC value for single reflection is 1/5, where ‘5’ represents the number of classes in the system; CBAC value for reflecting changes on two methods or classes is 2/5; CBAC value for adapting the changes that affect three methods or classes is 3/5; and CBAC value for ‘n’ number of reflections is n/5. Adaptability values observed for the changes stated in the above case studies clearly justify the statement stated as „The adaptability of the system (AOS) is increased with increasing the CBAC factor”. This result is clearly shown in the adaptability chart depicted in Figure 5.1.
Figure 5.1 Adaptability chart
5.2.4 Comparison of Adaptability Efficiency of a System

The adaptability efficiency of a system is measured using CBC and CBAC metrics based on the procedure followed for implementing the adaptability in the system. The adaptability of a system that accommodates the changes using object-oriented principles is measured using CBC, whereas the system that adapts to the changes using the proposed adaptable middleware is measured using CBAC. In Sections 5.1.2 and 5.2.3, these metrics are used to measure the adaptability of a banking transaction system to the changes described in the various scenarios.

According to the demonstration of CBC metric using banking transaction system in Section 5.1.2, the adaptability of the changes is performed as follows.

- While incorporating the changes in the user validation and account validation procedures, all the methods defined the SB-Acc Transaction and Current-Acc Transaction classes are to be regenerated to apply the revised version of these validation procedures. Since the number of methods associated with these validation procedures are more, the adaptability of these changes become complex and consumes more effort. This is shown through the CBC values generated for these validation methods defined in Authentication class. The CBC value for the methods associated with the changes is 36/5, and the corresponding adaptability efficiency is 0.14.
Similarly, the associations of the methods defined in the SB-Acc Transaction and Current-Acc Transaction classes are considered as 9 for each class. Any change that occurs in the methods of these classes should be reflected on nine methods in other classes. Hence, the CBC value of each class is 9/5 and the corresponding adaptability efficiency is 0.56.

While displaying the account details, the structure of the display forms and representation of the content will vary according to the device. The displayFormatController and displayContentController methods of UI Management class are defined to realize the display format structure and content structure. These two methods are used in viewStatus method defined SB-Acc Transaction and Current-Acc Transaction classes. Hence, the changes in the display format and content structure should get reflected in the account status display operations defined in the corresponding classes. The CBC value of these methods is specified as 4/5 and the adaptability efficiency is shown as 1.25.

All the above results show that an increase in CBC value of the class reduces the efficiency of accommodating changes in the methods of that class. It is also derived that if a change in the association between the change and the system is one, that is named as single reflection; then the adaptability value is high; otherwise the adaptability value is low.
Applying CBAC metric to measure the adaptability efficiency of banking application that is integrated with the proposed adaptable middleware is discussed in Section 5.2.3 and the results derived from the discussion are given below.

- While incorporating the changes in the requirements of validating the user and account, the component functions that realize those requirements will get affected. Hence, aspect that is realizing changes in the user validation strategy should get bind with userValidation function defined in Authentication component. Weaving of validation change implemented in the advice of an aspect with the function will not affect the structure of the function, which implies that the functions associated with the userValidation are not to be regenerated. This shows that the efficiency of adapting the changes in the user validation is measured as the association of aspect with the userValidation function, which is specified as CBAC (user validation change) and its value is 1/5. In the same way, CBAC (account validation change) is measured as 1/5. This implies that for a single reflection, the effort taken for implementing the changes is to be applied in only one function of the system, which reduces the efficiency of the system to adapt the single reflection changes.

- The deposit, withdrawal, fund transfer, bill payment, opening and closing account transaction services are implemented in SB-Acc Transaction and Current-Acc Transaction classes. Hence, CBAC value for the changes in each transaction service is represented as 2/5, which implies that the
association of aspect that implements changes in these services with the system is 2. While the association between the changes and system is more, then by expending the same effort that is incurred for a single reflection change, the changes are incorporated in more than one function in a system, which increases the adaptability efficiency of a system. This is clearly shown through the CBAC value for adapting the changes in a transaction service.

- The variations in the device through which the users are accessing the banking system services is incorporated by weaving the implementation of changes in display format and content structure with the corresponding methods in UI Management class. The observed CBAC value is 2/5.

- The overall observation is the efficiency of adapting the changes by a system that is integrated with the proposed adaptable middleware is high, if more number of functions in a system is associated with the changes; otherwise the adaptability efficiency is low.

The comparison result of the adaptability of a system that is not using the proposed adaptable middleware is evaluated using CBC metric with the same system that uses the proposed adaptable middleware for adapting changes in the requirements, which is evaluated using CBAC metric. This is depicted in Figure 5.2.
Figure 5.2 AOS comparison result
5.3 EXECUTION LEVEL METRICS

5.3.1 Metric Formulation

Functional Change Adaptation Time (FCAT) - Compositional Approach

The general way of adapting the changes in requirements at runtime is through reconfiguration techniques. The reconfiguration involves the composition of components of a system with the components that implement the changes. This approach is used generally in object-oriented systems for adapting changes in the requirements. Changing the requirements imply changes in the functions of the components. Hence, the time taken for a system to adapt the changes in the requirements is represented as Functional Change Adaptation Time (FCAT). According to the compositional approach, FCAT is specified as the sum of the Reconfiguration Time (RCT) and the Execution Time of the service along with the changes ($ET(f+\delta F)$), which has been expressed in Equation 5.8.

\[
FCAT = RCT + ET(f+\delta F) \tag{5.8}
\]

The value of RCT varies according to the complexity incurred in the component structure. Hence, RCT increases with the increased component structure complexity.

FCAT – Proposed Adaptable Middleware

The method for measuring the adaptive efficiency of the proposed adaptable middleware model is deduced using the micro-measurements stated in Haupt and Mezini (2004). The adaptive efficiency of the system is specified as the time taken by the system to adapt the functional changes, which is represented as FCAT. FCAT of the proposed adaptive middleware is derived as the sum of (i) the time taken for defining the advice that is referred to as Advice Definition Time (ADT), (ii) the time taken for weaving the
advice with the joint-points (JP) that is specified as Advice Weaving Time (AWT) and (iii) the time taken for executing the function along with the advice that is mentioned as Functional Change Execution Time (FCET). This FCAT derivation is expressed in Equation (5.9).

\[ FCAT = ADT + AWT + FCET \]  \hspace{1cm} (5.9)

The Advice Weaving Time (AWT) is described as the time taken to wrap the function associated with a change using introspection, which is represented as Wrapping Time (WrapT) and the time taken to weave the advice with a wrapper that is referred to as Advice Wrapper Weaving Time (AWWT). By extending AWT as stated above, the FCAT expression specified in (5.9) can be rewritten as stated in Equation (5.10).

\[ FCAT = ADT + (WrapT + AWWT) + FCET \]  \hspace{1cm} (5.10)

The Wrapping time and Advice Wrapper Weaving Time are calculated by measuring the time complexity of the wrapping and weaving functions. The time complexity of the wrapping and weaving functions are measured as linear, which shows that the time taken for executing the processes is minimum that is negligible. Hence, FCAT specified in Equation (5.10) is rewritten as follows:

\[ FCAT = ADT + FCET \]  \hspace{1cm} (5.11)

The Advice Definition Time (ADT) varies according to the complexity of the functional change specification. Hence, the value of FCAT is directly proportional to FCET by assigning the common value for ADT and so Equation (5.11) is rewritten as stated in Equation (5.12).

\[ FCAT = \alpha \cdot FCET, \text{ where } \alpha \text{ represents ADT value} \]  \hspace{1cm} (5.12)
5.3.2 Metric Demonstration

Case Study: Banking Transaction System using RMI

The banking transaction system described in the Section 4.4 was implemented using Java Remote Method Invocation (RMI) framework according to the design structure shown in Figure 4.4. The design shows the adaptable structure of a banking transaction system using the proposed adaptable middleware model. The evaluation of adaptability of this system is performed while the system is in execution mode. The complexity involved in executing the changes stated in validation strategies, transaction service policies, account management procedures and user interface management controls are measured using FCAT metric for adaptable middleware model proposed in Equation 5.12 of Section 5.3.1.

According to Equation 5.12, the product of time taken for defining the change implementation code in the advice of the aspect (ADT) and the time taken for executing the function along with the change (FCET) is considered as the measure for describing the adaptability performance of a system. Since the operations involved in the change description are of low complexity, the value of ADT is negligible. Hence, the functional change adaptation time (FCAT) is measured using FCET, that is the time taken for executing the function associated with the strategy change is calculated by measuring the execution time of user validation function along with the change of incorporating the bio-metric validation; FCAT value for transaction service policy changes is measured by executing the deposit or withdrawal or fund transfer functions with the change of integrating with the income tax service; FCAT value for account maintenance procedure changes is determined by measuring the execution time of the functions get account and
set account with the changes; and the execution time of the functions display format and content structure along with the change in the device type is considered to represent FCAT value for the corresponding changes. The FCAT values observed for the changes are specified in Table 5.7.

The adaptability of changes in a banking transaction system is also achieved using compositional approach. Here, change in the user verification procedure is implemented in a separate class and composed with Authentication class through function overloading. Similarly, integration of deposit, withdrawal and fund transfer transaction services along with income tax service is implemented by defining an interface with an operation to record transaction amount in income tax record of the respective customer. The display format and content structures are defined in a separate class for a different device type. According to the device used by the customer for accessing the banking application, the corresponding function defined in a class is invoked.

The FCAT value for these changes are calculated by executing these functions with the modification as specified above, which is shown in Table 5.7. The generated FCAT data in Table 5.7 implies that 30 to 35 percentage of time is saved while adapting the changes using AMM approach when compared with compositional approach. In the same way, the multiple changes on each function and single change on multiple functions of a banking transaction system is implemented using the proposed adaptable middleware model (AMM) and the compositional approaches and the efficiency of adaptability is measured using FCAT metric. From the generated FCAT data, it is observed that FCAT using compositional approach is 30 to 35 percentages higher than the FCAT using AMM approach.
Table 5.7 FCAT for requirement changes in banking transaction system

<table>
<thead>
<tr>
<th>Requirement Change Specification</th>
<th>FCAT Value – Compositional Approach (milliseconds)</th>
<th>FCAT Value – AMM (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change in the User Verification</td>
<td>281</td>
<td>203</td>
</tr>
<tr>
<td>Change in the deposit policy</td>
<td>297</td>
<td>219</td>
</tr>
<tr>
<td>Modification in fund transfer strategy</td>
<td>289</td>
<td>215</td>
</tr>
<tr>
<td>Variation in device type</td>
<td>281</td>
<td>212</td>
</tr>
<tr>
<td>Modification in account maintenance</td>
<td>293</td>
<td>217</td>
</tr>
</tbody>
</table>

**Case Study: Banking Transaction System using Android**

The Banking Transaction application was developed in Android platform to perform the operation including account creation, money deposit and withdrawal and view transactions details. The various android activities are created to (i) open a new account and populate the account details in an Account Table (ii) perform the deposit and withdrawal transactions and also populate the transaction details in the Transaction Table and (iii) view the transaction details. The navigation design of the Banking Transaction application is shown in Figure 5.3. The following are the run-time changes in the above mentioned activities: (i) including the permanent income tax access number, permanent national resident number details while creating the account (ii) providing digital signature option for the transaction which involves the amount that exceeds 100,000 and (iii) listing the transaction details based on the type of the transaction. These run-time changes are incorporated through the integration of the adaptable middleware with this application.
Figure 5.3 Android banking transaction application
The run-time changes mentioned above were implemented using the proposed Adaptive Middleware Model (AMM). The time taken for adapting those changes was calculated using the formula as stated in Equation (5.12) of Section 5.3.1. Since the complexity of the run-time changes specified on the activities is considered as very low, the value of \( \alpha \) in FCAT is assigned one and FCAT is considered as FCET.

The efficiency of the compositional approach for adapting the run-time changes stated on the mobile banking transaction services was calculated using the formula as stated in Equation (5.8) of Section 5.3.1. The changes in the services were implemented using a separate set of components and they were deployed in the same server in which the components are deployed, which makes the time taken for reconfiguration (RCT) is negligible. But this is not applicable for distributed context. The time taken for executing the run-time changes, which were implemented using the compositional approach (considering RCT value as negligible) and using proposed adaptable middleware model has been shown in Table 5.8 and Table 5.9. The result shows that the time taken for adapting the run-time changes using AMM is approximately 5 percent greater than the time taken for adapting the run-time changes using the compositional approach (neglecting RCT). Without neglecting RCT, it is observed that the adaptability time using the compositional approach will be greater than or equal to the time taken using AMM.
Table 5.8 FCAT for single reflection on android banking service

<table>
<thead>
<tr>
<th>Run-time Changes</th>
<th>FCAT Value – Compositional Approach (Execution time in ms)</th>
<th>FCAT Value - AMM (Execution time in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change on creating/adding the account</td>
<td>20000</td>
<td>21000</td>
</tr>
<tr>
<td>Change on adding the transaction</td>
<td>17700</td>
<td>18500</td>
</tr>
<tr>
<td>Change on listing the recent transaction</td>
<td>6800</td>
<td>7000</td>
</tr>
</tbody>
</table>

Table 5.9 FCAT value of multiple reflections on android banking service

<table>
<thead>
<tr>
<th>Run-time Changes</th>
<th>Implemented using Compositional Approach (Execution time in ms)</th>
<th>Implemented using AMM (Execution time in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes on creating/adding the account</td>
<td>22700</td>
<td>21300</td>
</tr>
<tr>
<td>Changes on adding the transaction</td>
<td>19800</td>
<td>18500</td>
</tr>
<tr>
<td>Changes on listing the recent transaction</td>
<td>8100</td>
<td>7500</td>
</tr>
</tbody>
</table>
The result shown in Table 5.8 is the FCAT for incorporating changes on a single mobile service, which is referred to as Single Reflection. The same experiment was carried out by considering multiple changes on each service and also by considering a change on multiple services, which is referred to as Multiple Reflections. The result of the experiment without considering RCT value has been shown in Table 5.9. From this result, it is observed that the time taken for adapting the changes using the compositional approach is approximately 7 percent higher than the time taken for adapting the changes using aspects.

By observing the FCAT value generated for various changes specified in the banking transaction system implemented as RMI and Android applications, it is deduced that applications integrated with the proposed adaptable middleware model for adapting the changes consumes less time than adapting the changes using the compositional approach. The ratio of deviation of FCAT value between compositional approach (CA) and proposed adaptable middleware model (AMM) used by RMI and Android banking application for adapting single and multiple changes in the requirements are shown in Figure 5.4.

![Figure 5.4 CA and AMM comparison result](image-url)
5.4 SUMMARY

The metrics described in this chapter are used to evaluate the adaptability efficiency of a software system in the design and execution levels. The adaptation capability of an object-oriented system that is achieved using compositional approach or/and inheritance approach is measured by applying CBC metric on the structure of the classes and by using FCAT metric at the time of executing the invocation of the changes in the requirements. The adaptability of the software system designed using the proposed adaptable middleware model is measured using CBAC and FCAT metrics, which showed the effectiveness of using aspects to reflect the changes on the functions of the components that are associated with the changes. These proposed metrics are applied on a banking transaction system to evaluate its efficiency of adapting the changes and it has been observed that the efficiency of accommodating the changes into more than one component using the proposed adaptable middleware is better than using compositional approach.