# Adaptability Metric for Adaptation of the Dynamic Changes

Subbian Suganthi[1] and Rethanaswamy Nadarajan[2]
[1]Department of Computer Technology and Applications, Coimbatore Institute of Technology, India
[2]Department of Applied Mathematics and Computational Sciences, PSG College of Technology, India

**Abstract**: *Adapting dynamic changes in the user needs or in the environment is considered as one of the important quality attributes of a system in the pervasive or ubiquitous environment. An aspect-oriented framework to modularize the dynamic changes using aspects is considered as a solution for creating dynamic adaptable systems. This framework allows the system to reflect the dynamic changes on the associated components through aspects without altering the structure of the components. For evaluating the adaptability of this framework, a new adaptability metric has been proposed using the principles of coupling. In this work, coupling is defined as a Conceptual coupling between Aspects and Classes (CBAC), which represents the semantic association between the aspects that are used to represent dynamic changes and the components that are associated with the dynamic changes at the architecture level. The adaptable efficiency of the system that is the ability of reflecting the dynamic changes on the components associated with those changes is measured using the proposed conceptual coupling metric. Based on the measures it is concluded that adaptability efficiency of the system is increased with increasing the coupling between the aspect and the components. The proposed CBAC metric is evaluated and demonstrated by measuring the adaptability of the dynamic changes in the requirements of the various software systems.*

**Keywords**: *Software adaptability, modularization, aspect-oriented approach, dynamic changes, adaptability metric, coupling metric.*

*Received February 11, 2013; accepted May 6, 2013; published online August 9, 2015*

## 1. Introduction

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 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 [12] 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 and representing them as aspects. Here adaptability metric is proposed to evaluate this adaptable middleware framework. The work described in this paper refers the dynamic adaptable solution stated in this adaptable middleware framework as an aspect-oriented solution for dynamic adaptability.

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; and applying the metric and calculating the value of adaptability degree [13]. 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, number of external services requested from other classes and number of attributes. Hence, it is concluded that measuring the adaptability of the software at the architecture level is a complex task. In this work the above strategy is refined and the evaluation of the adaptability of dynamic changes is performed 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 service and classes/components. It implies coupling metrics are appropriate for measuring the adaptability.

In this work, the metric for evaluating the adaptability of an adaptable middleware framework is proposed using coupling principle. Based on the dynamic quantification of the system behavior feature of an aspect-oriented approach [3] and aspect-oriented design principles stated in [14], it is derived that the coupling principle is appropriate to evaluate the adaptability of the aspect-oriented framework. In general, coupling metric is used to measure the level of interdependency between modules/components/classes in a system [8, 9]. In this way, coupling between the dynamic changes represented as aspects and the component/classes associated with those changes is used measure the adaptability of an adaptable middleware framework. This coupling measure is named as Conceptual binding between Aspects and Classes (CBAC).

## 2. Related Works

The following are the discussions on the various research works carried out on determining the measures for evaluating the modularity and Adaptability of the System (AOS).

The cohesion and coupling of a class can be measured using method signatures at design level were proposed by Kuljit and Hardeep [5]. The paper [7] presented a comparative study on modularizing the systems using object-oriented and aspect-oriented approaches. In that work, the systems were evaluated using the Coupling between Object classes (CBO) and Lack of Cohesion in Methods (LCOM), which are the general metrics used for assessing modularity of the object-oriented systems. These general metrics were refined and proposed as the metrics for assessing the modularity of the aspect-oriented system. The CBO metric principles stated in the above work are redefined and specified as conceptual binding between aspects and classes in our approach. The semantic information shared between the elements of the source code of the classes was defined as the conceptual coupling between the classes and proposed as the coupling measure for object-oriented systems in [6]. But in our work, the semantic information sharing stated above is redefined as number of classes sharing an aspect using point-cut specification at the architectural level.

The metrics for adaptability described below were proposed in [11].

- Element Adaptability Index (EAI), where EAI=1 for an adaptable element and EAI=0 for a non-adaptable element.
- Architecture Adaptability Index (AAI)=EAI for all elements of architecture/total number of elements.
- Software Adaptability Index (SAI)=AAI for all architecture of the software/total number of architectures for that software.

These metrics were used to evaluate the adaptability at the architectural level. In the above metrics, the semantic coherence existing between the methods across the classes was not included whereas in our work the semantic coherence between the aspects and the methods of different classes is measured to evaluate the adaptability at the architectural level.

Making use of object-oriented metrics suite stated in [2], for evaluating the aspect-oriented system is analyzed in the work proposed in [15]. The Chidamber and Kemerer (C and K) metric suite includes Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Lack of Cohesion of Methods (LOCM), CBO and Response for a Class (RFC). The analysis shown that the concepts used in C and K metrics were appropriate to evaluate an aspect-oriented system. Based on this analysis result, concept of CBO is used to design adaptability metric.

Measuring the impact of the aspect-oriented approach in maintainability using coupling metrics was proposed by Rachel *et al.* [8]. The coupling metrics stated in that work are coupling on advice execution and number of degree diffusion point-cuts. These metrics can be used to measure the adaptability at the code level not at the architectural level.

The work described by Haupt and Mezini [4] proposed micro measurements for dynamic AOP systems, which includes cost of dynamic (un)weaving, cost of executing the method along with the advice and cost of passing the advised method's parameters to an advice. It implies that the performance measure of the dynamism involved in an AOP system at the execution level.

AOP framework to encapsulate the software measurement process without affecting the software under analysis was proposed by Cazzola and Marchetto [1]. This framework could be extended to measure the adaptability efficiency of the system.

## 3. Adaptability Metric

Using aspect-oriented paradigm, coupling between the classes can be reduced by modularizing the cross-cutting concerns [10], which improve the AOS. This leads to propose the technique of representing the dynamic changes using aspects in the adaptable middleware framework [12]. As coupling is considered as the primary property that influences the maintenance task [6], the AOS can be measured using coupling metrics. Since, aspect-oriented approach is an extension of object-oriented approach, CBO metric stated for object-oriented systems can be used to measure aspect-oriented systems [15]. A new dimension for measuring the AOS through redefining CBO metric is proposed here. According to the proposed adaptable middleware framework, system adapts to the dynamic changes by representing them as aspects and weaving those aspects with the corresponding functions of the classes/components. Hence, adapting the dynamic change by a system is based on the number of components/classes associated with that change, which is represented as aspect or the reflections of dynamic changes specified in the aspect over the classes. This measure is referred as the CBAC or the distribution of aspects among the classes. Hence, adaptability achieved using the adaptable middleware framework is measured using the distribution factor of an aspect or the conceptual binding between the aspect and the classes. Conceptual binding between the aspect and classes is measured with the Conceptual binding between the Aspect and Methods (CBAM) in the classes.

Let us denote the set of classes $C=\{c_1, c_2, ..., c_n\}$, where '*n*' is the number of classes in a software system and set of methods in each class $c_i \varepsilon C$ is represented as $M(c_i)=\{m_{i1}, m_{i2}, ..., m_{ik}\}$, where $1 \leq i \leq n$ and $k$ is the number of methods in a class $c_i$ ($|M(c_i)|$.

Initially, the association between the methods of a class with dynamic changes is to be measured, which is referred as CBAM. The changes in the functions of the software system can be interpreted as changes in the requirements specification of the system. Since, the methods of the classes are considered as the realization units of the requirements in the software system, CBAM is measured as the number of methods of the classes realized the requirements associated with the changes.

The association between the requirements and the classes is represented using Requirements Class Association Matrix (RCAM) as shown in the Table 1. RCAM ($p$, $q$), ($1 \leq p \leq n$ and $1 \leq q \leq h$, where $h = \sum_{i=1}^{n} \left| M\left(c_i\right) \right|$ takes value 1, if Req#p is defined in the method $M(c_i)$; otherwise, it takes value 0. Expression of CBAM using RCAM is shown in the Equation 1.

$$CBAM\left(a, M\left(c_i\right)\right) = \sum_{j=1}^{k} w\left(a, m_{jk}\left(c_j\right)\right) \tag{1}$$

Where $c_i$™$C$, $m_{ik}$™$M(c_i)$ and '*a*' denotes the aspect defined for representing dynamic changes in the requirement; and $w(a, m_{i,k}(c_i))$= RCAM($p$, $q$), where aspect 'a' is associated with the Req#p.

CBAC is specified as the average of CBAM of each class in the system, which is expressed in the Equation 2.

$$CBAC\left(a, c\right) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{\left|M\left(c_i\right)\right|} (CBAM\left(a, m_{ij}\left(c_i\right)\right)}{\left|c\right|} \tag{2}$$

Where '*C*' represents the set of classes in the system, $m_{ij}$™$M(c)$ and '*a*' denotes the aspect implementing the dynamic changes.

Table 1. Requirements class association matrix.

| | **c₁** | | | | **c₂** | | | | **..** | **cₙ** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | m₁₁ | m₁₂ | … | m₁ₙ | m₁₁ | m₁₂ | … | m₁ₙ | … | m₁₁ | m₁₂ | .. | m₁ₙ |
| Req#1 | 0/1 | 0/1 | … | 0/1 | 0/1 | 0/1 | … | 0/1 | … | 0/1 | 0/1 | | 0/1 |
| Req#2 | 0/1 | 0/1 | … | 0/1 | 0/1 | 0/1 | … | 0/1 | … | 0/1 | 0/1 | | 0/1 |
| | | | … | | | | … | | … | | | | |
| Req#n | 0/1 | 0/1 | … | 0/1 | 0/1 | 0/1 | … | 0/1 | … | 0/1 | 0/1 | | 0/1 |

Assume that there are '*n*' aspects say $A=\{a_1, a_2, ..., a_{n,}\}$, then the conceptual binding between the set of aspect '*A*' with the set of classes '*C*' is represented as CBAC(*A*, *C*), which is also expressed as distribution of aspects in the set '*A*' over the system (DOF(*A*)) that is shown in the Equation 3.

$$DOF\left(A\right) = \frac{\sum_{i=1}^{n} CBAC\left(a_i, C\right)}{\left|A\right|} \tag{3}$$

From the above discussion, it is clearly stated 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 or the distribution of the aspects over the methods in the classes, which is expressed in the Equation 4. It leads to derive a lemma stated below:

$$AOS_{dynamic\ requirements\ change}=DOF(Aspects) \tag{4}$$

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

The adaptable middleware framework [12] provides the facility of adapt to changes by defining those changes as aspects and weaving them with 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 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 and then possibility of adapting the changes implemented in those aspects will be high. This discussion concludes that increasing the CBAC will increase the AOS.

## 4. Adaptable Metric Evaluation and Application

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 [2] 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 are described in the following sections.

### 4.1. Metric Evaluation

The CBAC is evaluated based on the properties stated in [2]. The property list includes non-coarseness, non-uniqueness, permutation is significant, function implementation is important, monotonicity, non-equivalence interaction and Interaction complexity.

Non-Coarseness: The CBAC metric of two different aspects are not same. CBAC ($a_1$, $C$)≠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, CBAC ($a_1$, $C$)=CBAC ($a_2$, $C$).

Permutation is Significant: If the aspect $a_1$ is the permutation of $a_2$, which does not change the coupling between the aspect with the classes. Hence, CBAC ($a_1$, $C$)=CBAC($a_2$, $C$). Hence, it does not satisfy this property.

Function Implementation is important: If the functions of aspects $a_1$ and $a_2$ are similar and are having different implementation then CBAC ($a_1$, $C$)$\neq$ CBAC($a_2$, $C$). This implies that coupling is determined based on the implementation of the aspects not on the type of the operation.

Monotonicity: Let $a_1$ and $a_2$ are the aspects and CBAC ($a_1$, $C$)=$n_1$, CBAC($a_2$, $C$)=$n_2$. 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$0. Hence, CBAC ($a_1$+$a_2$)$\geq$ CBAC($a_1$) and CBAC($a_1$+$a_2$)$\geq$ CBAC($a_2$), which implies CBAC metric satisfies monotonicity.

Non-equivalence interaction: Let $a_1$, $a_2$ and $a_3$ are the aspects and CBAC ($a_1$, $C$)=$n_1$, CBAC ($a_2$, $C$)=$n_2$, CBAC($a_3$, $C$)=$n_3$. 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.

CBAC (($a_2$+$a_3$), $C$)=$n_2$+$n_3$-$\lambda$, where '$\alpha$' 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, CBAC (($a_1$+$a_3$), $C$) is not equal to CBAC (($a_2$+$a_3$), $C$). It implies 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.

## 2.2. Metric Application

In this work, the conceptual binding between the set of aspects '$A$' with the classes $c_i \varepsilon C$ (CBAC ($A$, $C$)) represents the AOS that is designed using the aspect-oriented solution proposed in an adaptable middleware framework. The CBAC metric is also considered as the measure shows 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.

### 4.2.1. Banking Transaction System

The requirements for banking transaction system are stated as follows:

- Req#BT1: Authenticate the user.
- Req#BT2: Allowing the user to perform deposit.
- Req#BT3: Perform withdrawal transaction.
- Req#BT4: Perform fund transfer transaction.
- Req#BT5: Maintain the account details.
- Req#BT6: Retrieve the account details.

These requirements are realized in the system through the methods defined in the classes specified in the class set $C$ and detailed descriptions regarding the class design is given below:

*C={Authentication, SBAccTransaction, CurrentAccTransaction, LoanAccTransaction, Account}*

- User authentication process is implemented in the validate method of the Authentication class.
- Deposit, Withdrawal and Fund transfer processes are implemented in the deposit, withdrawal and fundtransfer methods of SBAccTransaction, CurrentAccTransaction and LoanAccTransaction classes.
- Account maintenance related operations are implemented in Account class through setAccountdetails and getAccountdetails methods.

Above mentioned information are specified in the Requirements Class Association Matrix for Banking System as shown in Table 2.

Table 2. Requirements Class Association Matrix for Banking System.

| | **Auth** | **SBAccTr** | | | **CAccTr** | | | **LAccTr** | | | **Account** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | v() | d() | w() | f() | d() | w() | f() | d() | w() | f() | g() | s() |
| **Req#BT1** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Req#BT2** | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Req#BT3** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **Req#BT4** | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **Req#BT5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **Req#BT6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Notation Description: Auth denotes Authentication class, SBAccTr denotes SBAccTransaction class, CAccTr denotes CurrentAccTransaction class, LAccTr denotes LoanAccTransaction class, d() denotes deposit(), w() denotes withdrawal(), f() denotes fundtransfer(), g() denotes getAccountdetails() and s() denotes setAccountdetails().

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 the Table 2.

Adaptability of an authentication strategy change is represented with the number of classes associated with authentication requirements, which is determined as 1 from Table 2 and the corresponding CBAC is 1/5. Similarly, CBAC value for change in the deposit policy is observed as 3/5, where 3 represents the number of classes associated with the deposit process and CBAC value for change in account maintenance process is derived as 1/5. The dynamic changes proposed in the banking system and the corresponding CBAC values are shown in Table 3.

Table 3. CBAC factor for banking system.

| Dynamic Changes | CBAC Value |
|---|---|
| **Authentication Strategy Change (Single Reflection)** | 0.2 |
| **Deposit Policy Change (Multiple Reflection)** | 0.6 |
| **Account Maintenance Process Change (Single Reflection)** | 0.2 |

### 4.2.2 Sales Processing System

Following are the requirements stated for the sales processing system:

- Req # SO1: Check the validity of the sales order.
- Req# SO2: Order confirmation and delivery scheduling.
- Req # SO3: Update order status to the customer.
- Req # SO4: Bill generation and Payment.
- Req# SO5: Shipment process.

These requirements are specified in the software system through Order Processor, Order Status Publisher, Bill Generator, Payment Handler and Shipment Processor classes and the set C is defined with these classes. Methods defined in each class and their association with the requirements is shown in the Table 4. Requirement #SO1 is realized in the order Validation() method of Order Processor class. Hence, changes in the order validation should get reflected only on that method and the corresponding CBAC value is 1/5, where 5 is the number of classes. The changes in the order confirmation and notification to refection on the order Confirmation () method of Order Processor class and eMail() and sMs() methods of Order Status Publisher class. Hence, the association between process of order confirmation and system is derived as 3 and adaptability of changes in this process is specified as 3/5. Similarly order status notification process is realized in the methods of order status publisher and shipment process classes and CBAC value of adapting changes in this process is 3/5. The coupling between the Requirement#SO4 with the classes in the system is measured as 3 that is salesInvoice(), cash() and credit() methods define this requirement. Hence, adapting the changes in the bill generation and payment process is measured as 3/5. The adaptability measures determined for incorporating the dynamic changes stated in the sales processing system is shown in Table 5.

Table 4. Requirements class association matrix for sales processing system.

| | OP | | OSP | | BG | PH | | SP |
|---|---|---|---|---|---|---|---|---|
| | oV() | oC() | em() | sMs() | sIv() | ch() | cr() | ny() |
| **Req#SO1** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Req#SO2** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **Req#SO3** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| **Req#SO4** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| **Req#SO5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Notation Description: Order Processor class denotes (OP), Order Status Publisher class denotes (OSP), Bill Generator class denotes (BG), Payment Handler class denotes (PH), Shipment Processor class denotes (SP) order Validation()denotes (oV()), order Confirmation() denotes (oC()), email() denotes (em()), salesInvoice() denotes (sIv()), cash() denotes (ch()), credit() denotes (cr()) and notify denotes (nv).

Table 5. CBAC factor for sales processing system.

| Dynamic Changes | CBAC Value |
|---|---|
| **Order Validation Strategy Changes (Single Reflection)** | 0.2 |
| **Order Confirmation Policy Changes (Multiple Reflection)** | 0.6 |
| **Changes in the Format/ Mode of Order Status Notification (Multiple Reflection)** | 0.6 |
| **Payment Process Changes (Multiple Reflection)** | 0.6 |
| **Changes in the Shipment Procedure (Single Reflection)** | 0.2 |

The above discussion shows the ways in which the adaptability of changes in the requirements is measured using the number of methods/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 are used to implement that requirement. Changes in the requirements are classified in to two types namely, changes to be reflected on one method/class (Single Reflection) and changes to be reflected on multiple methods/classes (Multiple Reflection). The CBAC values observed for adapting these categories of changes shown in Table 3 and Table 4. From this observation, it is concluded that the CBAC value for single reflection is 1/5, where '5' represents number of classes in the system; CBAC value for two reflections is 2/5; CBAC value for three reflection is 3/5; and CBAC value for 'n' number of reflection is n/5. Adaptability values observed for the changes stated in the above case studies clearly justify the lemma of The AOS is increased with increasing the CBAC factor'. This result is clearly shown in the adaptability chart depicted in Figure 1.

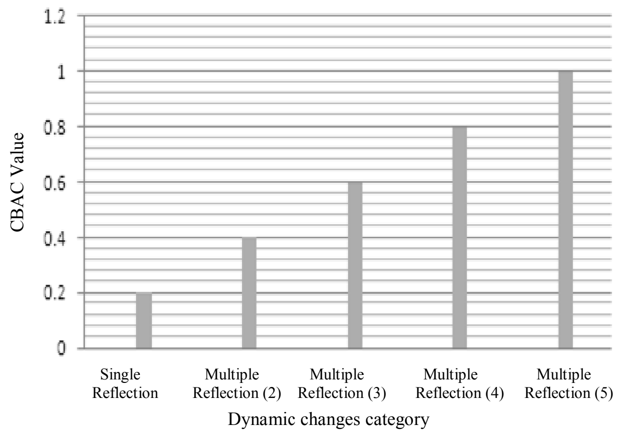

Figure 1. Adaptability chart.

## 5. Conclusions

The work described in this paper proposed a metric for measuring the adaptability efficiency of the system designed using the adaptable middleware framework. An aspect-oriented solution for adapting the dynamic changes was implemented in that framework. The proposed metric was designed using the coupling principle stated for an object-oriented system. Here this coupling principle was redefined based on aspect weaving mechanism stated in aspect-oriented paradigm and named as CBAC. Requirements realization mechanism was used to determine the association between the classes with the aspect, which is represented as an implementation unit of dynamic changes. This metric was evaluated using the properties stated for the software metrics. The validity of the metric was also shown by measuring the adaptability efficiency of the various systems designed with dynamic adaptability feature using that metric. It is also derived that the adaptability efficiency of an adaptable middleware framework based system increases with increasing the value of CBAC. Hence, this metric was considered to measure the adaptability of any system designed with aspects to represent the dynamic changes. In future work, it is proposed to automate the process of measuring the CBAC.

## References

[1] Cazzola W. and Marchetto A., "AOP-Hidden Metrics: Separation, Extensibility and Adaptability in SW Measurement," *Journal of Object Technology*, vol. 7, no. 2, pp. 53-68, 2008.

[2] Chidamber S. and Kemerer C., "Towards A Metrics Suite for Object Oriented Design," *in Proceedings of the 6th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, Arizona, USA. pp. 197-211, 1991.

[3] Filman R. and Friedman P., "Aspect-Oriented Programming is Quantification and Obliviousness," *in Proceedings of Workshop on Advanced Separation of Concerns*, Minnesota, USA, 2000.

[4] Haupt M. and Mezini M., "Micro-Measurements for Dynamic Aspect-Oriented Systems," *Lecture Notes in Computer Science*, vol. 3263, pp. 81-96, 2004.

[5] Kuljit K. and Hardeep S., "An Investigation of Design Level Class Cohesion Metrics," *the International Arab Journal of Information Technology*, vol. 9, no. 1, pp. 66-73, 2012.

[6] Poshyvanyk D. and Marcus A., "The Conceptual Coupling Metrics for Object-Oriented Systems," *in Proceedings of the 22nd IEEE International Conference on Software Maintenance*, Philadelphia, Pennsylvania, USA, pp. 469-478, 2006.

[7] Przybylek A., "An Empirical Assessment of the Impact of Aspect-Oriented Programming on Software Modularity," *in Proceedings of International Conference on Evaluation of Novel Approaches to Software Engineering*, Athens, Greece, pp. 139-148, 2010.

[8] Rachel B., Fabiano F., Alessandro G., and Francois T., "An Empirical Evaluation of Coupling Metrics on Aspect-Oriented Programs," *in Proceedings of the ICDSE Workshop on Emerging Trends in Software Metrics*, Cape Town, South Africa. pp. 53-58, 2010.

[9] Rachel B., Fabiano F., Alessandro G., and Francois T., "Coupling Metrics for Aspect-Oriented Programming: A systematic Review of Maintainability Studies," *in Proceedings of the 4th International Conference on the Evaluation of Novel Approaches in Software Engineering*, Italy, pp. 277-290, 2010.

[10] Steimann F., "The Paradoxical Success of Aspect-Oriented Programming," *in Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems and Applications*, Portland, USA, pp. 481-497, 2006.

[11] Subramanian N. and Chung L., "Metrics for Software Adaptability", available at: www.utdallas.edu/~chung/ftp/sqm.pdf, last visited 1999.

[12] Suganthi S. and Nadarajan R., "Middleware Model for Adapting Dynamic Requirements," *Journal of Digital Information Management*, vol. 10, no. 1, pp. 20-29, 2012.

[13] Tarvainen P., "Adaptability Evaluation at Software Architectural Level," *Open Software Engineering Journal*, vol. 2, pp.1-30, 2008.

[14] Wampler D., "Aspect-Oriented Design Principles: Lessons from Object-Oriented Design," *in Proceedings of the 6th International Conference on Aspect-Oriented Software Development*, Vancouver, pp. 1-10, 2007.

[15] Zakaria A. and Hosny H., "Metrics for Aspect-Oriented Software Design," *in Proceedings of the 6th Workshop on Aspect-Oriented Modeling with UML*, San Francisco, California, USA, pp. 1-6, 2003.

**Subbian Suganthi** recived her Bs degree in 1991 at PSG College of Arts and Science, Bharathiar University, India; and MSc in 1993, MPhil in 1994 at PSG College of Technology, Bharathiar University, India. she is pursuing her PhD degree. in Faculty of Science and Humanities, Anna University, India. Currently, she is an Assistant Professor in the Department of Computer Technology and Applications, Coimbatore Institute of Technology, India. Her research interest is in the area of software architecture, design patterns, component technology and object-oriented analysis and design.

**Rethanaswamy Nadarajan** is the Professor and Head of the Department of Applied Mathematics and Computational Sciences, PSG College of Technology, India. His research areas include object-oriented computing, software engineering, data mining and database management systems. He published many research papers in International Referred Journals.