An Evaluation of Coupling Measures for AspectJ

Marc Bartsch
School of Systems Engineering
The University of Reading, Reading
RG6 6AY, UK
+44(0)118 378 8617
m.bartsch@reading.ac.uk

Rachel Harrison
School of Systems Engineering
The University of Reading, Reading
RG6 6AY, UK
+44(0)118 378 8617
rachel.harrison@reading.ac.uk

ABSTRACT
The maintenance of aspect-oriented software requires measures that are theoretically valid. Management or project decisions made using metrics that have not been validated may be detrimental or unhelpful. Recently, measures have been suggested that focus on aspect-oriented concepts, such as the crosscutting behaviour of aspects. Before these new measures can be put to use they should be evaluated to determine how far they indeed measure what they purport to quantify. This paper focuses on the theoretical evaluation of five aspect-oriented coupling measures with the aim of constructively increasing the quality of software evolution.

Categories and Subject Descriptors

General Terms
Measurement.

Keywords
Measurement, aspect-oriented, coupling.

1. INTRODUCTION
Aspect-orientation is an emerging paradigm that is based on the separation of concerns principle. It offers the idea of a new modular unit that encapsulates crosscutting concerns which would otherwise be scattered across multiple modules. Aspect-oriented languages introduce new forms of coupling which are unknown to object-oriented languages. The execution of base code might trigger the execution of aspect code leading to coupling relationships between classes and aspects which are not transparent. In addition to that, intertype declarations can change class implementations by adding attributes or methods.

Measuring these new kinds of coupling relationships is an issue which has been addressed lately with the definition of coupling measures specifically designed to support aspect-oriented concepts. A maintenance process for aspect-oriented software that relies on the results of these measures must have the confidence that the measures involved do indeed measure what they purport to quantify. Also, the comparison of measurement results is an error prone task if measures can be interpreted in different ways.

The position of the authors is that all measures including coupling measures need to be validated to gain confidence in the results taken from measurement. However, others have pointed out that measures that cannot be validated may still be useful [6]. Since research into measurement of aspect-oriented systems is at an early stage, it is particularly important to validate aspect-oriented measures thoroughly. Also, the authors would like to stress the fact that the validation of aspect-oriented coupling measures as presented in this paper depends on at least two frameworks. First, the validation criteria need to be agreed upon. A measure that validates successfully in the context of one framework might not validate in another. Second, aspect-oriented coupling measures depend on a specific idea of coupling. Mechanisms that constitute coupling in one aspect-oriented language might not exist in another. Hence, a specific aspect-oriented language implementation has to be considered, when aspect-oriented coupling measures are validated. In this paper we focus on measures for AspectJ [1].

This paper presents the evaluation of five selected aspect-oriented coupling measures suggested by Ceccato and Tonella [4]. They define five coupling measures, CAE, CIM, CFA, CMC and CDA, in order to investigate the trade-off between the advantages obtained from a separation of concerns and the disadvantages caused by coupling introduced by aspects. We selected the five measures for two reasons: first, two of the five measures (CMC, CFA) are derived from a well-known object-oriented measure (CBO). Second, measures like CAE, CIM and CDA focus on aspect-oriented core concepts such as static and dynamic crosscutting. We believe that the five measures are good representatives for measuring different kinds of coupling in AspectJ and that they deserve further investigation to make sure they are theoretically valid.

The paper is structured as follows: section 2 presents related work, section 3 introduces two evaluation frameworks used in this paper; section 4 introduces and evaluates the coupling measures. Finally, section 5 offers conclusion and points to further research.

2. RELATED WORK
Aspect-oriented measures derived from Chidamber and Kemerer’s suite of object-oriented measures [5] have been suggested by Ceccato and Tonella [4] and by Sant’Anna et al. [9].

The issue of validating measures has been approached from different viewpoints. Kitchenham et al. define a set of criteria that all measures must obey to be considered a valid measure from a measurement theory point of view [8]. Other validation criteria have been suggested by Briand et al. [3]. They define mathematical criteria that all coupling measures must conform to. As far as construct validity is concerned, these criteria are only necessary but not sufficient [3]. In contrast to the criteria put forward by Kitchenham et al. they do not consider the relationship between the attribute being measured and the mathematical formula that is used to calculate the measure. Instead, they focus on properties of the mathematical formula only. Such criteria can verify that a certain algorithm does indeed measures coupling. They cannot, however, detect whether all coupling relationships that are necessary to measure a certain coupling attribute will be counted. We have chosen the validation framework by Kitchenham et al. in this paper since we are more interested in the question of whether an algorithm indeed represents a specific kind of coupling attribute than in the question of whether an algorithm represents coupling at all.

In [7], Kaner and Bond define a list of 10 questions that they consider relevant for any measure from a practical viewpoint and also raise the issue of construct validity.

3. EVALUATION FRAMEWORKS

The evaluation process for the five coupling measures will be guided by two frameworks. First, a coupling framework for AspectJ which is an extension of a coupling framework for object-oriented systems put forward by Briand et al. [3]. Our coupling framework [2] focuses on coupling mechanisms that do not exist in object-oriented languages or that exists between members which are unknown in object-oriented languages.

The criteria of the framework are:

1. **Type of Connection.** The type of a particular coupling connection is determined by the mechanism that is used to establish the coupling connection. The use of a class identifier as a return type of a piece of advice, for example, is a mechanism that leads to coupling between the aspect that implements the advice and the class whose identifier has been used.
2. **Locus of Impact.** If an aspect is used in a coupling connection, a distinction is made between import and export coupling which defines a counting rule for a coupling connection. If aspects invoke methods of other classes, then import coupling counts the number of classes whose methods would be called by a given aspect. Export coupling counts the number of aspects that make calls to a method of a given class.
3. **Granularity.** Granularity refers to the level of detail at which coupling information is collected. It indicates the components that are counted and how to count multiple occurrences of a connection.
4. **Stability of Server.** Stability of server expresses whether components at the receiving end of a coupling connection are subject to modifications and might influence coupled classes if modifications are applied. This criterion is independent of the distinction between object-oriented or aspect-oriented languages and is beyond the scope of this paper. For the remainder of this paper, only stable classes and aspects will be considered.
5. **Direct or Indirect Connections.** Direct or indirect connections refer to whether to count direct coupling connections only or also indirect connections. For the remainder of this paper, only direct connections will be considered.
6. **Inheritance.** The following questions should be answered with regard to inheritance: First, does the use of members, such as the use of methods, attributes or pointcuts, of an ancestor class or aspect constitute coupling or not? Second, does a measure consider both polymorphically and statically invoked methods and, third, do inherited members belong to the inheriting aspect or not?
7. **Static Crosscutting/Intertype Declarations.** Similar to the question of whether an inherited method or an attribute belongs to a class or an aspect, we have to determine whether introduced methods or attributes belong to a class or not. There are three options. First, an intertype declaration only belongs to the aspect that defines it. Second, an intertype declaration only belongs to the class it defines for or, third, an intertype declaration belongs to both the class and the aspect.
8. **Dynamic Crosscutting.** Join points can reference an executing object and a target object. When a coupling measure is being defined a choice has to be made whether to count only the executing object, the target object, or both.
9. **Instantiation.** Instantiation refers to the question whether to count aspects at a per-instance level or not. This criterion refers to runtime measures and will not be considered further.

Ambiguities can arise from the fact that certain criteria are not addressed in a measure where a choice is possible. For example, criterion 7 refers to a decision of how to assign intertype declarations. A coupling measure that does not address this issue might be interpreted in different ways. We will apply the coupling framework to investigate the well-definedness of each measure, i.e. to find and to resolve ambiguities of each measure.

### 3.1 Coupling Framework

Bartsch and Harrison [2] suggest a coupling framework for AspectJ which is an extension of a coupling framework for object-oriented systems put forward by Briand et al. [3]. Our coupling framework [2] focuses on coupling mechanisms that do not exist in object-oriented languages or that exists between members which are unknown in object-oriented languages.

The criteria of the framework are:

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Type of Connection</td>
<td>The type of a particular coupling connection is determined by the mechanism that is used to establish the coupling connection. The use of a class identifier as a return type of a piece of advice, for example, is a mechanism that leads to coupling between the aspect that implements the advice and the class whose identifier has been used.</td>
</tr>
<tr>
<td>2. Locus of Impact</td>
<td>If an aspect is used in a coupling connection, a distinction is made between import and export coupling which defines a counting rule for a coupling connection. If aspects invoke methods of other classes, then import coupling counts the number of classes whose methods would be called by a given aspect. Export coupling counts the number of aspects that make calls to a method of a given class.</td>
</tr>
<tr>
<td>3. Granularity</td>
<td>Granularity refers to the level of detail at which coupling information is collected. It indicates the components that are counted and how to count multiple occurrences of a connection.</td>
</tr>
<tr>
<td>4. Stability of Server</td>
<td>Stability of server expresses whether components at the receiving end of a coupling connection are subject to modifications and might influence coupled classes if modifications are applied. This criterion is independent of the distinction between object-oriented or aspect-oriented languages and is beyond the scope of this paper. For the remainder of this paper, only stable classes and aspects will be considered.</td>
</tr>
<tr>
<td>5. Direct or Indirect Connections</td>
<td>Direct or indirect connections refer to whether to count direct coupling connections only or also indirect connections. For the remainder of this paper, only direct connections will be considered.</td>
</tr>
<tr>
<td>6. Inheritance</td>
<td>The following questions should be answered with regard to inheritance: First, does the use of members, such as the use of methods, attributes or pointcuts, of an ancestor class or aspect constitute coupling or not? Second, does a measure consider both polymorphically and statically invoked methods and, third, do inherited members belong to the inheriting aspect or not?</td>
</tr>
<tr>
<td>7. Static Crosscutting/Intertype Declarations</td>
<td>Similar to the question of whether an inherited method or an attribute belongs to a class or an aspect, we have to determine whether introduced methods or attributes belong to a class or not. There are three options. First, an intertype declaration only belongs to the aspect that defines it. Second, an intertype declaration only belongs to the class it defines for or, third, an intertype declaration belongs to both the class and the aspect.</td>
</tr>
<tr>
<td>8. Dynamic Crosscutting</td>
<td>Join points can reference an executing object and a target object. When a coupling measure is being defined a choice has to be made whether to count only the executing object, the target object, or both.</td>
</tr>
<tr>
<td>9. Instantiation</td>
<td>Instantiation refers to the question whether to count aspects at a per-instance level or not. This criterion refers to runtime measures and will not be considered further.</td>
</tr>
</tbody>
</table>

**3.2 Validation Framework**

Kitchenham et al. propose a set of four criteria that all valid measures must obey [8]:

<table>
<thead>
<tr>
<th>Validation Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Type of Connection</td>
<td>The type of a particular coupling connection is determined by the mechanism that is used to establish the coupling connection. The use of a class identifier as a return type of a piece of advice, for example, is a mechanism that leads to coupling between the aspect that implements the advice and the class whose identifier has been used.</td>
</tr>
<tr>
<td>2. Locus of Impact</td>
<td>If an aspect is used in a coupling connection, a distinction is made between import and export coupling which defines a counting rule for a coupling connection. If aspects invoke methods of other classes, then import coupling counts the number of classes whose methods would be called by a given aspect. Export coupling counts the number of aspects that make calls to a method of a given class.</td>
</tr>
<tr>
<td>3. Granularity</td>
<td>Granularity refers to the level of detail at which coupling information is collected. It indicates the components that are counted and how to count multiple occurrences of a connection.</td>
</tr>
<tr>
<td>4. Stability of Server</td>
<td>Stability of server expresses whether components at the receiving end of a coupling connection are subject to modifications and might influence coupled classes if modifications are applied. This criterion is independent of the distinction between object-oriented or aspect-oriented languages and is beyond the scope of this paper. For the remainder of this paper, only stable classes and aspects will be considered.</td>
</tr>
<tr>
<td>5. Direct or Indirect Connections</td>
<td>Direct or indirect connections refer to whether to count direct coupling connections only or also indirect connections. For the remainder of this paper, only direct connections will be considered.</td>
</tr>
<tr>
<td>6. Inheritance</td>
<td>The following questions should be answered with regard to inheritance: First, does the use of members, such as the use of methods, attributes or pointcuts, of an ancestor class or aspect constitute coupling or not? Second, does a measure consider both polymorphically and statically invoked methods and, third, do inherited members belong to the inheriting aspect or not?</td>
</tr>
<tr>
<td>7. Static Crosscutting/Intertype Declarations</td>
<td>Similar to the question of whether an inherited method or an attribute belongs to a class or an aspect, we have to determine whether introduced methods or attributes belong to a class or not. There are three options. First, an intertype declaration only belongs to the aspect that defines it. Second, an intertype declaration only belongs to the class it defines for or, third, an intertype declaration belongs to both the class and the aspect.</td>
</tr>
<tr>
<td>8. Dynamic Crosscutting</td>
<td>Join points can reference an executing object and a target object. When a coupling measure is being defined a choice has to be made whether to count only the executing object, the target object, or both.</td>
</tr>
<tr>
<td>9. Instantiation</td>
<td>Instantiation refers to the question whether to count aspects at a per-instance level or not. This criterion refers to runtime measures and will not be considered further.</td>
</tr>
</tbody>
</table>
1. For an attribute to be measurable, it must allow different entities to be distinguished from each other.

2. A valid measure must obey the Representation Condition [6], i.e., it must preserve our intuitive notion about the attribute and the way in which it distinguishes different entities.

3. Each unit of an attribute contributing to a valid measure is equivalent.

4. Different entities can have the same attribute value (within the limits of measurement).

In this paper, we will focus on criterion 2, the Representation Condition. Since all coupling measures which we will discuss are based on counting coupling connections only, criteria 1, 3 and 4 can be validated intuitively and will not be considered any further.

Notions about attributes are subjective and can vary. Not everyone will agree upon what constitutes coupling in aspect-oriented systems or in a particular language like AspectJ. In order to validate a measure against the Representation Condition, first, the notion of the attribute being measured has to be defined. Second, an algorithm needs to be specified that defines how the values of the measure will be computed. The validation process consists of verifying that the values support the agreed notions about the measure. The Representation Condition is also highlighted during consideration of Construct Validity [7]. Construct Validity generalises from a measure or the algorithm of a measure to the concept of it, and asks whether the algorithm really captures the notion of a certain measure. The process of validation is then an investigation into how far the algorithm agrees with the attribute being defined for a certain measure.

4. EVALUATION

As a result of the discussion in section 3, the evaluation process will include the following steps for each measure: first, the attribute being measured by each measure will be presented. Second, the original algorithm will be stated and, third, the well-definedness of each algorithm will be investigated. Fourth, each measure will be validated by an investigation of the Representation Condition, i.e. an investigation into the notion and the algorithm involved. Usually this involves the question of whether a certain measure includes all the necessary coupling mechanisms that are associated with a certain notion of coupling. If necessary, changes to the definition of the measure will be suggested.

4.1 CAE (Coupling on Advice Execution)

Attribute. Aspects can cause control flow shifts so that advice is executed in the course of a program's execution. CAE [4] quantifies the amount of coupling caused by these shifts for a given class or aspect.

Algorithm. CAE counts the number of aspects containing advices possibly triggered by the execution of methods, advices or method intertype declarations, attribute and attribute intertype declarations in a given class or aspect.

Well-Definedness. The measure uses selected join point coupling mechanisms, i.e. all mechanisms that refer to executions: method, constructor or advice execution and attribute and attribute intertype declaration join point coupling. The granularity is aspect and every aspect will be counted only once. The locus of impact is export coupling. We interpret the phrase possibly triggered as a hint to the dynamic nature of pointcuts. Pointcut designators such as cflow cannot be determined at design time and have to be approximated. As far as intertype declarations are concerned, we assign them to the aspect they are defined in. Advice that is executed due to the fact that a method intertype declaration is executed will be considered coupled to the aspect that defines the intertype declaration. With these interpretations, it is possible to classify CAE as well-defined.

Validation. CAE counts only certain selected join point coupling mechanisms that can lead to the execution of advice. AspectJ supports more types of join points that can also cause the execution of advice, such as object initialization join points, exception handler join points, call join points and advice execution join points. A valid measure of coupling on advice execution needs to counts all of these join point coupling mechanisms.

Suggested Changes. In order to achieve theoretical validity we suggest including all types of join point coupling mechanism, i.e. all join points that can cause advice to be executed. The following table summarizes all the join points that may need to be counted by CAE:

<table>
<thead>
<tr>
<th>#</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>method execution join point</td>
</tr>
<tr>
<td>2</td>
<td>method call join point</td>
</tr>
<tr>
<td>3</td>
<td>constructor call join point</td>
</tr>
<tr>
<td>4</td>
<td>constructor execution join point</td>
</tr>
<tr>
<td>5</td>
<td>object initialization join point</td>
</tr>
<tr>
<td>6</td>
<td>attribute reference join point</td>
</tr>
<tr>
<td>7</td>
<td>attribute assignment join point</td>
</tr>
<tr>
<td>8</td>
<td>handler execution join point</td>
</tr>
<tr>
<td>9</td>
<td>advice execution join point</td>
</tr>
</tbody>
</table>

If the suggested changes are applied, we can classify CAE as a theoretically valid measure.

4.2 CIM (Coupling on Intercepted Modules)

Attribute. CIM [4] quantifies the explicit knowledge that an aspect has in its pointcuts of another class or interface that it crosses. It indicates the direct knowledge an aspect has of the rest of the system. High values indicate high coupling, due to high crosscutting.

Algorithm. CIM counts the number of classes or interfaces explicitly named in the pointcuts of a given aspect.

Well-Definedness. This algorithm can be mapped unambiguously to the criteria of the coupling framework. The type of connection is type pattern coupling, the granularity is class or interface and every class or interface will be counted once. The locus of impact
is import coupling. An aspect that selects a join point will be coupled with the executing object exposed by that join point. With this addition, we can classify CIM as well-defined.

Validation. CIM cannot be validated successfully. This measure makes the assumption that an explicitly mentioned class or interface in a pointcut always leads to crosscutting or to crosscutting with the mentioned class or interface. This is not always the case. Consider the following examples:

```java
// Example code snippet
pointcut execMethods() : execution( void *.*add() )
    && execution( void Manager.add() );

pointcut callMethods() : call( void Manager.add() );
```

Figure 1. Type pattern based coupling.

Although the Manager class is explicitly mentioned in both pointcuts, it is excluded from the set of join points in the first example and does not lead to crosscutting with the Manager class. In the second example, Manager is also explicitly mentioned, but it also does not lead to crosscutting with the Manager class. A minor problem is that the mere definition of a pointcut is not enough to crosscut base code. It also needs to be referred to in at least one piece of advice. As a result, the given algorithm does not measure the attribute defined.

Suggested Changes. The problems with this measure can be resolved in two ways. A first approach would be alter the attribute (and the name) so that CIM only quantifies the explicit knowledge an aspect has of the rest of the system, regardless whether this knowledge leads to crosscutting. An alternative approach would be to alter the algorithm to count only explicitly mentioned classes or interfaces that in fact contribute to crosscutting and that are referred to in at least one piece of advice. With one of these changes, CIM can be considered a theoretically valid measure.

### 4.3 CMC (Coupling on Method Call)

**Attribute.** CMC [4] is based on Chidamber and Kemerer’s CBO measure. CMC focuses on method calls only and quantifies the amount of coupling due to the call of methods or method intertype declarations of other classes or aspects.

**Algorithm.** CMC counts the number of classes or aspects that declare methods which are possibly called by a given class or aspect, including those methods that have been introduced by intertype declarations.

**Well-Definedness.** The type of connection is method execution, the granularity is class or aspect and every class or aspect will be counted once. The locus of impact of CMC is import coupling. We associate intertype declarations with the aspect that defines them and we associate inherited methods with the class that defines them. We interpret the phrase possibly called as potential coupling due to polymorphism. If a polymorphic method is called, we associate the calling class with all possible classes that could be called due to polymorphism. With these additions, we can consider CMC to be well-defined.

**Validation.** The theoretical validity of this measure depends to a great extent on how much the attribute that CMC measures is based on CBO. The attribute that CBO measures is coupling between objects: two objects are coupled if and only if one of them acts upon the other. X is said to act upon Y if the history of Y is affected by X, where history is defined as the chronologically ordered states that a thing traverses in time [5]. In terms of software, acting upon is defined as calling an object’s method or accessing an object’s attribute. Obviously, CMC does not support the entire attribute that CBO measures, since it only considers method calls. Another deviation from the CBO measure is the fact that CMC only counts import coupling whereas the acting upon relationship implies import and export coupling. We draw the following conclusion: if CMC is modeled on CBO it should count import and export coupling in order to be a theoretically valid measure. If it is not modeled on CBO, the current definition of CMC can be considered theoretically valid.

Suggested Changes. If CMC is based on CBO, we suggest changing the algorithm to include import and export coupling.

### 4.4 CFA (Coupling on Field Access)

**Attribute.** CFA [4] is designed similarly to CMC. Where CMC measures coupling due to method calls, CFA measures coupling due to attribute accesses.

**Algorithm.** CFA counts the number of classes, interfaces or aspects declaring attributes that are accessed by a given class or aspect.

**Well-Definedness.** The type of connection is attribute access, the granularity is class, interface or aspect and every class, interface or aspect will be counted only once. Just like CMC, we associate methods and attributes with the class or aspect that defines them. With these additions we consider CFA to be well-defined.

**Validation.** The theoretical validity of this measure follows the same argument as for CMC. If CFA is modeled on CBO, it should consider import and export coupling to be theoretically valid. If it is not modeled on CBO, it can be considered theoretically valid without any changes. This consideration is important if values of CBO are compared with the sum of the values of CMC and CFA. Since they do not measure the same attribute, a comparison might be problematic.

Suggested Changes. If CFA is based on CBO, we suggest changing the algorithm to include import and export coupling.

### 4.5 CDA (Crosscutting Degree of an Aspect)

**Attribute.** CDA [4] captures the overall crosscutting impact that an aspect has on the rest of the system through pointcuts and intertype declarations. The difference between CDA and CIM gives the number of modules that are affected by an aspect without being referenced explicitly by the aspect.

**Algorithm.** CDA counts the number of classes or aspects affected by the pointcuts and by the intertype declarations of a given aspect.

**Well-Definedness.** The locus of impact is import coupling, the granularity is class or aspect and every class or aspect will be counted only once. We interpret the crosscutting degree of an aspect as being dynamic and static crosscutting. A class a or an aspect b is affected by an aspect c, if the aspect c defines join points whose executing object can be associated with either class a or aspect b. The set of join points can only be approximated at
design time. A class a is also affected by an aspect b if the aspect defines an intertype declaration for class a. With these additions, we consider CDA well-defined.

Validation. CDA can be considered a theoretically valid measure since it considers both dynamic and static crosscutting.

As far as the difference between CDA and CIM is concerned, it does not indicate the number of modules that are affected by an aspect without being referenced explicitly. Since CDA considers intertype declarations, which CIM neglects, it counts modules which are affected by an aspect but which are referenced explicitly. The degree of generality intended by CDA can only be maintained if it considers modules affected by dynamic and not by static crosscutting.

4.6 Summary
The following table gives an overview of the measures and validation status. For CMC and CFA two validation states are given. As discussed, these refer to the extent to which they are based on the underlying CBO measure. We consider CDA to be a valid measure without any changes.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Validation status before changes</th>
<th>Validation status after changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAE</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CIM</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CMC</td>
<td>Yes/No</td>
<td>Yes</td>
</tr>
<tr>
<td>CFA</td>
<td>Yes/No</td>
<td>Yes</td>
</tr>
<tr>
<td>CDA</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

5. Conclusion
In this paper, we investigated the theoretical validity of five coupling measures and identified areas of improvement for four of the five measures. In two cases the improvements depend on how the attribute being measured is defined. We also suggested changes where necessary and conclude that with these changes all five measures can be considered theoretically valid. In particular, we conclude from our research that measures like CIM can now be applied with more confidence in a software evolution process.

The coupling framework used in this paper describes decisions that need to be considered when a measure is being defined. It served as a tool to explore the expressiveness of all five measures with regard to the criteria defined in the framework. We investigated the well-definedness of the measures and gave our interpretation of each in terms of this framework. For example, we followed the principle to assign methods and attributes to those classes and aspects that define them. These decisions are important when inheritance or intertype declarations are involved, because they greatly influence a measure: CMC counts method calls, but if a method being called is an intertype declaration then there is a coupling relationship towards the aspect that defines the method and not towards the class the method was defined for. Our interpretation of each measure highlights the degrees of freedom that designers of coupling measures need to consider.

A theoretical evaluation of measures can point to critical areas. We have shown that CMC and CFA do not cover the same attribute that CBO measures and that a comparison of CBO values with the total of CMC plus CFA is likely to be problematic. Only if CMC and CFA are defined in a similar way to CBO can comparability be guaranteed.

Future work will include the formal definition of aspect-oriented coupling measures and an investigation into the validity of other sets of coupling measures. Also, we would like to work towards an empirical validation of the measures discussed in this paper.

6. Acknowledgment
We would like to thank all anonymous reviewers for their valuable comments.

7. References