On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework

Eduardo Figueiredo[1], Claudio Sant'Anna[2], Alessandro Garcia[1],
Thiago T. Bartolomei[3], Walter Cazzola[4], and Alessandro Marchetto[5]

[1]Computing Department, Lancaster University, United Kingdom
[2]Computer Science Department, Pontifical Catholic University of Rio de Janeiro – PUC-Rio, Brazil
[3]Generative Software Development Lab, University of Waterloo, Canada
[4]Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, Italy
[5]Fondazione Bruno Kessler - IRST, Trento, Italy

{e.figueiredo, a.garcia}@lancaster.ac.uk, claudios@inf.puc-rio.br, ttonelli@uwaterloo.ca, cazzola@dico.unimi.it, marchetto@fbk.eu

Abstract

Aspect-oriented design needs to be systematically assessed with respect to modularity flaws caused by the realization of driving system concerns, such as tangling, scattering, and excessive concern dependencies. As a result, innovative concern metrics have been defined to support quantitative analyses of concern's properties. However, the vast majority of these measures have not yet being theoretically validated and managed to get accepted in the academic or industrial settings. The core reason for this problem is the fact that they have not been built by using a clearly-defined terminology and criteria. This paper defines a concern-oriented framework that supports the instantiation and comparison of concern measures. The framework subsumes the definition of a core terminology and criteria in order to lay down a rigorous process to foster the definition of meaningful and well-founded concern measures. In order to evaluate the framework generality, we demonstrate the framework instantiation and extension to a number of concern measures suites previously used in empirical studies of aspect-oriented software maintenance.

1. Introduction

Designing maintainable aspect-oriented software requires that software developers reason about the modularity of driving system concerns as the system evolves. With the emergence of aspect-oriented (AO) software development [16], there is an increasing awareness that observable phenomena relative to evolving system concerns might be the key factors to the deterioration of system maintainability, such as increasing concern tangling and scattering [16]. These observations provide classical indicators on when to use AO composition mechanisms [16]. On the other hand, outcomes of recent empirical studies have pointed out that the use of AO techniques can: (i) increase the number of undesirable concern couplings [12, 15], and (ii) decrease the cohesion of modules realizing a certain concern [14]. These concern modularity flaws make the change and removal of involved concerns error prone [11, 12] and lead to the manifestation of ripple effects [15].

Concern-driven design anomalies cannot be straightforwardly detected with conventional module-oriented metrics [12, 14, 15], such as Chidamber and Kemerer metrics [7]. As a result, a growing number of concern measures have been proposed in the literature [6, 8, 9, 18, 21, 22]. Their common goal is the association of quantification of concern properties with their impact on modularity flaws. A concern is any consideration that can impact the implementation of a program [19]. Concern measures lead to a shift in the measurement process: instead of quantifying properties of a particular module, they quantify properties of one or multiple concerns with respect to the underlying modular structure. Even though the usefulness of concern measures is paradigm-agnostic [8, 22] they have been consistently used to support the maintainability assessment of AO designs [9, 18] and their comparison with OO designs [11, 12, 14, 15].

However, the area of concern measurement is still in its infancy and it suffers from not subsuming to a unified measurement framework. The terminology used in existing definitions of concern measures is diverse and ambiguous by nature. Their definitions make it not clear the target level of abstraction, ranging from architecture-level to implementation-specific metrics, and the target concern modularity property. They rely on the jargon of specific research groups, thereby hampering: (i) the process of instantiating, comparing, and theoretically

validating concern measures, (ii) their adoption in academic and industry settings, (iii) independent interpretation of the measurement results, and (iv) replication of empirical studies using concern measures.

In this context, this paper presents a concern-oriented framework that supports the instantiation and comparison of concern measures. The framework subsumes the definition of a core terminology (Section 3) and criteria (Section 4) in order to foster the definition of meaningful and well-founded concern measures. Before presenting the measurement framework, we discuss the limitations of the state-of-the-art on concern measurement (Section 2). To evaluate the proposed framework's generality we have demonstrated the framework instantiation and extension to a number of concern measures (Section 5). These concern measures have been used in empirical studies on maintainability of AO software. Hence, we also discuss how the proposed measurement framework can help to point out limitations on the used measures.

2. Concern Measurement

This section presents a comprehensive survey and critical review of existing measurement frameworks (Section 2.1) and concern measures (Section 2.2) for AO systems. Section 2.3 summarises the shortcomings of existing concern measurement approaches.

2.1. Measurement Frameworks

Measurement frameworks have been proposed to validate [17], compare and instantiate measures [1, 3, 4]. Kitchenham *et al.* [17] defined a measurement framework that identifies the elementary components for measures validation. However, these components are generic and not tailored to the context of concern measurement. Moreover, this framework does not target the instantiation of new concern metrics and their comparison. For instance, it does not define criteria for selecting the granularity of concern-related elements being measured or for restricting the domain of such elements [1, 3, 4].

To cope with metrics definition support, Briand *et al.* proposed measurement frameworks for coupling [3] and cohesion [4] in object-oriented systems. As these frameworks do not take aspect-oriented constructs into consideration, Bartolomei *et al.* [1] extended the coupling framework to deal with abstractions and new composition mechanisms supported by aspect-oriented programming. These frameworks provided formalisms for expressing coupling and cohesion measures in a fully consistent and operational manner. Besides, they aim at comparing measures, integrating measures which examine the same attributes in different ways, and supporting the definition of new coupling and cohesion measures. On the other hand, none of the aforementioned frameworks can be tailored to the measurement of driving design concerns.

In this context, although a large number of different concern measures have been proposed [6, 8, 9, 18, 21, 22], there is a lack of standard terminology and formalism, leading to definitions of concern measures which are either ambiguous or difficult to understand. The extension of existing measurement frameworks to cope with concern measures is not straightforward since they need to be adapted in a number of ways. For instance, a concern-aware measurement framework has to specify a set of criteria regarding the projection of concerns onto the system modular structure, which is not the focus of existing measurement frameworks.

2.2. Concern Measures

This section presents a survey of existing concern measures [6, 8, 9, 18, 20, 21, 22]. All analysed measures have a common underlying characteristic that distinguishes them from conventional modularity measures [7]: they capture information about concerns traversing one or more structural component. Each concern measure of this section is presented in terms of a brief definition and an example. As far as the example is concerned, we rely on an OO design slice of a product line for mobile device applications [11], presented in Figure 1. This figure shows a partial class diagram realizing the Chain of Responsibility (CoR) design pattern [13] implemented in the product line. The concern measures are computed based on projecting a concern onto structural elements. For instance, in order to quantify the measurement attributes of the CoR pattern, elements realising it are shadowed in the design of Figure 1.

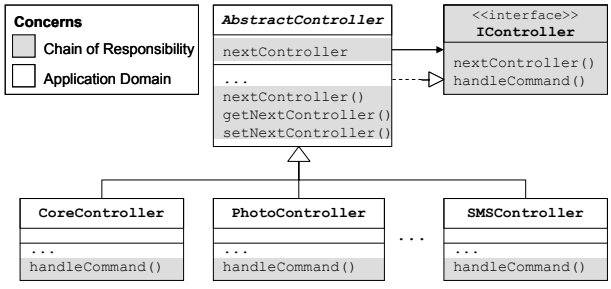

Figure 1. Concern projection of the CoR design pattern.

Concern Measures by Sant'Anna *et al*. Sant'Anna and his colleagues [21] defined three concern measures which quantify the diffusion of a concern over components, operations, and lines of code. *Concern Diffusion over Components (CDC)* and *Concern*

Diffusion over Operations (CDO) [21] measure the degree of concern scattering at different levels of granularity. CDC counts the number of classes and aspects related to a concern whereas CDO counts the number of methods and advices. Figure 1 shows that behaviour related to the CoR pattern is spread over five components (CDC) and nine operations (CDO). In another work [20], Sant'Anna *et al.* tailored CDC and CDO for measuring concern scattering in the system architectural description. For instance, CDC counts the number of architectural components instead of classes.

In addition to CDC and CDO, these authors define *Concern Diffusion over Lines of Code (CDLOC)* which aims at computing the degree of concern tangling. This metric counts the number of concern switches (Figure 2) for each concern through the lines of code [21]. Figure 2 shows the code regarding the CoR pattern shadowed in the `AbstractController` class. In this class, there are four concern switches between CoR and other concerns. Hence, the value of the CDLOC metric for this concern is four.

```
public class AbstractController          concern switch
               implements IController {
  private IController nextController;
                                         concern switch
  ...
                                         concern switch
  public void nextController(Command command) {
    if (handleCommand(command) == false)
      getNextController().handleCommand(command);
  }
  public IController getNextController() {...}
  public void setNextController(...) {...}
                                         concern switch
}
```

Figure 2. Projection in the *AbstractController* class.

Concern Measures by Ducasse *et al*. Ducasse *et al*. [8] defined a generic technique, called Distribution Map, to analyse *properties* of the system. Based on this technique, they describe four concern measures. In their approach, boxes can represent the program structure (Figure 3). For instance, large rectangles, called *partitions*, can be used to represent classes whereas small squares can correspond to internal members of classes, i.e., methods and attributes. Besides, small squares are filled with the colour that represents their corresponding property. Figure 3 presents our running example (Figure 1) using Distribution Map. The grey squares represent methods and attributes that implement the CoR design pattern.

Four concern measures are proposed by Ducasse *et al*. [8]: *Size, Touch, Spread, and Focus*. The *Size* metric counts the number of small squares associated to a property. The *Touch* metric counts the relative size given in terms of the percentage of small squares realising a property. For instance, Figure 3 shows that the Size value of CoR is nine and the Touch value is 0.23 (9/39). *Spread* counts the number of partitions that contains shadowed squares. Note that, in our running example (Figure 3), Spread gives the same result (five) of the CDC metric proposed by Sant'Anna [21]. Finally, the *Focus* metric quantifies the closeness between a particular partition and the property.

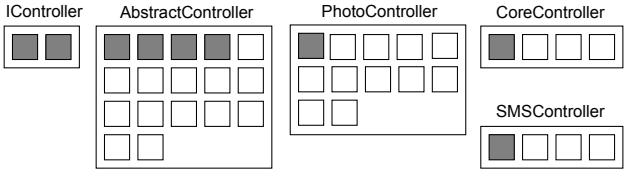

Figure 3. Distribution Map showing the CoR property.

Concentration, Dedication, and Disparity. Wong *et al*. [22] introduced three concern measures, namely *Disparity*, *Concentration*, and *Dedication*. Disparity measures how many "blocks" related to a feature are localised in a particular component. A feature is the functionality exercised by a given input and a block is a sequence of consecutive statements, so that if one element is executed, all are [22]. The more blocks in either a component *C* or a feature *F*, but not in both, the larger the disparity between *C* and *F*. Concentration and Dedication are also defined in terms of blocks and they quantify how much a feature is concentrated in a component and how much a component is dedicated to a feature [22].

Eaddy, Aho, and Murphy [9] presented two concern measures based on lines of code (LOC) that capture different facets of concern concentration and component dedication. In their work, *Concentration* is defined as the quotient of LOC in a component realising a concern by the total LOC realizing it in the system. Similarly, they also define the *Dedication* metric as quotient of LOC in a component realising a concern by the LOC of the component [9].

Basic Concern Measures. Lopez-Herrejon and Apel [18] define two of what they call basic concern measures: *Number of Features (NOF)* and *Feature Crosscutting Degree (FCD)*. The NOF metric counts the number of features in a system or subsystem. The FCD metric counts the number of classes that are crosscut by a feature. Besides, Ceccato and Tonella [6] present a concern measure, called *Crosscutting Degree of an Aspect (CDA),* which counts the number of modules affected by a given aspect. FCD [18] and CDA [6] have the same value (five) for the CoR concern in Figure 1.

2.3. Liabilities of Concern Measurement

A systematic analysis of existing concern measures points out fundamental divergences in the manner

concerns are addressed and quantified. One reason for the differences is the distinct objectives pursued by the authors. For example, to investigate maintainability indicators, Wong *et al*. [22] focus only on the implementation level while Sant'Anna *et al*. [20, 21] examined architecture design, detailed design and implementation. Besides, Ducasse *et al*. also uses a distinct representation of the system. We discuss in the following the significant differences observed among the concern measures and the liabilities associated.

Inconsistent Terminology. The non-uniform, distributed manner in which concern measures are often being defined and developed result in a lack of standard terminology. Many concern measures are expressed in an ambiguous manner which limits their use. For instance, the basic modularity unit is called (i) *component* by Sant'Anna *et al*., (ii) *partition* by Ducasse *et al*., and (iii) *module* by Ceccato and Tonella. Similarly, a *concern* is called *property* [8] and *feature* [22]. This also makes it difficult to understand how different concern measures relate to each other.

Incomplete Attributes. Existing concern measures target at quantifying four categories of concern properties: scattering, tangling, size, and closeness. Examples of metrics in these categories were presented in Section 2.2. For instance, closeness metrics quantify how close the intention of a design element is in relation to the concern. The metrics Disparity, Concentration, and Dedication, proposed by Wong *et al.* [22], are examples of closeness measures. However, existing metrics (Section 2.2) do not capture the whole spectrum of modularity properties associated with one or more concerns. For example, in spite of the wide recognition that coupling and cohesion play pivotal roles on the system maintainability [3, 4, 7, 15], there are no formal measures defined to quantify those concern's modularity properties.

Overlapping Measurement Goals. There are many different decisions that have to be made when defining a concern measure, such as with respect to the goal of the measure. Unfortunately, for many concern measures these decisions are not documented. It is, therefore, often unclear what the potential uses of existing measures are and how different concern measures could be used in a complementary manner. For instance, the definitions of CDC [21], Spread [8], FCD [18], and CDA [6] are very similar since all these measures have value of five for the CoR concern in the example of Section 2.2 (Figures 1 to 3). However, without a set of criteria to compare concern measures, similar measurement goals are not easy to spot and empirical studies, relying on those metrics, cannot be replicated in a reliable fashion [8, 17, 18, 21].

3. Terminology

This section presents a standard terminology which allows all existing concern measures to be expressed in a consistent and meaningful manner. We seek to define a terminology and formalism that is, as much as possible, language independent and extensible.

3.1. Concern-Oriented Meta-Model

Figure 4 presents a generic concern-oriented meta-model of the structural abstractions defined for an aspect-oriented system. It not only defines possible relations of concerns and the system's structure, but also subsumes key abstractions for module specifications. Each type of abstraction is alternatively called an *element*. Concerns can be realized by an arbitrary set of elements. An aspect-oriented system S consists of a set of components, denoted by $C(S)$. A component c has an interface, $I(c)$. Besides, each component c consists of a set of attributes, $Att(c)$, a set of operations, $Op(c)$, and a set of declarations, $Dec(c)$. The set of members of a component c is defined by $M(c) = Att(c) \cup Op(c) \cup Dec(c)$.

The reader should notice that for generality purposes, a component is a unified abstraction to both aspectual and non-aspectual modules. This decision makes the meta-model paradigm and language independent [5]. For example, a component represents either a class or an interface in Java programs, and a component is a class, an interface, or an aspect in AspectJ programs [16]. An operation o consists of a return type, $Rt(o)$, a set of parameters, $Par(o)$, a pointcut expression, $PE(o)$, and a set of statements, $St(o)$. A declaration d can also have a pointcut expression, $PE(d)$.

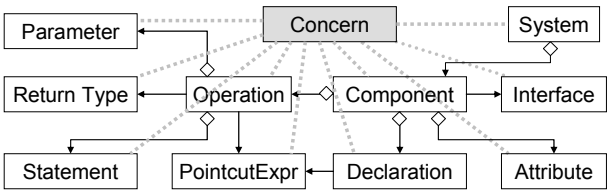

Figure 4. Abstract meta-model of aspect-oriented systems.

On top of this structure, we can define *concerns* which are selections of any type of elements as presented in Figure 4. A concern is not an abstraction of a modelling or programming language, such as components and operations. However, a concern can be considered as an abstraction which is addressed by those elements that have the purpose of realising it. An example concern is a software requirement. In this way, in order to have concern-based measurement it is necessary to assign for each structural element (e.g., component) the concerns it is realising. The set of

concerns addressed by the system *S* is defined as *Con(S)*. Furthermore, a concern *con* can be realised by a set of components, *C(con)*, a set of attributes, *Att(con)*, a set of operation, *Op(con)*, or a set of declarations, *Dec(con)*. The set of members that implement a concern *con* is defined as $M(con) = Op(con) \cup Att(con) \cup Dec(con)$.

3.2. Components and Connections

Connection is defined as a dependency relationship between two elements, where one element, called *Server*, provides a service to another element, the *Client* [7]. Two types of connections can be defined: *explicit connection* and *implicit connection*. For instance, an explicit connection of a component *c*, *EC(c)*, is caused by elements of *c* calling an operation or accessing an attribute of other components. On the other hand, an implicit connection of a component *c*, *IC(c)*, is caused by a join point being reached or by a handler catching an exception. The set of connections of a component *c* is defined as $CC(c) = EC(c) \cup IC(c)$.

For convenience purpose, we define that the set of all components of a system, *C(S)*, can be partitioned in 3 subsets: Application, *AC(S)*, Framework, *FC(S)*, and Library, *LC(S)*. The set of all components, attributes, operations, and declarations of a system *S* is denoted by *AC(S)*, *Att(S)*, *Op(S)*, and *Dec(S)*, respectively. Furthermore, components may participate on inheritance relationships. For a given component *c*, the following sets are defined: (i) *Ancestors(c)* - all recursively defined parents; (ii) *Parents(c)* - the directly declared parents; (iii) *Children(c)* - the directly derived children, and (iv) *Descendants(c)* - the recursively derived children. Because of inheritance relationships between components, the following member sets are defined for a component *c*:

- $M_{NEW}(c)$ - Members newly implemented in the component (not inherited).
- $M_{VIR}(c)$ - Members declared as virtual.
- $M_{OVR}(c)$ - Members inherited and overridden.
- $M_{INH}(c)$ - Members inherited and not-overridden.

3.3. Language Instantiation

The aforementioned structures are abstract enough to be instantiated for different modelling and programming languages. This section provides a brief illustration on how our generic meta-model (Sections 3.1 and 3.2) can be instantiated to languages targeting different levels of abstraction. We use an architecture modelling language for the component-and-connector view [2], and two programming languages, namely Java and AspectJ [16]. We have chosen AspectJ as a representative of aspect-oriented programming languages, and Component-Connector models as an example of modelling language used in early design stages. Table 1 defines how a subset of elements of the meta-model can be mapped to these three languages. A blank cell means that the abstraction is not implemented by any elements of the language. For example, declarations are only valid for aspect-oriented languages such as AspectJ (Table 1).

Table 1. Meta-model instantiation.

Element	Component-Connector [2]	Java	AspectJ [16]
System	Architecture Design	Java System	AspectJ System
Concern	Architecture Concern	Concern	Concern
Component	Architecture Component	Class and Interface	Class, Interface, and Aspect
Interface	Architecture Interface	Method Signature	Method Signature
Attribute	-	Class Variable and Field	Class Variable, Field, and Inter-type Attribute
Operation	Operation and Event	Method and Constructor	Method, Constructor, Inter-type Method and Constructor, and Advice
Declaration	-	-	Pointcut and Declare Statement

4. The Framework Structure

This section presents the concern measurement framework which relies on the terminology presented in the previous section. In order to define a concern measurement framework, we have analysed and, whenever it is feasible, tried to maximise the use of criteria defined in existing measurement frameworks for OO [3, 4, 17] and aspect-oriented [1] systems. Moreover, we have identified recurring characteristics of existing concern measures proposed in the literature (Section 2.2). From these analyses a set of criteria that should be considered when comparing concern measures or developing a new measure has emerged. The next subsections provide details on each criterion.

4.1. Entities of Concern Measurement

One of the goals of concern measurement is to capture characteristics of concerns, such as size, and manipulate them in a formal way. The entity of measurement determines the elements that are going to be measured. When we choose a certain element type as the entity of measurement, it means that we are interested in characteristics of this type. For example, if we choose component, it means we are interested in concern-related information about components.

Criterion Instantiation. Usually concern measures use concerns as the entity of measurement, but other selections are also possible. For example, the metrics Concentration and Dedication [22] have distinct entities

of measurement. While Concentration has concerns as entities, the Dedication metric is interested in information of components. Although all elements in the meta-model of Figure 4 may be selected in this criterion, the most common entities of concern measurement are: (i) System, (ii) Concern, (iii) Component, (iv) Attribute, and (v) Operation.

4.2. Concern-Aware Attributes

Attributes are the properties that a concern (or an entity) possesses. For a given attribute, there is a relationship of interest in the empirical world that we want to capture formally in the mathematical world [17]. For example, if we observe two concerns we can say that one *is more spread than* the other. A concern measure allows us to captures the "is more spread than" relationship and maps it to a formal system, enabling us to explore the relationship mathematically. An entity possesses many attributes, while an attribute can qualify many different entities [17]. These relationships can be confirmed by example. To see that an entity can have many attributes, consider a concern as an entity of measurement which can exhibit attributes such as scattering and tangling. In addition, an attribute may apply to one or more different entities. For example, size can apply to several different software entities, such as components or concerns.

Criterion Instantiation. In the attribute selection, we may choose any property of the entity that we want to measure. In fact, existing concern measures in the literature cover a vast range of measurement attributes. For example, the measures proposed by Sant'Anna [21] (Section 2.2) target *scattering* (CDC and CDO) and *tangling* (CDLOC). On the other hand, Wong's measures Concentration, Dedication and Disparity [22] assess the *closeness* between components and concerns. Possible values of a measurement attribute include: (i) Scattering, (ii) Tangling, (iii) Closeness, (iv) Coupling, (v) Cohesion, and (vi) Size.

4.3. Units

A concern measurement unit determines how we measure an attribute. An attribute may be measured in one or more units, and the same unit may be used to measure more than one attribute [17]. For example, concern size might be measured by counting either the lines of code or the number of components which implement it. Similarly, the number of components may be used to measure the attributes size and scattering of a concern.

Criterion Instantiation. The concern measures discussed in Section 2.2 have heterogeneous units of measurement. For instance, the metrics CDC, CDO and CDLOC [21] count number of components, operations, and concern switches, respectively. On the other hand, none of the concern measures proposed by Wong *et al*. [22] have units of measurement. We may choose any countable elements as measurement units, for example, (i) Concerns, (ii) Components, (iii) Operations, (iv) Attributes, and (v) Lines of Code.

4.4. Concern Measurement Values

A measured value cannot be interpreted unless we know to what concern it applies, what attribute it measures and in what unit. Some concern measures, such as those ones proposed by Wong *et al*. [22], do not specify any unit of measurement. The lack of units in the metrics investigated in Section 2.2 is a result of equations which divide two values with the same unit, e.g., the Touch metric divides members (implementing a concern) per members (of the system).

Criterion Instantiation. We expect concern measures to be defined over a set of permissible values. For instance, the CDC measure value [21] is defined on the non-negative integers. Besides, values of Concentration, Dedication and Disparity [22] are bounded in the range of 0 and 1, inclusive. A set of permissible values may be *finite* or *infinite*, *bounded* or *unbounded*, *discrete* or *continuous*.

4.5. Concern Granularity

The granularity of a concern measure is the level of detail at which information is gathered. This criterion is determined by two factors: (i) *element granularity* - what elements are to be measured, and (ii) *element distinction* - how the elements are counted. The element granularity factor specifies what is being counted, i.e., which elements aggregate values. For example, when we say "the number of concerns of a component that..." the entity is component but what we are counting (granularity) is concern. The element distinction factor defines whether we ignore duplicated elements or not when re-applying the concern measure to a different goal. For instance, it is allowed to count the same component for different concerns in a given measure (e.g., CDC). The difference between element granularity and measurement unit is clear because all measures have to define an element which is being counted. However, the measurement unit can either be omitted or be coarser or finer than the granularity (e.g., giving weights to elements).

Criterion Instantiation. The survey of concern measures in Section 2.2 points out very heterogeneous options for element granularity and element distinction.

For instance, element granularity ranges from lines of code (e.g., CDLOC) to components (e.g., CDC) and concerns (e.g., NOF). There are also metrics which allow elements to be counted more than once, such as CDC and CDO [21], and metrics that allow each element to be counted only once, such as Size and Touch [8]. Possible values of granularity are, for example: (i) Concern, (ii) Component, (iii) Operation, (iv) Attribute, and (v) Lines of Code. Element distinction has to be "yes" (count only once) or "no" (count all possible occurrences).

4.6. Domain

There are two pertinent issues of domain: how to account for inheritance and how to restrict types of elements for being measured. Regarding inheritance, concern measures have to define how components related via inheritance should behave. For instance, inherited operations should be excluded or included in a given concern measure. In the domain criterion, measures have also to define the types of elements that should be accounted for. For example, they might strictly count elements of the application domain (excluding classes of the framework and libraries).

Criterion Instantiation. The possible values for inheritance are "yes" (account) or "no" (ignore). Besides, if inheritance is taken into consideration measures have to specify which set of elements should be included: Ancestors, Parents, Children, or Descendants. We may restrict elements in the domain based on: Application, Framework, and Libraries. Other categorizations are also conceivable. However, we suggest using a categorization scheme where the decision, into which category a given element belongs, can be made automatically. Based on the original definitions of the concern measures, we cannot decide if the metrics take into consideration inherited members or not and how they restrict the domain. Therefore, we acknowledge that the investigated concern measures do not consider inheritance and that they are applied to application elements only.

4.7. Concern Projection

Concern measures must specify a measurement protocol that must be followed in the empirical studies; otherwise these empirical studies cannot be replicated and replication is the basis of scientific validation [17]. A measurement protocol must be unambiguous and must prevent invalid measurement procedures such as double counting. For instance, one of the most sensitive parts in concern measurement is the projection of abstract concerns onto elements in the design. At least two definitions have to do with this projection: (i) what the concerns are and (ii) onto what artefacts the concern is going to be mapped. Besides, concern measures have to specify whether they allow overlapping of concerns or not. For instance, it is possible two different concerns be projected onto the same operation.

Criterion Instantiation. Most of the concern measures do not clearly state which sorts of concerns they are interested in. When the specification of a concern is not clear in the measure definition, we assume it is applied to all kinds of concerns. Typical concerns in a software project include [19]: (i) Features, (ii) Non-Functional Requirements, (iii) Design Idioms, and (iv) Implementation Mechanisms (e.g., caching). Examples of design elements which can be mapped to concerns are (i) Components, (ii) Operations, (iii) Attributes, and (iv) Lines of Code. All three concern measures proposed by Sant'Anna (i.e., CDC, CDO, CDLOC) are applied to all kinds of concerns. However, they map a concern to different artefacts: CDC requires a mapping of concerns to components, CDO to operations, and CDLOC to lines of code. Of course, a mapping to finer level such as operations can easily be extended to coarser level like classes. In other words, the CDC metric also accept a mapping of concerns to operations or lines of code.

5. Evaluation

This section introduces an evaluation of the concern measurement framework in three steps. First, we illustrate the application of our framework by describing and formalising a concern measure (Section 5.1). Second, we systematically apply the framework to existing concern measures in order to define and compare them in an unambiguous and fully operational manner (Section 5.2). Third, we identify extensions to the framework based on the instantiation of new concern measures for different assessment purposes (Section 5.3).

5.1. Application of the Framework

This section demonstrates the application of our concern measurement framework to the metric Concern Diffusion over Operations (CDO) [11, 14, 15, 21]. We first describe the metric textually using our standard terminology (in *italic*). Then, we go through the whole process of analysing and selecting each criterion and, finally, we derive a formal definition.

Description. Concern Diffusion over Operations (CDO) counts the number of *operations* whose main purpose is to contribute to the implementation of a *concern*. In addition, it counts the number of methods, constructors, and advice that access any primary *component* of the

concern by accessing its *attributes*, calling its *operations* or using it in *parameters*, *return types*, *declarations* and *statements*.

1. **Entity of Concern Measurement**. *Concern* is the entity of measurement for this metric.
2. **Attribute**. CDO quantifies *scattering* of a given concern over the operations of the system.
3. **Unit**. The unit is *number of operations*.
4. **Properties of Values**. Permissible values for this metric are not greater than *Op(S)* (*finite*), do not define any interval a priori (*unbounded*), and allow integers only (*discrete*).
5. **Granularity**. The granularity of elements that are being measured is *operation*.
6. **Domain**. It considers *application components* (not components in the framework or libraries) and *does not take inherited operations into account*.
7. **Concern Projection**. Concerns can be *anything* that directly contributes to the functionality of the system, i.e., which is materialised in the design or implementation. It requires projection of concerns onto *operations* (or finer grained artefacts). *Overlapping* of concerns is allowed.

Using the selected criteria and the concern terminology described in Section 3 we derive the following formal definition for CDO:

$$CDO = \left| \{ o \in (Op(c) \cap Op(con)) \mid c \in AC(S) \wedge con \in Con(S) \} \right|$$

Number of operations; Operations common to a component ***c*** and a concern ***con***; Component ***c*** belongs to the Application domain; Concern ***con*** belongs to the set of concerns of the system ***S***

5.2. Measures Instantiation and Comparison

In addition to CDO illustrated in Section 5.1, this section presents the instantiation of 14 concern measures in our framework. These measures were selected for several reasons. First, they were proposed by different authors in various research groups. Hence, they do not rely on a uniform terminology. Second, the selected measures present a very heterogeneous definition, apply to distinct abstract level ranging from architecture to implementation and quantify diverse assessment goals. Finally, these concern metrics have been applied in a number of maintenance case studies [8, 11, 12, 14, 15, 20] and they have been proved to be important internal quality indicators.

Table 2 shows the selected criteria for the sample set of concern measures. Rows list the concern measures and columns the framework criteria. It is interesting to note that concern metrics present heterogeneous values in some criteria, such as Attribute and Measurement Unit. For instance, existing concern measures in the literature cover at least four attributes of concerns, namely scattering, tangling, size, and closeness.

Table 2. Instantiation of concern measures

Concern Measures	1. Entity	2. Attribute	3. Unit	4. Values	5. Granularity and Distinct	6. Domain and Inheritance	7. Concern, Artefact and Overlapping
CDC [20, 21] CDA [6]	Concern	Scattering	Components	Finite, unbounded, discrete	Component, No	Application, No	All, Component, Yes
CDLOC [21]	Concern	Tangling	Concern Switches	Finite, unbounded, discrete (even values)	Line of Code, No	Application, No	All, Line of Code, No
Size [8]	Concern	Size	Members	Finite, unbounded, discrete	Member, Yes	Application, No	All, Member, No
Touch [8]	Concern	Size	None	Infinite, bounded, continuous	Member, Yes	Application, No	All, Member, No
Spread [8]	Concern	Scattering	Components	Finite, unbounded, discrete	Component, No	Application, No	All, Component, No
Focus [8]	Concern	Closeness	None	Infinite, bounded, continuous	Member, Yes	Application, No	All, Member, No
Disparity [22]	Concern, Component	Closeness	None	Infinite, bounded, continuous	Member, No	Application, No	Feature, Member, Yes
Concentration [22]	Concern	Closeness	None	Infinite, bounded, continuous	Member, No	Application, No	Feature, Member, Yes
Dedication [22]	Component	Closeness	None	Infinite, bounded, continuous	Member, No	Application, No	Feature, Member, Yes
NOF [18]	Component	Tangling	Concerns	Finite, unbounded, discrete	Concern, Yes	Application, No	Feature, Component, Yes
FCD [18]	Concern	Scattering	Components	Finite, unbounded, discrete	Component, No	Application, No	Feature, Component, Yes
Concentration [8]	Concern	Closeness	None	Infinite, bounded, continuous	Line of Code, No	Application, No	All, Line of Code, No
Dedication [9]	Component	Closeness	None	Infinite, bounded, continuous	Line of Code, No	Application, No	All, Line of Code, No

However, to the best of our knowledge, none of the existing concern measures target at other equally important concern-aware attributes, such as coupling and cohesion. Examples of concern measures addressing these attributes are presented in Section 5.3.

Table 2 also highlights the need of concern measures which contemplate other domains (criterion 6) rather than the Application one. For example, Briand *et al* [3] state a hypothesis that "*maintainability is influenced by dependencies on both stable (library) and unstable classes (application)*". To verify this hypothesis, one might use concern measures to evaluate the influence of library concerns in the design of the system. None of the concern measures in Table 2 deal with inheritance, which also indicates the lack of exiting concern measures covering some criteria.

As discussed in Section 2, the concern measures CDC, CDA, Spread and FCD have similar definitions. Actually, CDC and CDA have exactly the same instantiation in our framework. Although the framework confirms that similarity, it also helps to point out small differences among the concern measures. For instance, while CDC and Spread are applied to all sorts of concerns, the FCD metric is intended to features [18]. Besides, overlapping of concern is permitted in CDC and FCD, but it is not allowed in the definition of Spread due to constraints in the concern representation (Distribution Map [8]). Finally, the two concern measures Concentration and Dedication are both defined by Wong *et al.* [22] and Eaddy *et al.* [9]. In spite of their similarities, the granularity is member in the former and line of code in the latter (Table 2).

5.3. On the Framework Extensibility

This section evaluates whether the measurement framework accomplishes the definition of new concern measures to different goals. In this evaluation we attempt to create three new concern measures: Concern Sensitive Coupling (CSC), Lack of Concern-based Cohesion (LCC) and Dynamic Concern Diffusion over Components (dCDC). These concern metrics contemplate options (coupling, cohesion and dynamic issues) in the set of criteria which are not covered by existing concern measures. In the following, we describe these concern measures textually and present their formal definition. Table 3 shows the chosen options for each criterion in the measures' instantiation.

Concern Sensitive Coupling (CSC) quantifies the *number of server components* that a concern realised by a given *client component* is coupled to. In other words, CSC counts the number of *explicit connections* that are associated to the concern in a component.

$$CSC = \left| \{ r \in EC(c) \mid c \in (AC(S) \cap C(con)) \wedge con \in Con(S) \} \right|$$

Lack of Concern-based Cohesion (LCC) counts the number of *concerns* addressed by the assessed *component*. LCC is based on a similar metric defined in the architecture level [20].

$$LCC = \left| \{ con \in Con(c) \mid c \in AC(S) \} \right|$$

Dynamic Concern Diffusion over Components (dCDC) measures the number of *components* that are exercised (i.e., accessed, called, and instantiated) during the execution of a given *feature*.

$$dCDC = \left| \{ c \in (AC(S) \cap C(con)) \mid con \in Con(S) \} \right|$$

The instantiation of the three concern measures shown in Table 3 highlights the generality of our framework. Although being very heterogeneous, all three concern measures were able to be instantiated in the concern measurement framework. However, some minor extensions were required. For example, an annotation "dynamic" was attached to the measurement attribute of dCDC in order to make its dynamic nature explicit. Furthermore, a similar extension "efferent" was used in CSC. This annotation shows that the CSC metric counts afferent connections, i.e., where components are playing the *client* role.

6. Conclusion

This paper (i) presented a standard terminology and formalism; and (ii) provided a set of criteria for the comparison, evaluation, and definition of concern measures in aspect-oriented systems. Besides, it provided detailed guidance so that concern measures can be defined in a consistent and operational way. It also revisited the state-of-the-art, about which we draw the

Table 3. Instantiation of new concern measures

Concern Measures	1. Entity	2. Attribute	3. Unit	4. Values	5. Granularity and Distinct	6. Domain and Inheritance	7. Concern, Artefact and Overlapping
CSC	Concern	[Efferent] Coupling	Connections	Finite, unbounded, discrete	Component, No	Application, No	All, Statement, Yes
LCC	Component	Cohesion	Concerns	Finite, unbounded, discrete	Component, No	Application, No	All, Component, Yes
dCDC	Concern	[Dynamic] Scattering	Components	Finite, unbounded, discrete	Component, No	Application, Yes (M_{NEW}, M_{OVER}, M_{INH})	Feature, Component, Yes

following conclusions. There is a very rich body of ideas regarding the way to address concern measurement. However, some concern's modularity properties, such as coupling and cohesion, are not addressed and many concern measures have similar definitions and measurement goals.

The proposed terminology and formalism is abstract and language independent since the framework is intended to be extensible. By no means have we claimed that the set of framework extensions discussed in Section 5.3 are complete; further extensions might be necessary. In fact, the concern measurement framework proposed in this paper is a first step stone towards the well definition (and formalisation) of concern measures. As future work, we plan to explore the usefulness of the measurements derived by using our framework and the dynamic aspects of concern measurement.

7. Acknowledgements

This work is supported in part by the European Commission grant IST-33710 - Aspect-Oriented, Model-Driven Product Line Engineering (AMPLE), grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe). Eduardo is supported by CAPES - Brazil.

8. References

[1] Bartolomei, T.; Garcia, A.; Sant'Anna, C.; and Figueiredo, E. "Towards a Unified Coupling Framework for Measuring Aspect-Oriented Programs". *Proc. of Workshop on Software Quality Assurance (SOQUA)*. Portland, 2006.

[2] Bass, L., Clements, P., Kazman, R. "*Software Architecture In Practice*", 2nd ed., Addison-Wesley, 2003.

[3] Briand, L., Daly, J., and Wust, J. "A Unified Framework for Coupling Measurement in Object-Oriented Systems". *IEEE Transactions on Software Engineering*, 25(1), pp. 91-120, 1999.

[4] Briand, L.; Daly, J.; and Wust, J. "A Unified Framework for Cohesion Measurement in Object-Oriented Systems". *Proc. of the Software Metrics Symposium*, pp. 43-53, 1997.

[5] Bryton, S.; Brito e Abreu, F. "Towards Paradigm-Independent Software Assessment". *Int'l Conf. on the Quality of Information and Comm. Tech. (QUATIC)*, 2007.

[6] Ceccato, M. and Tonella, P. "Measuring the Effects of Software Aspectization". *In proceedings of the 1st Workshop on Aspect Reverse Engineering*, 2004.

[7] Chidamber, S. and Kemerer, C. "A Metrics Suite for Object Oriented Design". *IEEE Transactions on Software Engineering*, pp. 476-493, 1994.

[8] Ducasse, S.; Girba, T.; and Kuhn, A. "Distribution Map". *Proc. of the Int'l Conference on Software Maintenance (ICSM)*, pp. 203-212. Philadelphia, 2006.

[9] Eaddy, M.; Aho, A.; and Murphy, G. "Identifying, Assigning, and Quantifying Crosscutting Concerns". *Proc. of the Workshop on Assessment of Contemporary Modularization Techniques (ACoM)*, 2007.

[10] Fenton, N. "*Software Metrics: A Rigorous Approach*". London: Chapman-Hall, pp. 28-37, 1991.

[11] Figueiredo, E. et al. "Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability". *To appear in proceedings of the Int'l Conference on Software Engineering (ICSE)*, Leipzig, Germany, 2008.

[12] Filho, F. et al. "Exceptions and Aspects: The Devil is in the Details". *Proc. of the International Symposium on Foundations of Software Engineering (FSE)*, Portland, 2006.

[13] Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. "*Design Patterns: Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

[14] Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., and Staa, A. "Modularizing Design Patterns with Aspects: A Quantitative Study". *Proc. of the Aspect-Oriented Software Development Conference (AOSD)*, Chicago, 2005.

[15] Greenwood, P. et al. "On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study". *In proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. Berlin, Germany, 2007.

[16] Kiczales, G. et al. "An Overview of AspectJ". *In proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pp. 327-355, 2001.

[17] Kitchenham, B.; Pfleeger, S.; and Fenton, N. "Towards a Framework for Software Measurement Validation". *IEEE Transactions on Software Engineering*, vol. 21, no. 12, 1995.

[18] Lopez-Herrejon, R. and Apel, S. "Measuring and Characterizing Crosscutting in Aspect-Based Programs: Basic Metrics and Case Studies". *Proc. of the Int'l Conference on Fundamental Approaches to Software Engineering (FASE)*, 2007.

[19] Robillard, M., and Murphy, G. "Representing Concerns in Source Code". *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(1), 2007.

[20] Sant'Anna, C., Figueiredo, E., Garcia, A, Lucena, C. "On the Modularity of Software Architectures: A Concern-Driven Measurement Framework". *Proc. of the European Conference on Software Architecture (ECSA)*. Madrid, 2007.

[21] Sant'Anna, C. et al. "On the Reuse and Maintenance of. Aspect-Oriented Software: an Assessment Framework". *Proc of the Brazilian Symposium on Software Engineering (SBES)*, 2003.

[22] Wong, W.; Gokhale, S.; and Horgan, J. "Quantifying the Closeness between Program Components and Features". *Journal of Systems and Software*, 2000.