JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 7, September-October 2006

A Cohesion Measure for Aspects

Jean-François Gélinas, University of Quebec at Trois-Rivières
Mourad Badri, University of Quebec at Trois-Rivières
Linda Badri, University of Quebec at Trois-Rivières

Abstract

Aspect-Oriented Software Development is a promising new software engineering paradigm. It promotes, in particular, improved separation of crosscutting concerns into single units called aspects. AspectJ, the most used aspect-oriented programming language, represents an extension of Java. In fact, existing object-oriented programming languages suffer from a serious limitation in modularizing adequately crosscutting concerns. Many concerns crosscut several classes in an object-oriented system. Moreover, several metrics have been proposed in order to assess object-oriented software quality attributes. However, these metrics do not cover the new abstractions and complexity dimensions introduced by the aspect paradigm. As a consequence, new metrics must be developed to assess aspect-oriented systems quality attributes. Cohesion is considered as one of the most important software quality attributes. Cohesion refers to the degree of relatedness between members of a software component. We propose, in this paper, a new approach for aspect cohesion measurement based on dependencies analysis. We introduce several cohesion criteria taking into account aspects' features and capturing various dependencies between their members. We also propose a new aspect cohesion metric and compare it, using several case studies, to few existing aspect cohesion metrics.

1 INTRODUCTION

Aspect-Oriented Software Development (AOSD) is a promising new software engineering paradigm [Sabb04]. AspectJ, as an aspect-oriented programming language, represents an interesting extension of Java [Kicz01]. In fact, existing object-oriented programming languages suffer from a serious limitation in modularizing adequately crosscutting concerns [Kicz97, Kicz01]. Many concerns crosscut several classes in an object-oriented system. Crosscutting is a structure that goes beyond hierarchy as stated in [Kicz04]. The code related to a crosscutting concern is generally duplicated within several classes in an object-oriented system. Consequently, these classes would be difficult to understand, maintain and reuse. Aspect-Oriented Programming (AOP) deals with scattered and tangled code related to crosscutting concerns. It particularly promotes improved separation of crosscutting concerns into single units called aspects.

Several metrics have been proposed in the literature in order to assess quality attributes (complexity, coupling, cohesion, etc.) in object-oriented systems [Badr95, Badr04, Biem95, Chid94, Hend95, Hend96, Hitz95]. Metrics have become essential

Cite this article as follows: J.F. Gélinas, M. Badri, L. Badri: "A Cohesion Measure for Aspects", in *Journal of Object Technology*, vol. 5, no. 7, September - October 2006, pp. 97 - 114
http://www.jot.fm/issues/issue_2006_09/article5

tools in some disciplines of software engineering [Pres01]. However, existing object-oriented metrics are not adequate to capture all the features of aspect-oriented software as mentioned in many papers [Cecc04, Sant03, Zaca03, Zhao03, Zhao04]. AOSD introduces new abstractions and complexity dimensions to software engineering. As a consequence, new metrics taking into account aspects' features must be developed to assess aspect-oriented systems quality attributes.

High cohesion is a desirable property of software components. Cohesion is an underlying goal to continually consider during the design process [Larm04]. It is widely recognized that highly cohesive components tend to have high maintainability and reusability [Biem95, Bria98, Chae00, Li93]. Cohesion refers to the degree of relatedness between component members. Grady Booch describes high functional cohesion as existing when the elements of a component all work together to provide some well-bounded behavior [Booc94]. The cohesion degree of a component is high when it implements a single logical function. It should be difficult to split such a component. Cohesion can be used to identify the poorly designed components. A component with low cohesion may have disparate and non-related members. Such a component has probably been assigned many unrelated responsibilities. Design elements with low cohesion may be considered for restructuring. AOP promotes separating crosscutting concerns and addresses some of these problems. We can expect, for example, cohesion improvement in classes since the code corresponding to crosscutting concerns is implemented in a more modular way. However, we believe that aspect responsibility assignment should also follow these well-tested software engineering principles. An aspect must express a concern (or a part of a concern) in a cohesive manner. Like classes assigned disparate responsibilities, an aspect implementing several and disparate concerns will present a low cohesion. We can expect that such an aspect will be difficult to understand, to test, to reuse and to maintain.

Only few papers addressing aspect cohesion have been published in the literature. We discuss, in the following sections, some aspects' features and particularly the various dependencies that exist between their members. Based on these dependencies, several aspect members' connection criteria will be introduced. These criteria will define our foundations for aspect cohesion and the metric we propose for its assessment. The proposed metric is evaluated using several concrete case studies and compared to the few existing aspect cohesion metrics.

The rest of the paper is structured as follows: Section 2 summarizes the few related works. Section 3 discusses the different dependencies between aspect members. We present, in section 4, several aspect members' connection criteria. Section 5 presents our approach for aspect cohesion and a new metric allowing its measurement. In section 6, several small case studies will be presented illustrating our proposal and comparing it to the other related approaches. Section 7 presents the results of the first experimental study that we conducted in this field. Finally, section 8 gives some conclusions and future work directions.

2 RELATED WORK

Presently, few papers addressing aspect-oriented programs quality have been published in the literature [Cecc04, Sant03, Zaca03, Zhao03, Zhao04]. Zhao and Xu's approach [Zhao04] is the first proposal in the field of aspect cohesion measurement. It is based on a dependency model for aspect-oriented software that consists of a group of dependency graphs. According to Zhao and Xu's approach, cohesion is defined as the degree of relatedness between attributes and modules. Zhao and Xu present, in fact, two ways for measuring aspect cohesion based on inter-attributes (γ_a), inter-modules (γ_m) and module-attribute (γ_{ma}) dependencies. The first way suggests that each measurement (γ_a,γ_m,γ_{ma}) works as a field. Therefore, aspect cohesion for a given aspect A is defined as a 3-tuples $\Gamma(A) = (\gamma_a,\gamma_m,\gamma_{ma})$. They also suggest another way for cohesion measurement where each facet could be integrated as a whole with β parameters. Aspect cohesion is then expressed as:

$$\Gamma(A) = \begin{cases} 0 & n = 0 \\ \beta^* \gamma_m & k = 0 \text{ and } n \neq 0 \\ x & \text{Others} \end{cases}$$

Where $x = \beta_1^* \gamma_a + \beta_2^* \gamma_m + \beta_3 * \gamma_{ma}$, k the number of attributes and n the number of modules in aspect A.

The choice of the evaluation method (in tuples or as a whole) and/or the parameters weight β_1, β_2, and β_3 is arbitrary. In addition, some relationships definition (inter-attributes cohesion in particular) and their consideration in aspect cohesion measurement are difficult to capture. In general, this approach suggests a complex way to measure aspect cohesion that may be problematic to use in a real development context and particularly in the case of real scalable aspect-oriented software. Generating such dependency graphs is a time consuming process while parameter's weighting could be misleading.

Sant' Anna et al. proposed in [Sant03] an extension of the well-known LCOM (Lack of Cohesion in Methods) metric developed by Chidamber and Kemerer [Chid94]. The proposed metric LCOO (Lack of Cohesion in Operations) measures the amount of method/advice pairs that do not access to the same instance variables. This metric measures the lack of cohesion of a component. According to their approach, if a component C_i has n operations (methods and advice) O_i,…, O_n then $\{I_j\}$ is defined as the set of instance variables used by operation O_j. Let |P| be the number of null intersections between instance variables sets. Let |Q| be the number of non-empty intersections between instance variables sets. Then, the lack of cohesion is defined by:

LCOO = |P| – |Q|, if |P| > |Q|,
LCOO = 0 otherwise.

A high LCOO value, according to Sant' Anna et al., indicates disparateness in the functionality provided by the aspect. The LCOM metric, on which the LCOO metric is based, has been widely experimented and discussed in the literature. It suffers from several problems as stated, among others, by B. Henderson-Sellers in [Hend96]. The fact that this metric is not normalized leads to some difficulties in the interpretation of the results. We believe that the LCOO metric suffers from the same problems. This will be discussed in section 6.

3 ASPECT MEMBERS DEPENDENCIES

Our basic concepts will be illustrated using AspectJ [Aspe05]. The proposed approach is easy to implement using static analysis of the code. AspectJ introduces several new language constructs such as: aspect, join points, pointcuts, advice as well as inter-type declarations. These various elements (including attributes and methods), which we call *aspect members*, allow an aspect expressing a concern that crosscut several basic modules. A join point represents well-defined points in the program flow, such as method calls and field sets. Pointcuts describe join points and context to expose. Advice is a method-like abstraction that defines code to be executed when a join point is reached. Pointcuts are used in the definition of an advice. Inter-type declarations defines how an aspect modifies a program's static structure, namely, the members and the relationship between components. Pointcuts and advice dynamically affect program flow, and inter-type declarations statically affects a program's class hierarchy. For further information on AspectJ mechanisms, one can see [Aspe02].

Such as for class members, high relatedness must exist between aspect members. To achieve its role, various interactions (direct and indirect) between its members are necessary. So, the degree of relatedness between members of a cohesive aspect should be high. Pointcut usage is closely tied with the advice concept. Anonymous pointcuts definition seems to reinforce this idea. Pointcuts dependencies are indirectly taken into account by advice. We believe that dependencies based on context exposure (provided by pointcuts, for example) are rather related to coupling concept.

4 ASPECT COHESION CRITERIA

We believe that aspects should implement crosscutting concerns in a cohesive way. As for classes, high functional cohesion exists when the elements of an aspect all work together to provide some well-bounded behavior. Aspect cohesion is an internal software attribute that measures the degree to which its members are bounded together. Cohesion can be used to identify the poorly designed aspects. An aspect with low cohesion has probably been assigned unrelated concerns. We can expect that such an aspect will be difficult to understand, to test, to reuse and to maintain.

An aspect may be represented by its *data members* (attributes) and its *modules* (methods and advice). In this section, we define the various connections that may exist between aspect members. To this end, we present various dependencies between attributes and modules. The interactions between aspect's members are situated at two

distinct levels: *Modules-Data* level and *Modules-Modules* level. From these interactions, several relationships between aspect members will be defined.

Let's consider an aspect $Aspect_i$. Let $A=\{ A_1, A_2 \ldots A_a\}$ be the set of its attributes and $M=\{ M_1, M_2 \ldots M_m\}$ the set of its modules. The modules-data connection criterion, defined in what follows, is based on attributes usage. It allows capturing modules sharing common attributes. The modules-modules connection criterion is based on methods invocation. It allows capturing interactions between modules, which do not share any attribute in common.

Modules-Data Connection Criterion

Definition 1: Let UA_{Mi} be the set of attributes used directly or indirectly by the module M_i. An attribute is used directly by a module M_i if the attribute appears in its body. An attribute is indirectly used by a module M_i if it is used directly by another module invoked directly or indirectly by M_i. There are m sets UA_{M1}, UA_{M2}, … UA_{Mm}. Two modules M_i and M_j of an aspect are related by the *UA_M relationship* if $UA_{Mi} \cap UA_{Mj} \neq \varnothing$. It means that there is at least one attribute shared (directly or indirectly) by the two modules.

In the example illustrated by figure 1, we can notice for example that modules M_1 and M_2 are related by the *UA_M relationship*. They share the same attribute A_1. In the other hand, modules M_1, M_3 and M_4 are also related by the *UA_M relationship*. Module M_1 access directly to attribute A_2 as it is the case of module M_3, while the module M_4 uses the module M_3 which access to the attribute A_2. In fact, according to figure 1, we have: $UA_{M1} = \{A_1, A_2\}$, $UA_{M2} = \{A_1\}$, $UA_{M3} = \{A_2\}$ and $UA_{M4} = \{A_2\}$. We have then, $UA_{M1} \cap UA_{M2} = \{A_1\}$, $UA_{M1} \cap UA_{M3} = \{A_2\}$ and $UA_{M1} \cap UA_{M4} = \{A_2\}$.

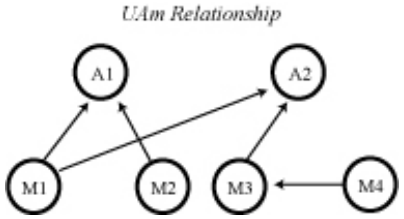

Figure 1. Modules sharing the same attributes.

Modules-Modules Connection Criterion

Definition 2: Let UM_{Mi} be the set of modules used directly or indirectly by the module M_i. A module M_j is used directly by a module M_i if M_j appears in the body of M_i. A module M_j is indirectly used by a module M_i if it is used directly by another module used directly or indirectly by M_i. There are m sets UM_{M1}, UM_{M2}, … UM_{Mm}. Two modules M_i and M_j of an aspect are related by the *UM_M relation* if $UM_{Mi} \cap UM_{Mj} \neq \varnothing$. It means that there is at least one module jointly used (directly or indirectly) by the two modules. We also consider that M_i and M_j are directly related if $M_j \in UM_{Mi}$ or $M_i \in UM_{Mj}$.

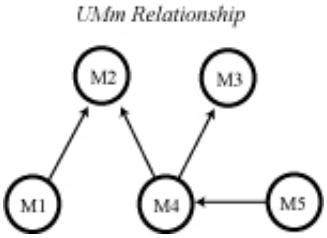

Figure 2. Modules calling the same module.

In the example illustrated by figure 2, we can notice for example that modules M_1 and M_4 are related by the *UM_M relationship*. They use the same module M_2. In the other hand, modules M_1 and M_5 are also related by the *UM_M relationship*. Module M_1 uses directly the module M_2 as it is the case of module M_4, which is used by the module M_5. In fact, according to figure 2, we have: $UM_{M1} = \{M_2\}$, $UM_{M4} = \{M_2, M_3\}$, $UM_{M5} = \{M_2, M_3, M_4\}$. We have then, $UM_{M1} \cap UM_{M4} = \{M_2\}$, $UM_{M1} \cap UM_{M5} = \{M_2\}$ and $UM_{M4} \cap UM_{M5} = \{ M_2, M_3\}$.

5 ASPECT COHESION MEASUREMENT

Based on the introduced connection criteria, we define the cohesion of an aspect by the degree of relatedness of its modules. The other members of the aspect are, in fact, indirectly considered. We are inspired by some approaches proposed for class cohesion measurement [Badr95, Badr04, Biem95]. We define, in what follows, our measurement of aspect cohesion. Let us define NM(Aspecti) as the total number of modules pairs in an aspect. NM is the maximum number of connections between aspect modules. Thus, in an aspect having N modules, $NM(Aspect_i) = N * (N -1) / 2$, $N > 1$.

Definition 3: Two modules M_i and M_j can be connected in many ways: by sharing attributes (definition 1) or by sharing modules (definition 2) or both.

Consider an undirected graph G_D where the vertices represent the modules of the aspect. There's an edge between two modules M_i and M_j if they are related. Let $NC(Aspect_i)$ be the number of connections between modules. $NC(Aspect_i)$ is given by the number of edges in the graph G_D. We define a new metric for aspect cohesion measurement based on relationships between its modules. The proposed aspect cohesion metric ACoh represents the relative number of connected modules: $ACoh(Aspect_i) = NC(Aspect_i) / NM(Aspect_i) \in [0,1]$.

Moreover, the considered graph may be used to generate the number of different groups of connected members (NCM) in the aspect. The number of groups of connected members (NCM) in an aspect may also be considered as an indicator of the lack of cohesion in the aspect. It is intended that an NCM value equal to 1 indicates that there is a single group of connected members, whereas an other value (> 1) of NCM may be used to indicate that an aspect may be more successful if split into two or more aspects. This may reveals the disparateness in the functionality provided by the aspect. The NCM value, in the case of an aspect where all its members work all together to achieve some bounded behavior, will be equal to 1.

The ACoh metric gives the degree of relatedness between aspect' members. A low value of ACoh indicates that the aspect' members are poorly related, in spite of the fact that they may constitute a single group of related members working together to implement a bounded behavior. However, it may also indicate (in an implicit way) the existence of several (two or more) groups of connected members. In fact, these different groups may reflect, in some cases, the disparateness of the roles assigned to an aspect. In this case, we will be able to determine it only by reviewing the code. A low value of ACoh may be interpreted in different ways and reveals in fact various situations: (1) the members of the aspect constitute a single group of connected members but are however poorly related, (2) the roles assigned to the aspect are in fact disparate (unrelated), and (3) possibly both. In practice, we may have two aspects with comparable values of cohesion (let us assume 0.50): In the case of the first aspect, the members are poorly related but constitute a single group of connected members, and in the case of the second aspect the roles assigned to the aspect are disparate (unrelated) which will be reflected in its implementation. Without the NCM metric, only by reviewing the code we will be able to determine it. The metric NCM reveals in an explicit way this problem. Together, they reflect in several situations some design problems. The case studies (section 6) illustrate this dimension. The ACoh metric indicates the cohesion degree of the aspect. The NCM (taken with ACoh) helps in interpreting the results.

Moreover, Briand et al. suggest in [Bria98] that cohesion measurement must have the following properties: nonnegative and standardized, has a minimum and a maximum, monotonous and that cohesion should not increase when two components are combined. ACoh metric provides a nonnegative and continuous value that range between a minimum of 0 and a maximum of 1. The last property will be illustrated in the next section.

6 CASE STUDIES

In this section, we present three case studies. Each one will illustrate our approach on a concrete example. Examples were selected in order to illustrate various situations while comparing our metric to the others proposals.

Case Study 1

Our first case study will be based on the example used by Zhao and Xu [Zhao04] to illustrate their approach. The code is presented in figure 3. This will allow us giving a first illustration of our proposal on a concrete example and positioning it in comparison with Zhao and Xu and Sant' Anna et al. approaches.

```
1  public class Point {
2   protected int x, y;
3   public Point(int _x, int _y) {
4    x = _x; y = _y;
5  }
6   public int getX() {return x;}
7   public int getY() {return y;}
8   public void setX(int _x) {x = _x;}
9   public void setY(int _y) {y = _y;}
10  public void printPosition() {
11   System.out.println("Point at (" + x + ", " +   y + ")");
12  }
13  public static void main(String[] args) {
14   Point p = new Point(1, 1);
15   p.setX(2);
16   p.setY(3);
17  }
18 }
19 public class Shadow {
20  public static final int offset = 10;
21  public int x, y;
22  Shadow(int _x, int _y) {
23   x = _x;
24   y = _y;
25  }
26  public void printPosition() {
27   System.out.println("Shadow at (" + x + ", " + y +   ")")
28  }
29 }
```

```
30 public aspect PointShadowProtocol {
31  private int shadowCount = 0;
32  public static int getCount() {
33   return   PointShadowProtocol.aspectOf().shadowCount;
34  }
35  public static void associate(Point p, Shadow s) {
36   p.shadow = s;
37  }
38  public static Shadow getShadow(Point p) {
39   return p.shadow;
40  }
41  private Shadow Point.shadow;
42  pointcut setting(Point p, int x, int y): args(x,  y)&&
43  target(p)&& initialization(Point.new(int, int));
44  pointcut settingX(Point p) : target(p) && call(void setX(int));
45  pointcut settingY(Point p) : target(p) && call(void setY(int));
46  after(Point p, int x, int y) returning() :     setting(p, x, y) {
47   Shadow s = new Shadow(x, y);
48   associate(p, s);
49   shadowCount++;
50  }
51  after(Point p) : settingX(p) {
52   Shadow s = getShadow(p);
53   s.x = p.getX() + Shadow.offset;
54   p.printPosition();
55   s.printPosition();
56 }
57  after(Point p) : settingY(p) {
58   Shadow s = getShadow(p);
59   s.y = p.getY() + Shadow.offset;
60   p.printPosition();
61   s.printPosition();
62  }
63 }
```

Figure 3: Point and Shadow example.

While examining the code of this aspect, several dependencies exist between aspects members. The association between a *Point* and a *Shadow* is adequately captured. Figure 4 illustrates the various members (data and modules) in the considered example with their respective interactions. The graph given by figure 5 represents pairs of connected modules captured by our approach. Modules are represented by vertices in the graph. Connections between modules are represented by edges.

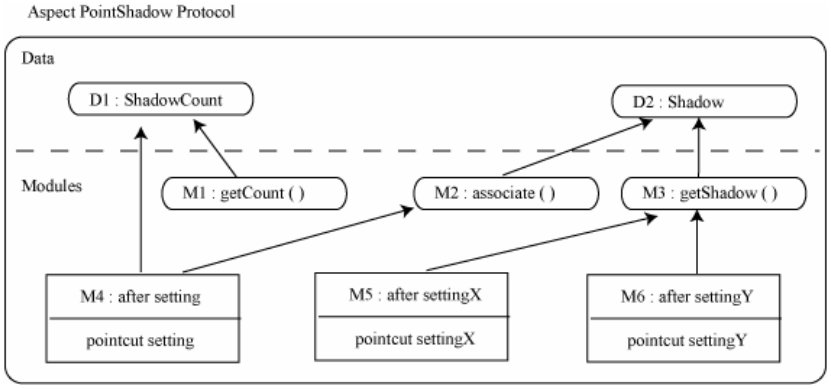

Figure 4: Interactions between aspect members.

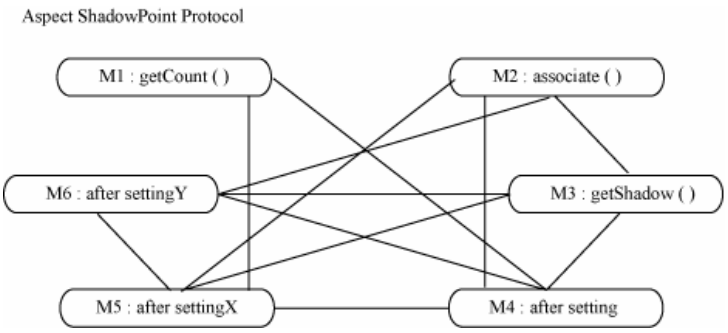

Figure 5: Connections between aspect modules.

For example, using definition 1, we get an edge between M2: associate() and M3: getShadow() (figure 5). Indeed, both M2: associate() and M3: getShadow() modules access the data member D2: shadow (figure 4). Using definition 1 again, one can directly connect Module M4: after setting with Module M6:after settingY (figure 5). Indeed, the attribute D2: shadow is indirectly used by both advices (M4 and M6) since it is respectively used directly by Module M2: associate () (which is used by M4) and M3: getShadow() (which is used by M6). Modules M5: after settingX and M6: afterSettingY can be related using definition 1 or definition 2. In order to simplify the notation, we'll use *PSP* to represent *PointShadowProtocol*. We give, in what follows, the obtained cohesion values for the considered example according to the three approaches:

ACoh (PSP) = NC (PSP) / NM(PSP) = 11 / 15 = 0.73
LCOO = |P| - |Q| = 13 – 2 = 11
Γ(A) (Zhao) = 1/18 = 0.056

As mentioned above, several dependencies exist between the aspect members. The obtained value (0.73), according to our approach, reflects these dependencies. The NCM number is equal to 1. In this example, the cohesion value suggested by Zhao and Xu' metric is equal to 1/18 (0.056) as stated in their paper [Zhao04] (based on a scale from 0 to 1). This measure indicates a very low cohesion, not to say a non-existent cohesion. However, this is not true. The obtained value does not reflect the structure of this aspect. With the present case study, LCOO gives 11. It is difficult to interpret since there is no guideline on the interpretation of any particular value.

Case Study 2

Gregor Kiczales stated in [Kicz04] that a *concern* is any element of a program (or design, requirements, test …) that you'd like to consider as a single unit. Concerns can be large or small. For example, the error-handling strategy of a system may itself be composed of many smaller aspects. We believe that aspect responsibility assignment should follow well-tested software engineering principles. An aspect must express a concern in a cohesive manner. Like classes that have been assigned disparate responsibilities, an aspect implementing several and disparate concerns will present a low cohesion. Such aspect will (intuitively) be difficult to understand, to test, to re-use and to maintain. In order to reflect this important design problem, let us take a concrete example.

Let us assign to aspect *PointShadowProtocol* (from previous case study 1) a new concern, the validation of pre-conditions assignment for attributes x and y in *Point*. A simple message will be displayed if the pre-conditions are not respected. Therefore, the aspect is expressing two concerns: association between a *Point* and its *Shadow* (concern A) as well as the validation of the pre-conditions (concern B). These two roles are, in fact, disparate. No direct or indirect connection exists between them.

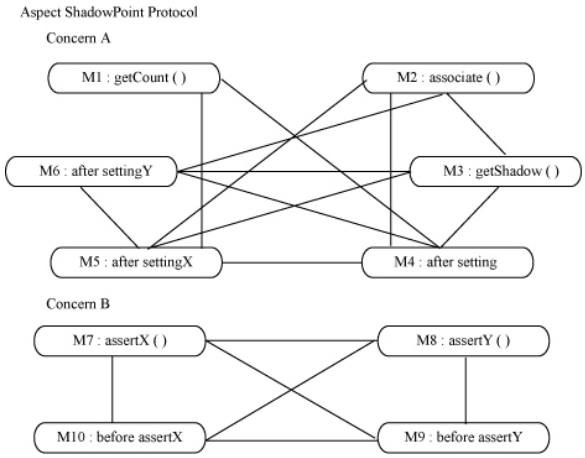

Figure 6: Connections between aspect members.

Our objective, through this case study, is to show that using concrete example our approach reflects (through the obtained measurement) the disparateness that may exist in the functionality provided by an aspect (several roles assigned to the aspect). This should result, inevitably, in a significant drop of cohesion within the aspect. Graph presented by figure 6 illustrates the pairs of connected modules in the aspect. Aspect modules are represented by vertices while edges represent relationships between them. Further examination of figure 6 content shows that the two roles do not share anything. We give, in what follows, the obtained cohesion values for the considered example according to the three approaches:

ACoh (PSP) = NC (PSP) / NM (PSP) = 17 / 45 = 0.38
LCOO = |P| - |Q| = 42 – 3 = 39
Γ(A) (Zhao) = 7 / 270 = 0.026

We have, in fact, two unrelated concerns that have been assigned to the same aspect. A significant drop of cohesion is reflected by the new value of ACoh. This drop is also reflected by LCOO metric since we get a value equals to 39. As stated, for example, by Henderson-Sellers in [Hend96] (for the LCOM metric), this is difficult to explain since there is no guideline on the interpretation of any particular value. When there is no cohesion, we expect the cardinality of the set P (the number of pairs that have no similarity) to be high and of the set Q (the number of pairs which have some similarity) to be low, and thus LCOO has a large value in this example. Another concern is that LCOM (and LCOO indirectly) attempts to measure only structural cohesion, whereas a major focus of OO is its ability to create logically (i.e., semantically) cohesive modules (classes) [Hend96]. LCOO do not capture this characteristic, which is important in an aspect-oriented system. Henderson-Sellers also suggest in [Hend95] that a better LCOM measure should have values on a percentage range (a fraction/percentage of a perfect value). The ACoh metric correctly reflects those properties. Finally, the measurement obtained with Zhao and Xu's metric in this case also does not reflect the structure (in terms of dependencies between members) of the considered aspect.

Moreover, and knowing that the responsibilities assigned to the aspect are (intuitively) non-related, we should have more than one group of connected members. In fact, this is also reflected by the NCM number (equal to 2 in this case) of the

considered example. By examining the G_D graph (figure 6) and the non represented details regarding the attributes (interactions between modules and attributes), we can observe that the aspect is composed of two unrelated groups of connected members. The two groups of connected members are non-related. This explains also the lack of cohesion in the considered example that is adequately reflected by the ACoh value. This aspect may be considered for restructuring (splitting it into two separate aspects).

Case Study 3

The following case studies will be based on the AspectJ implementation of some GOF Design Patterns [Gamm95] proposed by Hanneman and Kiczales [Hann02]. Hanneman and Kiczales stated in [Hann02] that improvements of using aspect paradigm come primarily from modularizing the implementation of the pattern. Using the aspect paradigm, 74% of GoF patterns are implemented in a more modular way and 52% are reusable. An integral part of achieving this consists in removing code-level dependencies from the participant classes to the implementation of the pattern. Using aspect programming, 12 of 23 design patterns are represented as abstract aspect into reusable library. Cohesion amongst those aspects should be high. This is our first investigation of an interesting dimension introduced by aspect programming. Reusable aspects could be seen as design pattern templates. For this first review, we choose to exclude abstract methods, interface declarations as well as inheritance considerations from our measurement. We present, in what follows, three of the GOF design patterns. We give, for each pattern, the obtained cohesion measures using our metric (ACoh), the LCOO metric [Sant03], and the Γ(A) metric [Zhao04].

Discussion

Table 1 gives a summary on the obtained cohesion values for the design patterns. For this first review, we present the Observer, the Mediator and the Command patterns. We obtained strong cohesion measurement using our metric (ACoh = 1) for the Observer and Mediator patterns. By examining the different details related to the dependencies that exist between the pattern members, we can note that their members constitute a single group of connected members. This indicates that all the members of the Observer and the Mediator pattern work together to provide some well-bounded behavior. The NCM number is also equal to 1. High cohesion for an aspect indicates that its various members express in a cohesive way a singular role (concern) that will be difficult to split. However, we obtained with the command pattern a significant drop of cohesion (ACoh = 0,5 and LCOO = 32). These results reflect the structure of the Command pattern. The members of the Command pattern are not as strongly related as the members of the Observer and Mediator patterns. Moreover, the obtained value in this case for the number of groups of connected members (NCM) is 2. This indicates that the members of the *Command pattern* constitute two groups of connected members in opposite to the two other patterns. This dimension is reflected by NCM value. We see, according to this case, that the ACoh metric discriminates the patterns according to their structures.

On the other hand, we can notice that the obtained values according to LCOO metric are difficult to interpret since there is no guideline on the interpretation of any particular value. If we consider the Observer pattern and the Mediator pattern, which

are quite similar in terms of members' relatedness, the corresponding LCOO values are respectively 6 and 1. It is difficult to interpret the difference between the two values since there is no range between a minimum and a maximum value.

Table 1: Case studies cohesion values.

Pattern	ACoh	LCOO	Γ(A)
Observer	1	6	0.41
Mediator	1	1	0.66
Command	0.50	32	0.032

Finally, plotting the results of Zhao and ACoh metrics for the patterns will show that, in relative terms, the two behave identically. It would thus seem that the essence of cohesiveness is captured by both metrics. However, upward shifts of Zhao's metric seem problematic. The reason why we get such a high value comparatively to the previous case studies could be partially explained by the fact that the Observer and Mediator patterns only uses a single attribute. Consequently, the measure of the inter-attribute cohesion (γ_a) suggested by their approach is 1. As a reminder, Γ(A) is calculated using $\beta_1 * \gamma_a + \beta_2 * \gamma_m + \beta_3 * \gamma_{ma}$; where every γ is a cohesion facet. Since we are using, in our opinion, an arbitrary value of 1/3 for our β parameters, the contribution of $\beta_1 * \gamma_a$ is important. While examining the values obtained using the Zhao and Xu' metric, the difference between the cohesion values of Observer and Mediator patterns, in the one hand, and the cohesion value of Command pattern, in the other hand, is important. ACoh suggests that Observer and Mediator patterns are twice cohesive than the Command pattern where Zhao and Xu's metric suggests that those patterns are, respectively, 12 and 20 times more cohesive. This behavior seems revealing instability of this metric. It is, however, necessary to make more investigations before drawing any final conclusion on this particular subject.

7 EXPERIMENTAL STUDY

In practice, the true value of cohesion metrics lies in their potential to help assess large pieces of code that cannot easily be characterized by developers. So far, the aspects used to contrast ACoh with the other two metrics are all small and uncomplicated. So far we only presented simple examples with limited interactions. However, given the situation of AOSD, it is quite difficult to find some real scalable aspect-oriented applications.

For this first experimental investigation, we developed a cohesion measurement tool (in Java) for AspectJ programs to automate (in the present version) the computation of the LCOO and ACoh metrics. We also choose to exclude abstract methods and interface declarations for this first review. While we believe that the proposed approach adequately reflects aspect cohesion compared to the others proposals, an interesting dimension is brought by inheritance. Concrete aspect will give a concrete implementation for abstract methods or will specify concrete participants with corresponding interface implementations. We plan to include this dimension in a future work and extend our approach with regards to inheritance mechanisms and relationships.

Selected system

Even if it is still in a development process, we choose to analyze the Atrack project, an open source bug tracking application developed by Bodkin and Mulez [Atra05] that demonstrates use of Aspect-Oriented Programming (AOP) with AspectJ. It uses AOP pragmatically to provide systematic support for technical, middleware, and business concerns. The project also provides and demonstrates effective use of a proposed common support library for AspectJ, which provides flexible support for exception handling, security, logging, tracing, transaction and persistent session management, and virtual mock objects for testing.

Results

We measured aspect cohesion values according to ACoh and LCOO metrics for the selected test system. Table 2 provides the obtained results. LCOO measures the amount of method/advice pairs that do not access the same instance variables. A high LCOO value indicates disparateness in the functionality provided by the aspect. ACoh is based on modules-data and modules-modules connection criteria and count the degree of relatedness of the modules.

List of aspects	#	LCOO	ACoh
AjeeExceptionHandling	1	0	1
AjeeLogManager	0	0	0
AtrackActionDefinition	0	0	0
AtrackLogManager	0	0	0
ATrackTransactionControl	1	0	1
Authentication	2	1	1
ControllerExceptionHandling	2	1	1
ExecutionMonitor	12	66	0.32
ExecutionTracer	20	170	0.47
HttpSessionTracer	3	3	1
Invariants.java	1	0	1
LogManager	15	105	0.74
ModelExceptionHandling	1	0	1
Observing	0	0	0
Standards	0	0	0
VirtualMocks	15	101	0.08

Table 2: Atrack project – Cohesion values.

It is intended that a small value of LCOO and a high value of ACoh imply high relatedness between aspect members. As stated by Henderson-Seller in [Hend95], a measure must give values that can be uniquely interpreted in terms of cohesion. LCOO does not have this ability. Since there's no guideline on the interpretation of any LCOO values, the results in table 2 are difficult to compare. This is partially due to the fact that LCOO measure does not represent a number of potential connections

ratio. Moreover, LCOO metric does not include the ability to give values across a full range and nor for any specific value to have a higher probability of attainment that any others, all other things being equals as stated in [Hend95]. This remark holds for LCOO metric since it is based on the LCOM metric. For example, a LCOO value of -12 or -3 will be treated as a 0. The probability of observing the 0 value is higher than any other value. This seems problematic and will be further discussed with an example.

Some results could be extracted from table 2. Let us consider the LCOO values for the aspects LogManager (LCOO = 105) and VirtualMocks (LCOO= 101). With these results, one can conclude that these aspects are quite disparate. The value of ACoh for VitualMocks (ACoh = 0.08) seems to support this assumption while the ACoh value of LogManager is fairly high (ACoh =0.74). Indeed, VirtualMocks aspect defines 15 modules that are quite disparate. The maximum number of connections between the aspect members is N * (N - 1) / 2 = 15 (15 -1) / 2 = 105. For this particular aspect, only 8 modules are directly related. We obtained a low cohesion measure (ACoh = 0.08). For the considered aspect VirtualMocks, both metrics LCOO and ACoh suggest a low cohesion value (LCOO = 101 et ACoh = 0.08). However, this is not the case with the LogManager aspect (LCOO = 105 et ACoh = 0.74).

The code presented in figure 7 is extracted from the LogManager aspect. An attribute, logManagerLogger, is defined and accessed by the getLogger() method. No other method, besides getLogger(), uses the logManagerLogger attribute directly. This partially explains the low value of LCOO since |P| is defined as the number of null intersections between instance variables sets. For this example, |P| value will be fairly high. With the LCOO metric, interactions between modules are not captured if the modules don't access directly to an instance variable. A fairly high number of related modules are not considered by the LCOO approach. On the other hand, the ACoh value = 0.74, can be explained by the fact that our approach takes these relationships into account. LogTrace, logWarn, logInfo methods access indirectly to logManagerLogger attribute with a direct call to getLogger() method. Indirect dependencies will be captured by the ACoh approach. This is an important dimension that is not reflected by LCOO metric.

```
1  public aspect LogManager {
2      private Log Loggable.logManagerLogger =
3              LogFactory.getLog(getClass());
4
5      // this should be protected
6      public Log Loggable.getLogger() {
7          if (logManagerLogger==null) {
8              logManagerLogger = LogFactory.getLog(getClass());
9          }
10         return logManagerLogger;
11     }
12     public void Loggable.logTrace(Object message) {
13         getLogger().trace(message);
14     }
15
16     public void Loggable.logInfo(Object message) {
17         getLogger().info(message);
18     }
19
20     public void Loggable.logWarn(Object message) {
21         getLogger().warn(message);
22     }
23 // …
24
```

Figure 7: LogManager aspect (partial).

Another example concerns the Authentication aspect. This aspect is fairly simple ; it only defines an attribute SUBJECT_ID, a method forceRedirect() and one advice

(around type). Using Sant' Anna et al approach, we obtain a cohesion measure: LCOO = |P| – |Q| = 1 – 0 = 1. Modules forceRedirect( ) and around advice do not access SUBJECT_ID attribute. However, multiple calls to forceRedirect() method are made within the aspect body. Again, this dimension is captured by ACoh. A cohesion value of 1 is obtained with ACoh since all the modules are clearly related. The Modules-Modules interactions definitions allow us to adequately capture these relations that are not reflected by the LCOO approach.

This first review allowed us to support some of our intuitions. Indirect dependencies will be captured by the ACoh approach. Moreover, some aspects (Observing, Standards, etc) presented in table 2 do not defined any modules. For these aspects, cohesion values of LCOO and ACoh metrics are equal to 0. LCOO suggests a perfect cohesion value for this particular aspect: one can obtain LCOO = 0. LCOO approach will suggest a perfect cohesion value whether the aspect is empty or implementing a well-bounded behavior. Let us consider an example where an aspect defines 3 perfectly related modules; the cohesion value LCOO = |P| – |Q| = 0 – 3 = 0. To interpret these 2 examples, we have to refer to the source code. Some values of LCOO have a higher probability of attainment that any others, all other things being equals. The null value is one example; -3 is being treated as 0. This is a major problem. With the ACoh approach, an empty aspect will lead to cohesion measure equals to 0. We believe that using empty aspects is questionable.

8 CONCLUSIONS AND FUTURE WORK

We presented, in this paper, a new approach for aspect cohesion measurement. It is based on dependencies between aspect members. The proposed metric measures the degree of relatedness of its modules. We believe that this work represents a first realistic proposal taking into account aspect's characteristics. ACoh metric is well adapted to reflect design problems such as assigning disparate roles to an aspect.

Also, an interesting dimension brought by inheritance was not addressed in the present work. In fact, concrete aspects will give a concrete implementation for abstract methods or will specify concrete participants with corresponding interface implementations. Moreover, this work was limited to direct relationships between aspect's members. It would be interesting to study aspects cohesion with regards to their indirect relationships. By indirect relationship we mean, for example, taking three modules M_1, M_2 and M_3, that module M_1 and module M_2 could share directly/indirectly a data D_1, and module M_2 and module M_3 could share directly/indirectly a data D_2. Transitive closure of this relationship suggests that M_1 and M_3 are indirectly connected. The concept of transitive closure has been used by Bieman et al. [Biem95] in object-oriented systems for measuring the Loose Class Cohesion. We feel while examining concrete examples that indirect relationships between aspect members also contribute to aspect cohesion.

As future work, we plan to extend our approach: (1) to take into account the indirect relationships between aspect members, (2) to include the inheritance dimension (3) to do more investigations on the impact of aspect paradigm on software quality attributes (classes' cohesion for example) (4) to conduct static performance analysis on large projects (to see how well our metric scale) and finally (5) to conduct

an evaluation of cohesion metrics to predict maintenability (using iterated versions of a system).

9 ACKNOWLEDGMENTS

This research was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) and NATEC (Nature et Technologies) grants.

REFERENCES

[Atra05] Atrack Project Website: https://atrack.dev.java.net/

[Aspe02] The AspectJ Team. The AspectJ Programming Guide. 2002

[Aspe05] The AspectJ Website: http://eclipse.org/aspectj/

[Badr95] L. Badri, M. Badri and S. Ferdenache, Towards Quality Control Metrics for Object-Oriented Systems Analysis, *Proceedings of TOOLS (Technology of Object-Oriented Languages and Systems) Europe'95*, Versailles, France, Prentice-Hall, March 1995.

[Badr04] L. Badri and M. Badri, A Proposal of a New Class Cohesion Criterion: An Empirical Study, *in Journal of Object Technology*, vol. 3, no. 4, April 2004. http://www.jot.fm/jot/issues/issue_2004_04/article8/index_html

[Biem95] J.M. Bieman and B.K. Kang, Cohesion and reuse in an object-oriented system, *Proceedings of the Symposium on Software Reusability (SSR'95)*, Seattle, WA, pp. 259-262, April 1995.

[Booc94] G. Booch, *Object-Oriented Analysis and Design With Applications*, Second edition, Benjamin / Cumming, 1994.

[Bria98] L.C. Briand, J. Daly and J. Wusr, A unified framework for cohesion measurement in object-oriented systems, *Empirical Software Engineering*, Vol.3, No.1, pp. 67-117, 1998.

[Cecc04] M. Ceccato and P. Tonella, *Measuring the Effects of Software Aspectization*, In Proceedings of the First Workshop on Aspect Reverse Engineering, November, 2004.

[Chae00] H.S. Chae, Y. R. Kwon and D H. Bae, *A cohesion measure for object-oriented classes*, Software Practice and Experience, No. 30, pp. 1405-1431, 2000.

[Chid94] S.R. Chidamber and C.F. Kemerer, A Metrics suite for object Oriented Design, *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-493, June 1994.

[Gamm95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[Hann02] J. Hannemann and G. Kiczales, Design pattern implementation in Java and AspectJ, *In Proceedings of the 17th Annual Conference on Object-oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 161–173, 2002.

[Hend95] B. Henderson-Sellers, A book of Object-Oriented Knowledge, 2nd ed., Prentice Hall, Sydney, Australia, 1995.

[Hend96] B. Henderson-Sellers, Object-Oriented Metrics Measures of Complexity, Prentice-Hall, 1996.

[Hitz95] M. Hitz and B. Montazeri, Measuring coupling and cohesion in object oriented systems, *Proceedings of the Int. Symposium on Applied Corporate Computing*, pp. 25-27, October 1995.

[Kicz97] G. Kiczales, J. Lamping, A. Menhdekar, C. Maeda, C. Lopes, J.M. Loingties, and J. Irwin, Aspect-oriented programming, In M. Aksit and S. Matsuoka, editors, *European Conference on Object-oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer, 1997.

[Kicz01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, An Overview of AspectJ, In J. Lindskov Knudsen, editor, *European Conference on Object-oriented Programming*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[Kicz04] G. Kiczales, It's the crosscutting, *Software Development Magazine*, February 2004.

[Larm04] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd edition, Prentice-Hall, 2004.

[Li93] W. Li and S. Henry, Object oriented metrics that predict maintainability, *Journal of Systems and Software*, Vol. 23, pp. 111-122, February 1993.

[Pres01] R. S. Pressman, Software Engineering, A practitioner's approach, Fifth edition, McGraw Hill, 2001.

[Sabb04] D. Sabbah, From Promise to Reality, *Proceedings of the 3rd international conference on Aspect-oriented software development (AOSD'04),* pages1-2, 2004.

[Sant03] C. Sant'Anna, Alessandro Garcia, Christina Chavez, Carlos Lucena & Arndt von Staa, On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. *XXIII Brazilian Symposium on Software Engineering*, Manaus, Brazil, October 2003.

[Zaca03] A. Zacaria, H. Hosny, Metrics for Aspect-Oriented Software Design, *In Proceedings of Third International Workshop on Aspect-Oriented Modeling, AOSD'03*, 2003.

[Zhao03] J. Zhao, Coupling Measurement in Aspect-Oriented Systems, Technical-Report SE-142-6, *Information Processing Society of Japan (IPSJ)*, July 2003.

[Zhao04] J. Zhao and B. Xu, Measuring Aspect Cohesion, *Proceeding of International Conference on Fundamental Approaches to Software Engineering (FASE'2004)*, LNCS 2984, pp.54-68, Springer-Verlag, Barcelona, Spain, March 29-31, 2004.

About the authors

Jean-François Gélinas (Gelinaje@uqtr.ca) is a student of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. He recently received his master in computer science from the University of Quebec at Trois-Rivières. His main areas of interest include aspect-oriented programming as well as various topic of software engineering.

Mourad Badri (Mourad.Badri@uqtr.ca) is professor of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. He holds a PhD in computer science (software engineering) from the National Institute of Applied Sciences in Lyon, France. His main areas of interest include object and aspect-oriented software engineering, software quality attributes, and formal methods.

Linda Badri (Linda.Badri@uqtr.ca) is professor of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. She holds a PhD in computer science (software engineering) from the National Institute of Applied Sciences in Lyon, France. Her main areas of interest include object and aspect-oriented software engineering, software quality attributes, maintenance, and web engineering.