5 Concern-Driven Design Heuristic Rules

The main issue in working with metrics is how we should deal with measurement results (Lanza & Marinescu, 2006). How can all those numbers help us improve the quality of our software? Usually a metric alone cannot help very much in answering this question and therefore metrics should be used in combination to provide relevant information. The question is: how should we combine metrics in order to make them serve our purposes?

In Section 2.5, we introduced the concept of design heuristic rules as a means to support the interpretation of design metrics and, as a consequence, increase their usefulness. Heuristic rules provide the engineer with information that is useful in the context of an investigation goal, in particular, an investigated design flaw. The main goal of design heuristic rules is to provide the engineers with a mechanism that allows them to work with metrics in a way conceptually much closer to the real intentions in using metrics.

A design heuristic rule is a logical expression based on metrics by means of which design fragments presenting specific problems can be detected. One of the main goals of this work is to propose design heuristic rules to support the detection of modularity-related problems in object-oriented or aspect-oriented detailed design. These rules will help software engineers to find the design fragments that are negatively affected by the poor modularization of concerns.

Currently, conventional object-oriented metrics (Section 2.4) are the basis upon which existing design heuristics rules are defined (Section 2.5). However, these metrics are limited by the fact that they are not driven by the system concerns. As a consequence, modularity assessment based on existing rules is impaired by limitations similar to the ones presented by conventional metrics (Section 1.2).

In this context, we propose a suite of concern-sensitive design heuristic rules that aims at supporting the modularity assessment of both object and aspect-oriented detailed design. In addition to concern-driven metrics, the detailed design assessment process was enhanced with more expressive heuristic rules. The main reason for that is because our previous experience in design measurement (Sant'Anna et al., 2003, 2004; Kulesza et al., 2006; Garcia et al., 2004a, 2005, 2006; Figueiredo et al., 2008b; Greenwood et al., 2007a; Cacho et al., 2006a) has shown that the data set obtained by applying concern-sensitive metrics to detailed design artifacts is usually large. This occurs due to the high number of detailed design elements (classes, aspect, methods, attributes, and so forth), when compared to the number of high-level elements defined in the architectural design. Therefore, mechanisms to support results interpretation, such as heuristic rules, are even more important in detailed design assessment.

This chapter defines our suite of concern-driven design heuristic rules. First, we point out the limitations of related work (Section 5.1). Section 5.2 depicts the model of concern representation, upon which the detailed design metrics are defined. Section 5.3 presents the suite of metrics used in the definition of the proposed rules. It includes concern-driven metrics and conventional metrics. Finally, Section 5.4 describes the proposed concern-driven design heuristic rules.

5.1.
Limitation of Conventional Heuristic Rules

To the best of our knowledge there is no suite of heuristic rules based on concern-driven metrics so far. Thus, within the area of software measurement, the most closely related work to ours is heuristic rules based on conventional object-oriented metrics, herein called as conventional heuristic rules, such as the ones proposed by Marinescu (2002, 2004).

In order to synthesize the main point of our criticism about conventional heuristic rules, we point out the limitations of one of Marinescu's rules (Marinescu, 2002, 2004). To this end, we analyze the effectiveness of the rule in the light of the detailed design showed in Figure 14. This figure presents a partial representation of the object-oriented design of an OpenOrb-compliant middleware system (Cacho et al, 2006a, 2006b, 2007), used in the evaluation of our rules (Section 8.1). Figure 14 also shows the design elements related to the Observer design pattern (Gamma, 1995). The Observer pattern is the concern analyzed in the example.

The analyzed rule aims at detecting a specific kind of modularity flaws, namely Shotgun Surgery bad smell (Fowler, 1999). *Bad Smells* are proposed by Kent Beck in Fowler's book (Fowler, 1999) to diagnose symptoms that may be indicative of something wrong in the design. According to Fowler (1999), the *Shotgun Surgery* bad smell is encountered when every time you make a kind of change, you also have to make a lot of little changes to a lot of different classes. When the changes are all over the place, they are hard to find, and it is easy to miss an important change. Thus, this design flaw strongly affects the design modularity.

The reason for choosing Shotgun Surgery as illustrative is because it is believed to be a symptom of design flaws caused by a poor modularization of concerns (Monteiro & Fernandes, 2005). Therefore, it *might be* avoided with the use of aspects. Monteiro & Fernandes (2005) claim that Shotgun Surgery is a symptom of crosscutting concern that can be solved by using the aspect-oriented refactoring *Extract Feature into Aspect* proposed in their work.

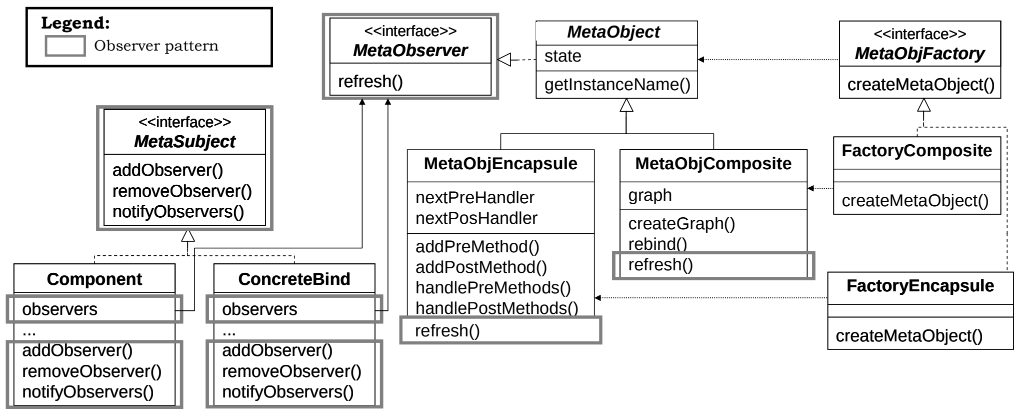

Figure 14: Design slice of an OpenOrb-compliant middleware system (Cacho et al, 2006a, 2006b, 2007).

In Section 2.5, we already presented the definition of Marinescu's heuristic rule for detecting Shotgun Surgery. We repeat it here in order to facilitate referring to it during the following discussion. This rule is based on two conventional coupling metrics. It is defined as follows (Marinescu, 2002):

Shotgun Surgery := ((CM, TopValues(20%)) and (CC, HigherThan(5)), where CM stands for the Changing Method metric (Marinescu, 2004), which counts the number of distinct methods that access an attribute or call a method of the given class. CC stands for the Changing Classes metric (Marinescu, 2002), which counts the number of classes that access an attribute or call a method of the given class. *TopValues* and *HigherThan* are filtering mechanisms parameterized with a value representing the threshold. Therefore, the Shotgun Surgery heuristic states that a class is suspect of having shotgun surgery whether it presents one of the 20% highest values for CM and has CC value higher than five. Note that this rule must be applied for each class in the design.

Applying CC and CM to the design in Figure 14, we obtain CC = 0 and CM = 15 for the MetaSubject interface (Figure 14). Based on these values and computing Marinescu's heuristic for Shotgun Surgery, this interface is not regarded as a suspect of a bad smell. This occurs because CC is 0, as no class in the system directly accesses MetaSubject. Nevertheless, this interface can be clearly considered as Shotgun Surgery because changes on its methods would trigger many other changes in every class implementing it and potentially in classes calling its overridden methods. For instance, a rename of the addObserver() method in the MetaSubject interface causes updates to the classes Component and ConcreteBind (Figure 14) and to several other classes which call addObserver().

This example aims at showing how conventional heuristic rules are limited to point out the overall influence of a concern – the Observer design pattern in this case – in other parts of the design. In particular, Marinescu's rule was not able to detect that a significant number of classes include design elements related to the Observer pattern and, as a consequence, that they could be affected due to a change in this concern. Hence, Marinescu's rule could not highlight the complete impact of the Observer pattern because it considers only measures based on class and method abstractions.

5.2.
Concern Representation at Detailed Design

The metrics presented here are also rooted at a concern-to-design mapping in the same way as the architectural concern-driven metrics (Chapter 4). However, instead of being mapped to architectural design elements, here the concerns are assigned to detailed design elements, such as classes, aspects, methods, attributes, and pieces of advice. It is important to bear in mind that the metrics can be applied to both object-oriented and aspect-oriented designs. Therefore, the concern representation is defined upon the set of aspect-oriented detailed design abstractions, as it also encompasses all the object-oriented abstractions.

5.2.1. Detailed Design

Before defining the metrics, we describe in this section the abstractions and composition mechanisms we consider as detailed design elements in this thesis. We will focus only on the elements which are essential to the definition of our metrics.

Components, Attributes and Operations

An aspect-oriented detailed design of a system consists of a set of classes and aspects. For generality purposes, the classes and aspects of a detailed design *S* are called as components and denoted by *C(S)*. Each component *c* consists of a set of attributes, denoted as *A(c)*, and a set of operations, represented as *O(c)*. In classes, operations are methods, and, in aspects, operations represent methods and pieces of advice. For notational convenience, we also define:

(i) the members of a component *c*, represented as $M(c) = A(c) \bigcup O(c)$,

(ii) the set of all attributes of a system, represented as $A(S) = \bigcup_{c \in C(S)} A(c)$,

(iii) the set of all operations of a system, represented as $O(S) = \bigcup_{c \in C(S)} O(c)$.

It is important to highlight that although the term component is used at both architectural and detailed design representations, it represents different abstractions in each context. In the detailed design context, a component is either a class or an aspect. In architectural design, a component is an abstraction of the component-and-connector viewtype, as described in Section 4.2.1. Similar reasoning applies to the term operation, which represents either a method or a piece of advice in the detailed design context.

Inter-component Connections

In Section 2.4, we have described a list of possible types of connections between classes summarized by Briand et al (1999). Our concern-sensitive coupling metrics (Section 5.3.4) take into account two of those connections: (i) a method of class *d* references an attribute of class *c*, and (ii) a method of class *d* invokes a method of class *c*. Actually, our metrics take into account versions of these type of connections tailored to aspect-oriented design: (i) an operation (method or advice) of a component (class or aspect) *d* references an attribute of component *c*, and (ii) an operation of a component *d* invokes a method of component *c*.

We define, therefore, the set of components which have an attribute referenced by a given operation *o* as *RC(o)*. Let *S* be the detailed design of a system, $c \in C(S)$ be a component of *S*, $o \in O(c)$ be an operation of *c*. Then $c' \in RC(o) \Leftrightarrow c' \in C(S) - \{c\} \wedge \exists a \in A(c')$ such as *o* references *a*.

We also define the set of components which have a method invoked by an operation *o* as *IC(o)*. Let *S* be the detailed design of a system, $c \in C(S)$ be a component of *S*, $o \in O(c)$ be an operation of *c*. Then $c' \in IC(o) \Leftrightarrow c' \in C(S) - \{c\} \wedge \exists o' \in O(c')$ such as *o* invokes *o’*. These two sets will be used in the definition of our concern-sensitive coupling metrics (Section 5.3.4).

Intra-component Connections

One of our concern-sensitive couplings is defined upon intra-component connections, which represent connections between internal members of a component. This metric takes into account two types of intra-component connections: (i) an operation *o* of a given component references an attribute of the same component, and (ii) an operation *o* of a given component invokes a method of the same component.

In this context, we define the set of attributes referenced by a given operation *o* as *RA(o)*. Let *S* be the detailed design of a system, $c \in C(S)$ be a component of *S*, $o \in O(c)$ be an operation of *c*. Then $a \in RA(o) \Leftrightarrow a \in A(c)$ and *o* references *a*.

We also define the set of operations invoked by a given operation *o* as *IO(o)*. Let *S* be the detailed design of a system, $c \in C(S)$ be a component of *S*, $o \in O(c)$ be an operation of *c*. Then $o' \in IO(o) \Leftrightarrow o' \in O(c) - \{o\}$ and *o* invokes *o'*.

Design Concern

With the definition of the detailed design elements that is considered in our approach, it is possible to define the notion of *concern representation in detailed design* (or simply, *design concern*). A design concern *con* consists of a list of detailed design elements assigned to it. These elements can be components (classes or aspects), operations (methods or pieces of advice), or attributes. A design element can be responsible for totally or partially realizing more than one concern. Therefore, a design element can be assigned to more than one concern.

Let *S* be the detailed design of a system, for each $c \in C(S)$, the set of concerns to which *c* is assigned is denoted as *Con(c)*. Let $o \in O(c)$ be an operation of *c*, the set of concerns to which *o* is assigned is denoted as *Con(o)*. Let $a \in A(c)$ be an attribute of *c*, the set of concerns to which *a* is assigned is denoted as *Con(a)*. *Con(S)* is the set of all concerns in the design and is represented as:

$$Con(S) = \bigcup_{c \in C(S)} Con(c) \cup \bigcup_{o \in O(S)} Con(o) \cup \bigcup_{a \in A(S)} Con(a)$$

Let S be the detailed design of a system, for each $con \in Con(S)$, the set of components assigned to *con* is denoted as:

$C(con) = \{c \mid c \in C(S) \wedge con \in Con(c)\}$.

Similarly, the set of operations assigned to *con* is denoted as:

$O(con) = \{o \mid o \in O(S) \wedge con \in Con(o)\}$.

The set of attributes assigned to *con* is denoted as:

$A(con) = \{a \mid a \in A(S) \wedge con \in Con(a)\}$.

Finally, the set of members assigned to *con* is denoted as:

$M(con) = A(con) \cup O(con)$.

5.3. Concern-Driven Metrics for Detailed Design

The concern-driven heuristics are based on the combination of concern-driven metrics and conventional metrics. This section presents the definition of the concern-driven metrics used in the definition of the proposed suite of heuristics (Section 5.4). Similarly to the architectural metrics, the used detailed design metrics quantify different facets of relations involving concerns and design elements which include:

- concern diffusion,
- interaction between concerns,
- concern-based cohesion,
- concern-sensitive coupling, and
- concern-sensitive size.

In the following sections, each metric is presented in terms of an informal definition, a formal definition, and an example. We do not discuss here the motivation behind the detailed design metrics, because the motivation behind detailed and architectural metrics in the same category is the same, and have already been discussed in Section 4.3. The examples are given in terms of a slice of an OpenOrb-compliant middleware system (Cacho et al., 2006a, 2006b, 2007) (Figure 15). Figure 15 also highlights the design elements related to two design patterns, namely Observer and Factory Method (Gamma, 1995), which are the assessed concerns in this example. Table 4 **Error! Reference source not found.**presents a summary of the detailed design metrics suite.

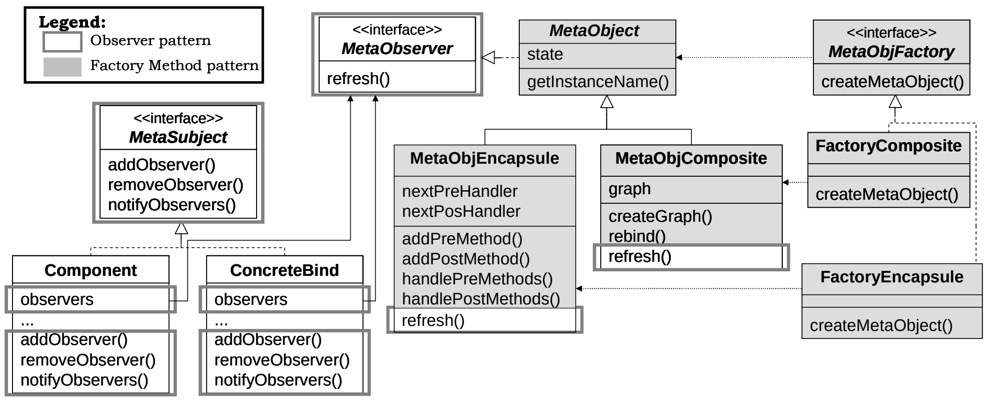

Figure 15: Observer and Factory Method patterns used in the design of an OpenOrb-compliant middleware system (Cacho et al., 2006a, 2006b, 2007).

Attribute	Metric	Definition
Concern Diffusion	Concern Diffusion over Components (CDC)	It counts the number of components (classes and aspects) which contributes to the realization of a given concern.
Interaction Between Concerns	Component-level Interlacing Between Concerns (CIBC)	It counts the number of other concerns with which a given concern shares at least a component.
Concern-based Cohesion	Lack of Concern-based Cohesion (LCC)	It counts the number of concerns addressed by a given component.
Concern-Sensitive Coupling	Concern-Sensitive Coupling (CSC)	It counts the number of components used by a given component by means of operations entirely assign to a given concern.
	Intra-component Concern-Sensitive Coupling (ICSC)	It counts the number of attributes and operations of a given component accessed by operations related to a given concern in the same component.
Concern-Sensitive Size	Number of Concern Operations (NCO)	It counts the number of operations of a given component related to a given concern.
	Number of Concern Operations (NCA)	It counts the number of attributes of a given component related to a given concern.

Table 4: Summary of the suite of concern-driven detailed design metrics

5.3.1. Concern Diffusion

Our suite of design heuristic rules uses one metric for concern diffusion: *Concern Diffusion over Components* (CDC) (Sant'Anna, 1993). This metric is similar to the Concern Diffusion over Architectural Components metric (CDAC), defined in Section 4.3.1. In fact, CDC inspired the definition of CDAC. As CDAC, CDC considers components whose purpose is to totally or partially contribute to the realization of a particular concern, enabling the designer to assess the degree of concern scattering.

CDC for a given concern *con* counts the number of components (classes and aspects) in the system design entirely assigned to *con*. The counting also includes the number of components where there is at least one attribute assigned to *con*, and the number of components where there is at least one operation assigned to *con*.

Formal Definition of CDC: Let *S* be the detailed design of a system, and *con* $\in$ *Con(S)* be a concern in *S*, CDC can be represented as:

$$
\begin{aligned}
CDC(con) = \big| & C(con) \cup \\
& \{c \mid c \in C(S) \wedge (O(c) \cap O(con)) \neq \varnothing\} \cup \\
& \{c \mid c \in C(S) \wedge (A(c) \cap A(con)) \neq \varnothing\} \big|
\end{aligned}
$$

Example: Figure 15 shows that the Factory Method pattern is present in six components: MetaObject, MetaObjFactory and respective subclasses. Therefore, the value of the CDC metric for the Factory Method concern is six.

5.3.2. Interaction between Concerns

The metric for interaction between concerns used in our heuristic rules is *Component-level Interlacing between Concerns* (CIBC). This metric is very similar to the architectural metric with the same name (Section 4.3.2). Both metrics have the same goal which is quantifying interaction between concerns which are not introduced by the dependence between components. Hence, CIBC indicates for a given concern the level of tangling in terms of how many concerns it interlaces with.

CIBC for a concern *con* counts the number of other concerns with which *con* is interlaced at the component level. A concern *con* is interlaced at the component level with another concern *con'* if *con* and *con'* have one or more components in common. At the detailed design level, this situation occurs in one of the following ways:

(i) a component is assigned to both *con* and *con'*, or

(ii) a component is assigned to *con*, and at least one member (attribute or operation) of the same component is assigned to *con'*, or

(iii) at least one member of a component is assigned to *con*, and at least one member of the same component is assigned to *con'*.

Formal Definition of CIBC. In order to represent that two concerns are interlaced at the component-level, we define the Boolean function *ComponentInterlaced*(*con, con'*), where *con* $\in$ *Con(S)* and *con'* $\in$ (*Con(S)* – *con*), as:

$$\begin{aligned} ComponentInterlaced(con, con') \Leftrightarrow & (\exists c \in C(con) : c \in C(con')) \vee \\ & (\exists c \in C(con) : \exists m \in M(c) : m \in M(con')) \vee \\ & (\exists c \in C(S) : \exists m \in M(c) : \exists m' \in M(c) : m \in M(con) : m' \in M(con')). \end{aligned}$$

Let *S* be the detailed design of a system, and *con* ∈ Con(*S*) be a concern in *S*, CIBC can be represented as:

$$CIBC(con) = \left|\{con' \mid con' \in Con(S) - \{con\} \wedge ComponentInterlaced(con, con')\}\right|.$$

Example: Figure 15 shows that the CIBC value is 1 for both Factory Method and Observer design patterns because these two concerns only interlace with each other. Component-level interlacing between Observer and Factory Method occurs in the classes Component, ConcreteBind, MetaObjEncapsule, and MetaObjComposite.

5.3.3. Concern-based Cohesion

Another metric used in our suite of design heuristic rules is Lack of Concern-based Cohesion (LCC). The definition of LCC is similar to the definition of the architectural metric with the same name (Section 4.3.4). The goal of this metric is to support designers on the observance of intra-component tangling degree. Thus, it counts for a given component the number of concerns it totally or partially implements.

LCC for a component *c* counts the number of concerns which *c* is assigned to, plus the number of distinct concerns which the operations of *c* are assigned to, plus the number of distinct concerns which the attributes of *c* are assigned to.

Formal Definition of LCC. Let *S* be the detailed design of a system, *c* ∈ *C(S)* be a component in *S*, LCC can be represented as:

$$LCC(c) = \left| Con(c) \cup \bigcup_{o \in O(c)} Con(o) \cup \bigcup_{a \in A(c)} Con(a) \right|.$$

Example: The value of LCC for the MetaObjComposite is two (Figure 15), since this component encompasses the concerns of both Factory Method and Observer patterns.

5.3.4. Concern-Sensitive Coupling

Our detailed design rules encompasses two metrics for concern-sensitive coupling: *Concern-Sensitive Coupling* (CSC), and *Intra-Component Concern-Sensitive Coupling* (ICSC). Analogously to the architectural metric with the same name, CSC aims at quantifying the contribution of a given concern to the coupling of a given component with other components. This metric is based on the assumption that if a component *c* has a method entirely related to a concern *con*, and that method calls a method or references an attribute of another component *c'*, the coupling between *c* and *c'* is due to the presence of the concern *con* in the component *c*. Again, it is important to note that the values for this metric are gathered per a pair of component and concern.

Therefore, CSC for a component *c* and a concern *con* counts the number of distinct components which have a method invoked or an attribute referenced by *c* by means of an operation entirely assign to *con*.

Formal Definition of CSC. Let S be the detailed design of a system, $c \in C(S)$ be a component in *S*, and *con* be a concern in *Con(S)*, CSC can be represented as:

$$CSC(c,con) = \left| \bigcup_{o \in CO(c,con)} IC(o) \cup \bigcup_{o \in CO(c,con)} RC(o) \right|, \text{ where } CO(c,con) = O(c) \cap O(con).$$

The sets *IC(o)* and *RC(o)* are defined in Section 5.2.1 and represent the components which have an operation invoked and an attribute referenced by an operation *o*, respectively.

Example: In the design of Figure 15, the value of CSC for the class ConcreteBind and the concern Observer (*CSC(ConcreteBind, Observer)*) is one. This occurs because the MetaObserver interface is the only component accessed by methods of ConcreteBind related to the Observer pattern: the notifyObservers() method of ConcreteBind invokes the method refresh() of MetaObserver. Sequence diagrams or source code are needed for applying CSC, as well as ICSC. In the case of this example, we used the source code.

The metric *Intra-Component Concern-Sensitive Coupling* (ICSC) is targeted at quantifying how different concerns are coupled to each other within a given component. The intra-component coupling of a given concern *con* within a given component *c* is measured in terms of the amount of operations and attributes related to other concerns within *c* are invoked and referenced by operations assigned to *con*.

The assumption behind this metric is that the higher the coupling of a concern to other concerns within a component, the harder it is to remove that concern from that component. A concern with a high intra-component coupling indicates that changing the design in order to remove that concern from the component could lead to a worse design alternative. This could occur because the concern strongly depends on information about other concerns within the component.

ICSC for a component *c* and a concern *con* counts the number of internal operations and attributes related to other concerns are invoked and referenced by operations assigned to *con*.

Formal Definition of ICSC. Let *S* be the detailed design of a system, $c \in C(S)$ be a component in *S*, and *con* be a concern in *Con(c)*, ICSC can be represented as:

$$ICSC(c,con) = \left| \bigcup_{o \in CO(c,con)} \{o' \mid o' \in IO(o) \wedge o' \notin CO(c,con)\} \right| + \left| \bigcup_{o \in CO(c,con)} \{a \in RA(o) \wedge a \notin CA(c,con)\} \right|,$$

where $CO(c,con) = O(c) \cap O(con)$ and $CA(c,con) = A(c) \cap A(con)$

The sets *IO(o)* and *RA(o)* are defined in Section 5.2.1 and represent the internal operations and attributes accessed by an operation *o*, respectively.

Example: In the design of Figure 15, the value of ICSC for the class MetaObjComposite and the concern Observer (*ICSC(MetaObjComposite, Observer)*) is one. The reason for that value is because only one attribute of MetaObjComposite not related to the Observer pattern is referenced by methods assigned to this pattern: the refresh() method of ConcreteBind references the attribute graph.

5.3.5. Concern-Sensitive Size

Two metrics for concern-sensitive size are used in our suite of design heuristic rules: *Number of Concern Operations* (NCO), and *Number of Concern Attributes* (NCA). The goal of these metrics is to quantify the contribution of a given concern to the size of a given component in terms of number of operations and attributes. Therefore, NCO and NCA count the number of operations and attributes, respectively, responsible for realizing a given concern.

NCO for a component *c* and a concern *con* counts the number of operations in *c* assigned to *con*. Similarly, NCA for a component *c* and a concern *con* counts the number of attributes in *c* assigned to *con*. The motivation for using these metrics is that a concern that comprises only few operations and attributes in a component might not be localized in that component.

Formal Definition of NCO. Let *S* be the detailed design of a system, $c \in C(S)$ be a component in *S*, and *con* be a concern in *Con(S)*, NCO can be represented as:

$$NCO(c, con) = |O(c) \cap O(con)|.$$

Formal Definition of NCA. Let *S* be the detailed design of a system, $c \in C(S)$ be a component in *S*, and *con* be a concern in *Con(S)*, NCA can be represented as:

$$NCA(c, con) = |A(c) \cap A(con)|.$$

Example: In the design of Figure 15, the value of the NCO metric for the class ConcreteBind and the concern Observer (*NCO(ConcreteBind, Observer)*) is 3, since there are three methods related to the Observer pattern on this class: addObserver(), removeObserver(), and notifyObservers(). The value of NCA for the same class and concern ((*NCO(ConcreteBind, Observer)*) is one, since there is only the observers attributed related to the Observer pattern in the ConcreteBind class.

5.4. Concern-Driven Design Heuristic Rules

This section defines design heuristic rules as mechanisms for supporting concern-sensitive modularity analysis. These rules are defined in terms of combined information collected from concern-driven metrics (Section 5.3) and conventional metrics. Table 5 presents the conventional metrics which are used by the proposed heuristics. The heuristic rules provide developers with complementary high-level assessment means rather than exclusively working with metrics. As such, each heuristic expression embodies knowledge about the modular realizations of concerns in a design. The motivation of concern-sensitive heuristics is to minimize the shortcomings of conventional metrics-based heuristics discussed in Section 5.1.

Metrics	Definitions
Number of Components (NC)	It counts the number of components (classes and aspects) of a system's design.
Number of Attributes (NOA)	It counts the number of attributes of a given component.
Number of Operations (NOO)	It counts the number of operations (methods and advice) of a given component.
Coupling Between Components (CBC)	It counts the number of components from which a given component invokes a method or references an attribute.

Table 5: Conventional metrics used in the definition of the heuristic rules

All the heuristic rules defined in this section are expressed using conditional statements in the following form (Tekinerdoğan & Akşit, 1998):

IF <condition> THEN <consequence>.

The condition part encompasses one or more outcomes of metrics related to the design concern under analysis. As we will see, the heuristic rules classify each concern into categories which describe the way it is modularized. Examples of categories that will be described in the following sections are: isolated concern, tangled concern, high scattered concern, and so forth. The following rule, for instance, is based on the outcomes of CDC and NC metrics, and classifies a tangled concern as a highly scattered concern, if the condition holds.

IF CDC / NC of Concern ≥ 0.5 THEN Tangled Concern is Highly Scattered

The heuristic rules are structured in such a way that the classification is systematically refined into a more specialized category. If the condition is not satisfied, then the concern analysis is concluded and the concern classification is not refined. If the condition holds, the role of the consequence part is to describe a change or refinement of the target concern classification. Some categories, such as *isolated concern*, indicate a well modularized concern. Other categories, such as *highly scattered concern*, warn about poorly modularized concern. The generated warnings encompass information that helps the designers to concentrate on certain concerns or parts of the design which are potentially problematic. The proposed concern heuristics suite is structured in two major parts: (i) crosscutting concern analysis (Sections 5.4.1 and 5.4.2), and (ii) detection of specific design flaws (Section 5.4.3).

5.4.1. Crosscutting Concern Analysis

This section presents a suite of design heuristic rules to classify concerns in terms of the degree of tangling and scattering. These rules classify a concern into six categories: *isolated*, *tangled*, *little scattered*, *highly scattered*, *well-encapsulated*, and *crosscutting*. Figure 16 presents a diagram which defines the relationship between heuristic rules and concern categories. The ellipses represent concern categories, and the labeled arrows represent the heuristic rules. Each heuristic rule is a transition relationship between two concern classifications. The definitions of the rules are presented in Figure 17. Figure 16 also shows the order in which the heuristics should be applied.

A *tangled* concern is a concern which is interleaved with other concerns in one particular component (class or aspect). If the concern is not tangled in any component, it is considered as *isolated*. A concern is *isolated* if it is the only concern in all components that realize it.

A *scattered* concern is a concern which is spread over multiple components. Our classification makes a distinction between *highly* scattered and *little scattered* concerns based on the number of affected components. A *highly scattered* concern is a concern spread over many components. A *little scattered* concern is concern spread over few components.

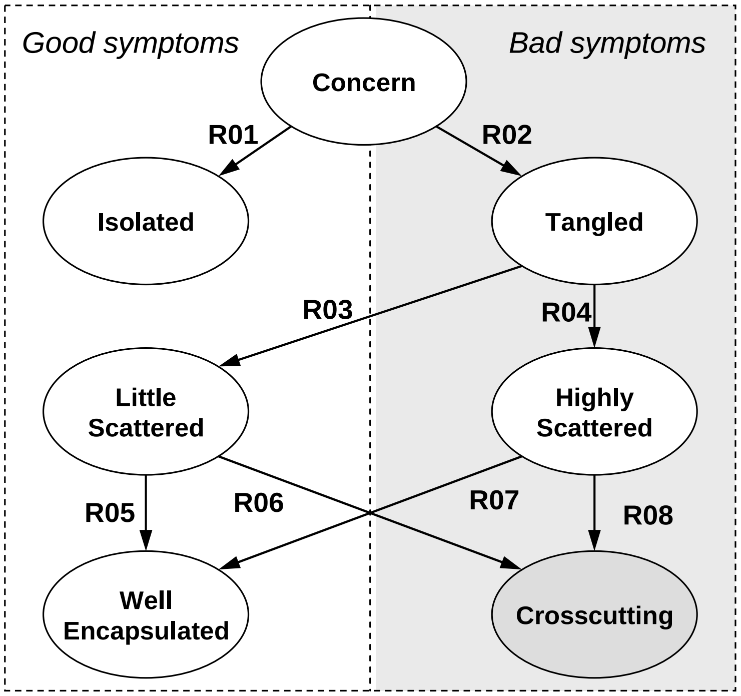

Figure 16: Concern classification

A *well-encapsulated* concern is a concern that is the dominant concern in all the components where it is. We define dominant concern of a component as the concern that is assigned to most of the attributes and operations of that concern. The reasoning behind this classification is that a concern is not harmful to the classes or aspects in which it is dominant, and, therefore, it does not need to be removed from them.

Note that there is a difference between *isolated* and *well-encapsulated* concerns. An *isolated* concern is also dominant in every component where it is present. However, it is the only concern in those components. On the other hand, in spite of being dominant in all components where it is, a *well-encapsulated* concern is tangled with other concerns that are present in at least one of those components. These other concerns are realized by less attributes and operation than the dominant concern. These are the concerns classified as *crosscutting*.

Crosscutting concerns generate warnings of inadequate separation of concerns and, consequently, opportunities for refactoring (Fowler, 1999; Monteiro & Fernandes, 2005). Although both *highly* and *little scattered* concerns can be crosscutting concerns, *highly scattered* crosscutting concerns tend to be more harmful to the design modularity. Therefore, it is important to classify a concern as *highly* or *little scattered* before classifying it as *crosscutting*.

R01 - *Isolated*:
if CIBC = 0
then CONCERN is ISOLATED

R02 - *Tangled*:
if CIBC > 0
then CONCERN is TANGLED

R03 - *Little Scattered*:
if CDC / NC of CONCERN < 0.5
then TANGLED CONCERN is LITTLE SCATTERED

R04 - *Highly Scattered*:
if CDC / NC of CONCERN ≥ 0.5
then TANGLED CONCERN is HIGHLY SCATTERED

R05 - *Well Encapsulated*:
if (NCA / NOA ≥ 0.5) and (NCO / NOO ≥ 0.5) for every component with CONCERN
then LITTLE SCATTERED CONCERN is WELL-ENCAPSULATED

R06 - *Crosscutting*:
if (NCA / NOA < 0.5) and (NCO / NOO < 0.5) for at least one component with CONCERN
then LITTLE SCATTERED CONCERN is CROSSCUTTING

R07 - *Well Encapsulated*:
if (NCA / NOA ≥ 0.5) and (NCO / NOO ≥ 0.5) for every component with CONCERN
then HIGHLY SCATTERED CONCERN is WELL-ENCAPSULATED

R08 - *Crosscutting*:
if (NCA / NOA < 0.5) and (NCO / NOO < 0.5) for at least one component with CONCERN
then HIGHLY SCATTERED CONCERN is CROSSCUTTING

Figure 17: Design heuristic rules for crosscutting concern analysis

The heuristic rules for crosscutting concern analysis are shown in Figure 17. The rules are numbered from R01 to R08 and include the classification of concerns shown in the diagram in Figure 16. The first two heuristic rules, R01 and R02, use the metric Component-level Interlacing between Concerns (CIBC) to classify the concern as *isolated* or *tangled* (Figure 17). If the CIBC value is zero, it means that there is only that concern in all the components which implement it and, therefore, the concern is *isolated*. However, if CIBC is higher than zero, it means that the concern is interlaced (*tangled*) with other concerns in at least one component, e.g. Factory Method and Observer patterns in the design of Figure 15.

Rules R03 and R04 verify whether a concern, besides tangled, is scattered over multiple components. These heuristics use the metrics Concern Diffusion over Components (CDC) and Number of Components (NC) in order to calculate the percentage of system components affected by the concern of interest. Based on this percentage, the concern is classified as *highly scattered* or *little scattered*. As we will discuss later in this section, one of the most sensitive parts in a heuristic rule is the selection of threshold values. Our strategy for these rules is to use 50% as the threshold. Developers should be aware of *highly scattered* concerns because they can potentially cause design flaws, such as Shotgun Surgery (Fowler, 1999) (Section 5.1).

Rules R05 and R06 decide whether a *little scattered* concern is either a *well-encapsulated* or *crosscutting* concern. Rules R07 and R08 perform similar analyses for a *highly scattered* concern. These four rules use the metrics Number of Concern Attributes (NCA) and Number of Concern Operations (NCO) and two size metrics presented in Table 5: Number of Attributes (NOA) and Number of Operations (NOO). They calculate for each component the percentage of attributes and operations which implements the concern being analyzed.

The role of the heuristics R05 and R07 is to detect components that dedicate a large number of attributes and operations (more than 50%) to realize the analyzed concern. If so, that concern is regarded as the dominant concern of those components. If a concern is dominant in every component where it is, this concern is classified as *well-encapsulated*.

A concern is classified as *crosscutting* (rules R06 and R08) if the percentage of attributes and the percentage of operations related to the concern are low (less than 50%) in at least one component. Hence, a concern is classified as *crosscutting* if it is located in at least one component which has another concern as dominant.

5.4.2. Octopus and Black Sheep

This section defines design heuristic rules for classifying crosscutting concerns as *octopus* and *black sheep*. These categories are inspired on the categories with the same names proposed by Ducasse et al (2006). *Black sheep* is a concern that crosscuts multiple components, but is realized by very few attributes and operations in all these components. *Octopus* is a crosscutting concern which is dominant in some components (octopus' body), but is realized by only few attributes and operations in other components (octopus' arms).

Therefore, *black sheep* and *octopus* are actually specialized categories of *crosscutting* concerns. Figure 18 shows two heuristic rules, R09 and R10, which aim at identifying *black sheep* and *octopus* concerns, respectively. Figure 18 also defines *condition A* (*Little Dedication*) and *condition B* (*High Dedication*) used in these rules. We explicitly separate the conditions from the heuristic rules in order to make the rules easier to understand and also to reuse the *Little Dedication* condition in both heuristics. In rules R09 and R10, a concern previously classified as *crosscutting* (Section 5.4.1) is thoroughly inspected in terms of (i) how much each component dedicates to that concern, and (ii) how many components have high and low dedication to such concern.

The heuristic rule R09 classifies a crosscutting concern as *black sheep* if all components which have this concern dedicate only a few percentage points of attributes and operations to that concern (less than 33%). The next rule R10 verifies if crosscutting concerns not classified as *black sheep* are potential *octopus*. According to this heuristic, a concern is classified as *octopus* if every component realizing parts of this concern has either little dedication (*condition A*) or high dedication (*condition B*) to it. Besides, at least two components have to be little dedicated to the concern (octopus' arms) and at least one component has to be highly dedicated (octopus' body). We define a component as highly dedicated to a concern when the percentage of attributes and operations are higher than 67%.

```
Condition A - Little Dedication:
(NCA / NOA < 0.33) and (NCO / NOO < 0.33)

Condition B - High Dedication:
(NCA / NOA ≥ 0.67) and (NCO / NOO ≥ 0.67))

R09 - Black Sheep:
if (Little Dedication) for every component with CONCERN
then CROSSCUTTING CONCERN is BLACK SHEEP

R10 - Octopus:
if ((Little Dedication) or (High Dedication) for every component with CONCERN)
   and ((Little Dedication) for at least 2 components with CONCERN)
   and ((High Dedication) for at least 1 component with CONCERN)
then CROSSCUTTING CONCERN is OCTOPUS
```

Figure 18: Design heuristics rules for Black Sheep and Octopus

5.4.3. Concern-Aware Bad Smells

This section focus on how concern-driven heuristic rules can also be applied to detect well-known design flaws, such as the bad smells proposed by Fowler (1999). We defined heuristics for two bad smells: Feature Envy (Fowler, 1999) and Shotgun Surgery (Fowler, 1999). The first bad smell, Feature Envy, is related to the misplacement of operations. It occurs when an operation seems more interested in a component other than the one it actually is in (Fowler, 1999). As stated before, Shotgun Surgery occurs when a change in a characteristic (or concern) of the system implies many changes to a lot of different places (Fowler, 1999). The reason for selecting these two bad smells is twofold. First, previous work already claimed they are related to inadequate modularization of concerns (Monteiro & Fernandes, 2005). Second, existing heuristic rules for detecting these bad smells are representative of those rules based on metrics for coupling and cohesion. As mentioned before, coupling and cohesion are the most used attributes on conventional modularity measurement approaches.

Figure 19 shows our concern-sensitive heuristic rules for detecting the two aforementioned bad smells. While the previous rules (R01 to R10) are applied per concern, the rules in this section (R11 and R12) are applied for each pair "component-concern". The first rule, R11, aims at detecting Feature Envy and uses a combination of concern-sensitive and conventional metrics for coupling and size.

According to R11, to be considered Feature Envy, a crosscutting concern has to satisfy two conditions: "condition C" (*High Inter-Component Coupling*) and "condition D" (*Low Intra-Component Coupling*) (Figure 19). In other words, this rule states that operations related to a given concern within a given component are suspect of Feature Envy if: (i) those operations are responsible for imposing high coupling with other components ("condition C"), and (ii) the same operations are weakly coupled to other concerns within the component ("condition D").

The assumption behind this rule is that a concern that imposes high coupling to a component might be removed from that component. Besides, if that concern is internally weakly coupled it might be easy to remove it from the component. Note that when we say "concern" here, we mean "the operations to which the concern is assigned". This is the main difference of our rule from Marinescu's rule (Marinescu, 2002): the analysis provided by our rule takes into account all operations related to a concern together rather than each operation in isolation.

Condition C - *High Inter-Component Coupling*:

(CSC / CBC) > ((NCA+NCO) / (NOA+NOO))

Condition D - *Low Intra-Component Coupling*:

(ICSC / ((NOA+NOO)-(NCA+NCO))) < ((NCA+NCO) / (NOA+NOO))

R11 - *Feature Envy*:

if (*High Inter-Component Coupling*) and (*Low Intra-Component Coupling*) and (LCC > 1)

then CROSSCUTTING CONCERN is FEATURE ENVY

R12 - *Shotgun Surgery*:

if CONCERN is (*Tangled*) and (*Highly Scattered*) and (*Crosscutting*)

then CROSSCUTTING CONCERN is SHOTGUN SURGERY

Figure 19: Heuristic rules for detecting bad smells

In order to quantify whether the inter-component coupling imposed by a concern to a component is high, "condition C" calculates the percentage of coupling related to a concern: CSC/CBC. CSC stands for Concern-Sensitive Coupling metric (Section 5.3.4) and CBC stands for Coupling Between Components metric (Table 5) The inter-component coupling is considered high when the percentage of coupling related to a concern (CSC/CBC) is higher than the percentage of operations and attributes that realize that concern within the assessed component ((NCA+NCO) / (NOA+NOO)). Similar computation is performed for identifying low intra-component coupling. In this case, the Intra-component Concern-Sensitive Coupling metric (ICSC) is used.

The rule R12 (Figure 19) is intended to detect Shotgun Surgery. Differently from the previous rules, R12 is composed of the outcomes from other heuristics. More precisely, a concern is classified as Shotgun Surgery if it was previously identified as *Tangled* (R02), *Highly Scattered* (R04), and *Crosscutting* (R08). R12 takes a concern classified as *crosscutting* and checks its previous classifications. If that concern, besides of being crosscutting, is high scattered, it is considered as a suspect of Shotgun Surgery. It is important to bear in mind that, according to our classification (Figure 16), a little scattered concern can also be classified as crosscutting. This occurs whenever a concern, even spread over only few components, is not dominant in some of them. However, in this case, there is no warning about Shotgun Surgery, because changing that concern would not generate changes in many components.

5.4.4. The Issue of Threshold Values

Before closing this chapter, we would like to point out to an aspect that has a decisive influence on the accuracy of a heuristic rule: the threshold values used in parameterizing any heuristic rule. The problem is far from being new and it characterizes intrinsically any metrics-based approach. In most of the cases setting the threshold values is a highly empirical process and it is guided by similar past experiences and by hints from metrics' authors (Lorenz & Kidd, 1994).

In the definition of our heuristic rules (Figure 17, Figure 18, and Figure 19), we selected the thresholds based on our previous experience applying concern-driven metrics (Sant’Anna et al., 2003, 2004; Kulesza et al., 2006; Garcia et al., 2004a, 2005, 2006; Figueiredo et al., 2008b; Greenwood et al., 2007a; Cacho et al., 2006a). In addition, we decided to set the thresholds rather permissive, as it is preferable to get more false positive results, rather than losing a large number of real flaws due to a very strict threshold value. Also, we used meaningful threshold values, such as 0.25 (1/4), 0.33 (1/3), 0.5 (1/2), 0.67 (2/3), and 0.75 (3/4), as suggested by Lanza & Marinescu (2006). Nevertheless, the threshold values used in our rules are not final. Rather, we believe that further use of the rules will allow us to adjust the threshold values in order to increase the number of correctly detected samples.