8
Evaluation of the Detailed Design Metrics and Heuristics

The goal of this chapter is to evaluate the applicability and effectiveness of the detailed design metrics and heuristic rules in order to detect modularity-related design problems. In order to evaluate the heuristic rules, we carried out a study involving six systems (Section 8.1). Four of them are medium-sized academic prototypes and the other two are real-life software projects. We have analyzed both object-oriented and aspect-oriented designs of such systems, which encompass heterogeneous forms of crosscutting and non-crosscutting concerns. We also conducted a study in which students used different suites of detailed design metrics to identify some bad smells (Section 8.2). Then, we compared the performance of conventional and concern-driven metrics.

8.1.
Design Heuristic Rules Study

This section introduces a systematic evaluation of the concern-sensitive heuristic rules (Section 5.4), which involved 22 distinct concerns addressed by the object-oriented and aspect-oriented design of six systems. These concerns were selected because they exercise different heuristic rules. Section 8.1.1 describes the target systems and concerns in more details. The heuristics rules empirical study was conducted under two dimensions: (i) evaluation of the rules accuracy to identify and classify crosscutting concerns as well as other modularity flaws in both OO and AO systems (Section 8.1.2), (ii) evaluation of the rules usefulness to detect bad smells in comparison with conventional heuristics (Section 8.1.3). We also discuss the applicability of the heuristic rules for addressing metrics limitations (Section 8.1.4).

The whole study involved the following activities:

- Mapping of the design elements of both AO and non-AO versions of the systems to the considered concerns.

- Computation of the detailed design metrics needed for the application of the design rules. The concern-driven metrics (Table 4) were manually computed, while the conventional metrics (Table 5) were computed with the AJATO tool support (Figueiredo et al., 2006).
- Application of the design heuristic rules R1 to R12. The rules were manually applied.
- Comparison of the results of rules R1 to R2 with specialist's opinion and literature assertion (first dimension of evaluation) (Section 8.1.2).
- Comparison of the results of the rules R11 and R12 with results of Marinescu's rules (second dimension of evaluation) (Section 8.1.3).

8.1.1. Target Systems

Previous works (Cacho et al., 2006a; Filho et al., 2006, Garcia et al., 2006b; Garcia et al., 2004a; Greenwood et al., 2007a; Hannemann & Kiczales, 2002) applied modularity-related metrics (including some of our concern-driven ones) to a number of comparative empirical studies. We selected six of the systems used in those studies in order to apply and assess the accuracy of our detailed design metrics and heuristics rules. One of these systems is the Health Watcher (Soares et al., 2002; Greenwood et al., 2007a), already exploited in previous sections. Besides Health Watcher, the study also involved another real software system, namely Eclipse CVS plug-in (Eclipse Foundation, 2007b). The other four systems are academic prototypes carefully designed with modularity attributes as main drivers (Figueiredo et al., 2006, Cacho et al., 2006a; Garcia et al., 2004a; Hannemann & Kiczales, 2002).

Table 23 provides a list of the systems and the concerns evaluated in each of them. The first system is, in fact, a design pattern library developed by Hannemann & Kiczales (2002) in Java and AspectJ (The AspectJ Team, 2007; Kickzales et al., 2001). Likewise the Hannemann & Kiczales' work, in the study involving their library we treated each pattern role as a concern because the roles are the primary sources of crosscutting structures. The second system is a middleware system (Cacho et al., 2006a, 2006b) and the third one is the AJATO measurement tool (Figueiredo et al., 2006; Cacho et al., 2006a). In these systems

we consider each design pattern as a concern in other to investigate their compositions in those systems.

Systems	Nature of the Concerns		Concerns
Hannemann & Kiczales' design pattern library	Design Patterns	Builder	Director role
		Chain of Resp.	Handler role
		Factory Method	Creator role
		Observer	Observer role
			Subject role
		Mediator	Colleague role
			Mediator role
OpenOrb Middleware	Design patterns and their compositions.		Fact. Method
			Observer
			Façade
			Singleton
AJATO Measurement tool			Prototype
			Interpreter
			Proxy
CVS core plugin	Recurring architectural concerns		Exception Handling
			Business
Health Watcher			Concurrency
			Distribution
			Persistence
Portalware	Domain-specific concerns		Adaptation
			Autonomy
			Collaboration

Table 23: Systems and concerns used in the evaluation study

The following two systems in Table 23 are the Eclipse CVS core plugin (Eclipse Foundation, 2007b; Filho et al., 2006) and the Health Watcher (Soares et al., 2002; Greenwood et al., 2007a); in both we evaluate recurring widely-scoped concerns. Finally, the last target application is a multi-agent system (MAS) for managing Web portals, called Portalware (Garcia et al., 2004a). The focus here was specifically on MAS-domain concerns. These systems were selected for several reasons. First, they encompass both aspect-oriented (AO) and object-oriented (OO) implementations. Second, a number of concern-oriented and conventional metrics have been previously used for assessing the design modularity of both implementations of these systems. This allowed us to evaluate to what extent concern-aware heuristics is helpful or not to enhance a design assessment exclusively based on metrics.

The third reason is the heterogeneity of the concerns found in these systems (Table 23, column 2), which include widely-scoped architectural concerns, such as persistence and exception handling, and concern that only manifest themselves in the detailed design or implementation, such as design patterns. They encompass

different characteristics and different degrees of complexity. Forth, the systems are representatives of different application domains, ranging from simple design pattern instances to real-life Web-based applications, reflective middleware systems, and multi-agent systems. Finally, such systems also serve as effective benchmarks because they involve scenarios where it has been far from trivial to decide when "aspectizing" or not certain concerns (Cacho et al., 2006a; Filho et al., 2006, 2007; Garcia et al., 2004a; Greenwood et al., 2007a).

8.1.2. Accuracy of the Heuristic Rules

This section aims at evaluating the accuracy of the design heuristic rules for crosscutting concern analysis (rules R01 to R08) (Section 5.4.1), including also the rules which classify concerns as octopus and black sheep (rules R09 and R10) (Section 5.4.2). We evaluated the accuracy of these heuristics comparing their outcomes with specialists' opinion or with literature assertion about the concerns involved in these studies (Cacho et al., 2006a; Filho et al., 2006, 2007; Garcia et al., 2006b; Garcia et al., 2004a; Greenwood et al., 2007a; Hannemann & Kiczales, 2002; Monteiro & Fernandes, 2005). Specialists are researchers that participated in development, maintenance and assessment of the systems and have documented their observations with respect to the concerns in these systems.

We applied the rules R01 to R10 (Section 5.4.1 and Section 5.4.2) in order to assess and classify the 22 aforementioned concerns (Table 23) as: isolated, well-encapsulated, crosscutting, black sheep or octopus. In fact, we applied the rules 44 times: 22 times for the concerns in the object-oriented design of the systems, and 22 times for classifying the same instances of concerns in the aspect-oriented designs. Then, we count how many times the literature and/or specialists agreed with the heuristics results and how many times they did not agree.

Table 24 describes how we confronted the specialists/literature opinion with the rules outcomes. First we classify the concerns according to literature/specialists opinion as "crosscutting" or "non-crosscutting" (column 1). A concern is classified as crosscutting, if the literature/specialists show evidences that: (i) if modularized based on object-oriented abstractions, the concern ends up scattered and tangled with other concerns, and (ii) the concern is better

modularizes by an aspect-oriented design, which eliminates its crosscutting nature. On the other hand a concern is classified as non-crosscutting, if the literature/specialists present evidences that the concern is well modularized by object-oriented abstractions and do not show a crosscutting nature.

Table 24 also shows what classification is expected to be provided by the rules (column 2). The expected classification is based on the literature/specialists opinions and the version of the design where the rules are applied: object-oriented (OO) or aspect-oriented (AO). The last two columns described how the rules are evaluated based on the information the first two columns and the classification provided by the rules:

- The rules succeed if the classification provided by them is one of the expected classifications.
- The rules fail if the expected classification is crosscutting, black sheep or octopus and the rules classify the concern as isolated or well-encapsulated; this is considered an occurrence of “false negative”.
- The rules also fail if the expected classification is isolated or well-encapsulated and the rules classify the concern as crosscutting, black sheep or octopus; this is considered an occurrence of “false positive”.

Specialists opinion	**Expected classification based on the version (OO or AO)**		**Classification provided by the rules**	**Rules Evaluation**
Crosscutting concern	OO	Crosscutting, black sheep or octopus	Crosscutting, black sheep or octopus	Succeeded
			Isolated or well-encapsulated	Failed (false negative)
	AO	Isolated or well-encapsulated	Crosscutting, black sheep or octopus	Failed (false positive)
			Isolated or well-encapsulated	Succeeded
Non-crosscutting concern	OO	Isolated or well-encapsulated	Crosscutting, black sheep or octopus	Failed (false positive)
			Isolated or well-encapsulated	Succeeded
	AO	Isolated or well-encapsulated	Crosscutting, black sheep or octopus	Failed (false positive)
			Isolated or well-encapsulated	Succeeded

Table 24: Comparing the specialists opinion with the rules outcomes

Table 25 provides an overview of the application of heuristic rules R01 to R10 (Section 5.4.1 and Section 5.4.2) to the object-oriented version of target

systems. The leftmost column lists the six applications. Note that the study related to Hannemann & Kiczales' design pattern library (first study) is subdivided in terms of design patterns (column 2), and the assessed concerns are roles of those patterns (column 3). The third column shows all 22 concerns analysed in the respective studies. The forth column presents the classification of each concern according to specialists and literature claims (Cacho et al., 2006a; Filho et al., 2006, 2007, Garcia et al., 2006b; Garcia et al., 2004a; Greenwood et al., 2007a; Hannemann & Kiczales, 2002; Monteiro & Fernandes, 2005). The two possible values are: "ccc", which stands for "crosscutting concern", and "non-ccc", which stands for "non-crosscutting concern". As previously explained, this information is used to check whether the final concern classification provided by the rules failed or not.

Systems	Nature of the Concerns		Concerns	Special.	Heuristic Rules										Results
					01	02	03	04	05	06	07	08	09	10	
Hannemann & Kiczales' design pattern library	Design Patterns	Builder	Director role	non-ccc	n	y	y	n	y	n					well-e. / hit
		Chain of Resp.	Handler role	ccc	n	y	n	y			y	n			well-e. / FAIL
		Factory Method	Creator role	non-ccc	n	y	n	y			n	y	n	n	ccc / FAIL
		Observer	Observer role	ccc	n	y	n	y			n	y	n	y	octopus / hit
			Subject role	ccc	n	y	n	y			n	y	n	y	octopus / hit
		Mediator	Colleague role	ccc	n	y	n	y			n	y	n	y	octopus / hit
			Mediator role	ccc	n	y	n	y			y	n			well-e. / FAIL
OpenOrb Middleware	Design patterns and their compositions.		Fact. Method	non-ccc	n	y	y	n	y	n					well-e. / hit
			Observer	ccc	n	y	y	n	n	y			n	y	octopus / hit
			Façade	non-ccc	y	n									isolated / hit
			Singleton	ccc	n	y	y	n	n	y			y	n	b-sheep / hit
Measurement tool			Prototype	ccc	n	y	y	n	n	y			y	n	b-sheep / hit
			Interpreter	ccc	n	y	y	n	n	y			n	y	octopus / hit
			Proxy	ccc	n	y	y	n	n	y			n	n	ccc / hit
CVS plugin	Recurring architectural concerns		Exc. Handling	ccc	n	y	n	y			n	y	n	y	octopus / hit
Health Watcher			Business	non-ccc	n	y	n	y			y	n			well-e. / hit
			Concurrency	AO	n	y	n	y			n	y	n	y	octopus / hit
			Distribution	AO	n	y	n	y			n	y	n	y	octopus / hit
			Persistence	AO	n	y	n	y			n	y	n	y	octopus / hit
Portalware	Domain-specific concerns		Adaptation	AO	n	y	y	n	n	y			n	n	ccc / hit
			Autonomy	AO	n	y	y	n	n	y			n	n	ccc / hit
			Collaboration	AO	n	y	n	y			n	y	n	y	octopus / hit

Table 25: Results of the heuristics application in the object-oriented version of the systems

The columns labeled with 01 to 10 present if each rule was satisfied or not. When we say that a rule was satisfied, we mean that the condition part of the rule (Section 5.3) was satisfied. If the rule was satisfied the cell is filled with “y”, which stands for “yes”. If the rule was not satisfied, the corresponding cell is filled with “n”, which stands for “no”. Blank cells mean that the rule is not applicable.

Finally, the last column indicates: (i) how the heuristics classify each concern, and (ii) if the classification matches with the specialists’ opinion (‘hit’) or not (‘FAIL’). The possible classifications are isolated, well-encapsulated (‘well-e.’), crosscutting concern (‘ccc’), octopus and black sheep (‘b-sheep’). For instance, the “y” label in the forth row (related to the Observer role concern) and the column before the last indicates that the R10 rule was satisfied when applied to the Observer role concern. As the R10 rule classifies the concerns as octopus, the last column of this row is labeled with “octopus/hit”, which also indicates that this classification matches with specialists’ opinion. Table 26 presents the general results for the application of the heuristic rules in the aspect-oriented implementation of each system.

Table 27 provides overviews of the hits, false positives and false negatives of the rules for the 44 concern instances involved in this study (22 in the OO version and 22 in AO version). The rows of Table 27 are organized in three parts: the object-oriented instances (OO), the aspect-oriented instances (AO) and the general data for both paradigms (OO + AO). Each row describes the absolute number and the percentage in relation to the total of concerns.

Table 27 shows that the heuristics failed in 15.9% of the cases (5 false positives and 2 false negatives). In the object-oriented designs, there were one false positive and two false negatives. The false positive occurs with the creator role of the Factory Method pattern (Table 25). In this pattern, the class which plays the creator role has a lot of elements (mainly methods) related to other concern. This made the rules interpret that concern as the dominant concern of the class, instead of the creator role concern. As a consequence, the creator role was classified as crosscutting, when there is no problem with its design modularity.

Systems	Nature of the Concerns		Concerns	Special.	Heuristic Rules										Results
					01	02	03	04	05	06	07	08	09	10	
Hannemann & Kiczales' design pattern library	Design Patterns	Builder	Director role	non-ccc	n	y	y	n	y	n					well-e. / hit
		Chain of Resp.	Handler role	ccc	n	y	y	n	y	n					well-e. / hit
		Factory Method	Creator role	non-ccc	n	y	n	y			y	n			well-e. / hit
		Observer	Observer role	ccc	n	y	y	n	n	y			n	n	ccc / FAIL
			Subject role	ccc	n	y	y	n	n	y			n	y	octopus/FAIL
		Mediator	Colleague role	ccc	n	y	y	n	n	y			n	n	ccc / FAIL
			Mediator role	ccc	n	y	y	n	n	y			n	n	ccc / FAIL
OpenOrb Middleware	Design patterns and their compositions.		Fact. Method	non-ccc	y	n									isolated / hit
			Observer	ccc	y	n									isolated / hit
			Façade	non-ccc	y	n									isolated / hit
			Singleton	ccc	y	n									isolated / hit
Measurement tool			Prototype	ccc	y	n									isolated / hit
			Interpreter	ccc	y	n									isolated / hit
			Proxy	ccc	y	n									isolated / hit
CVS plugin	Recurring architectural concerns		Exc. Handling	ccc	n	y	n	y			n	y	n	y	octopus / hit[6]
Health Watcher			Business	non-ccc	n	y	n	y			y	n			well-e. / hit
			Concurrency	AO	y	n									isolated / hit
			Distribution	AO	n	y	y	n	y	n					well-e. / hit
			Persistence	AO	n	y	n	y			y	n			well-e. / hit
Portalware	Domain-specific concerns		Adaptation	AO	y	n									isolated / hit
			Autonomy	AO	y	n									isolated / hit
			Collaboration	AO	y	n									isolated / hit

Table 26: Results of the heuristics application in the aspect-oriented version of the systems

Versions	Hits (%)	False Positives (%)	False Negatives (%)	Total (%)
OO	19 (86.4)	1 (4.5)	2 (9.1)	22 (100)
AO	18 (81.8)	4 (18.2)	0 (0.0)	22 (100)
OO + AO	37 (84.1)	5 (11.4)	2 (4.5)	44 (100)

Table 27: Statistics about the application of the heuristic rules

The heuristics have presented two occurrences of false negatives in the assessment of the object-oriented designs: (i) Chain of Responsibility pattern, and (ii) Mediator role of the Mediator pattern (Table 25). The Chain of Responsibility pattern was not detected as a crosscutting concern because the pattern instance

[6] The exception handling concern was not totally aspectized in the CVS plugin. Therefore, we did not consider that the rules failed for classifying this concern as octopus in the aspect-oriented version of this system.

provided by Hannemann & Kiczales (2002) is too simple (due to its library nature) in order to expose the pattern's crosscutting nature. In fact, classes playing the Handler role have only three members: one attribute, one method and one constructor. The first two members realize the Chain of Responsibility pattern. Because of this the heuristics erroneously consider the Chain of Responsibility concern as the main purpose of those classes, i.e., the dominant concern, instead of identifying it as secondary to this classes, and, as a consequence, crosscutting. A similar situation occurs in the Mediator pattern (Gamma et al., 1995) (Table 25).

Table 27 presents four false positives in the heuristic assessment of the aspect-oriented designs. These false positives occurred when a pattern defines more than one role. This was the case of Observer pattern (Subject and Observer roles) and Mediator pattern (Mediator and Colleague roles) (Table 26). When applying the metrics to Hannemann & Kiczales (2002) library, we considered each role as a separate concern, as stated before and shown in Table 26. Although each of these patterns was successfully modularized using aspects, their roles are tangled to each other in the aspects. As a result, the rules classified than as crosscutting. We considered that the rules failed in this case, because the patterns as a role are well modularized and it does not make sense to separate their roles from each other.

8.1.3. Detection of Specific Design Flaws

We also evaluated the usefulness of our heuristic rules in order to detect specific design flaws. In particular, we applied rules R11 to R12 (Section 5.4.3) to identify Shotgun Surgery and Feature Envy bad smells (Fowler, 1999) in the object-oriented design of the target systems. We also applied the conventional heuristic rules proposed by Marinescu (2002) for detecting the same bad smells and compared the results from the two suites of rules.

In order to perform this comparison, we proceeded with the following steps. First we applied each suite of rules to the same systems independently. The application of the rules pointed out design fragments suspect of having one of the two aforementioned bad smells. After that, we undertook a manual investigation

in which we inspected all the suspect design fragments. In circumstances when it was not clear about the existence of a bad smell, we contacted professionals and researchers with long-term experience on the development and assessment of the target case studies. The manual investigation allowed us to verify whether the suspect design fragments were indeed affected by the design flaw suggested by the heuristic rules.

After the inspection, the total number of suspects for each bad smell were classified in two categories: (i) “Hits”, which contains those suspect fragments that have confirmed to be affected by the bad smell that the heuristic rule claims to find, and (ii) “False Positive”, which includes those suspect fragments that revealed not to be affected by the bad smell supposed to be captured by the rule. Table 28 summarizes the results of applying both concern-sensitive and conventional design heuristics rules. This table also shows the total number of hits and false positives for each bad smell and the percentage that this value represents from the total number of suspect fragments (under brackets).

It is important to bear in mind that the results in Table 28 are given in different points of view. The values for concern-driven rules represent number of concerns, since rules R11 and R12 classify concerns (Section 5.4.3). While the values for the conventional rules represent number of classes (Shotgun Surgery) or operations (Feature Envy). Therefore, when we say that the number of hits for the concern-driven rules for identifying Shotgun Surgery was eight, we mean that these rules identified eight concerns affected by this bed smell. While when we say that the conventional rules had nine hits while seeking for Shotgun Surgery, we mean that these rules identified nine classes affected by this bed smell.

Bad Smell	**Concern-driven Rules**		**Conventional Rules**	
	Hits (%)	**False Positive (%)**	**Hits (%)**	**False Positive (%)**
Shotgun Surgery	8 (89%)	1 (11%)	9 (56%)	7 (44%)
Feature Envy	3 (100%)	0 (0%)	1 (17%)	5 (83%)

Table 28: Concern-driven vs. Conventional heuristic rules: statistics about the application of the heuristic rules for detecting bed smells

We can see from Table 28 that the concern-driven heuristics presented superior accuracy than the conventional rules for detecting the two studied bad smells. The former presented no more than 20% of false positives, whereas the

latter pointed out very high number of false positives. We could verify by means of manual inspection that this advantage in favor of our rules was mainly caused by the fact that they are concern sensitive. For instance, many of the false positives of the conventional rule for Shotgun Surgery occurred because this metric does not distinguee coupling between classes of the same concerns and classes with different concerns.

8.1.4. Solving Measurement Shortcomings

This section discusses problems that affect not only conventional metrics but also the use of metrics in general, including concern-driven metrics. We discuss here drawbacks related to the use of concern-driven metrics, and how these drawbacks motivate the use of these metrics in the context concern-driven heuristics rules. We classify the limitations into three categories: (i) false crosscutting warnings, (ii) hiding concern-sensitive flaws, and (iii) controversial outcomes from concern measures.

False crosscutting warnings

The problem of *false crosscutting warnings* occurs when the concern-driven metrics erroneously warn the developer of a possible crosscutting concern. However, a subsequent careful analysis of the design shows that the concern is well encapsulated. Figure 61 presents an example of this problem category in an instance of the Factory Method pattern (Gamma et al., 1995). Consider the Factory Method pattern as the assessed concern and apply the Concern Diffusion over Components metric (CDC) (Section 5.3.1). The obtained result shows that the Factory Method concern is spread over six components (CSC = 6). In addition, the Lack of Concern-based Cohesion metric shows that there are two concerns in both MetaObjEncapsule and MetaObjComposite classes. As one of these concerns is the Factory Method, this indicates that it is tangled with another concern in these components.

Analyzing the results of these two metrics, the designer could conclude that the Factory Method concern is crosscutting, because it is scattered over multiple components and tangled with another concern. However, this is a false warning.

In fact, the object-oriented abstractions provide adequate means of modularizing the Factory Method: the main purpose of the classes which implement this pattern is to realize it. In this case, the false warning was a result of methods related to the Observer pattern in the classes of the Factory Method pattern. Therefore, this is not a problem related to the Factory Method pattern design at all. Our studies indicate that shortcomings of this category are ameliorated with the support of concern-sensitive heuristic rules. For instance, the Factory Method example just discussed does not produce a false warning when the heuristic rules are applied (Table 25).

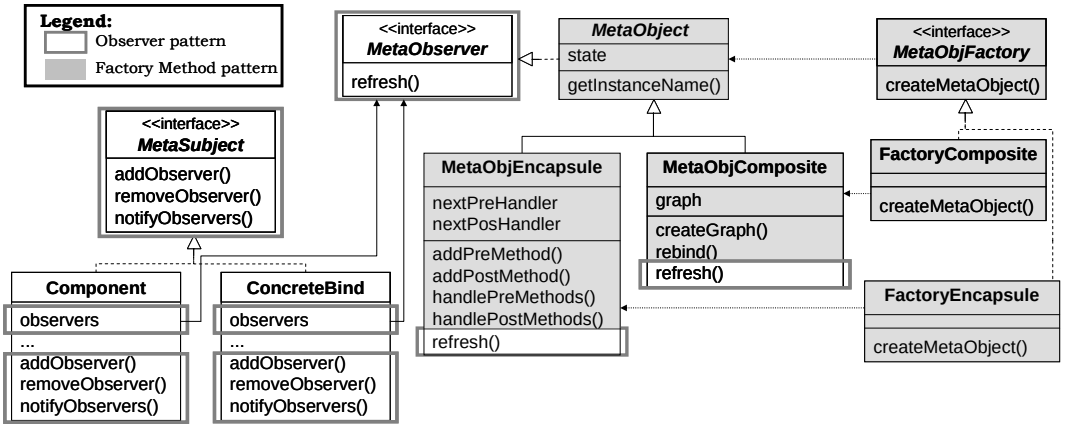


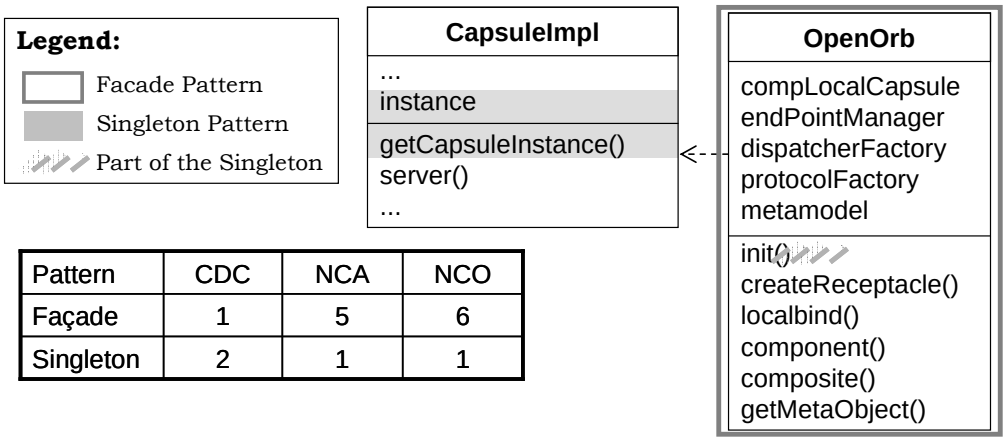
Figure 61: Observer and Factory Method patterns used in the design of an OpenOrb-compliant middleware system (Cacho et al., 2006a, 2006b, 2007)

Hiding concern-sensitive flaws

Sometimes design flaws are omitted in the measurement outcomes just because the metrics are not able to reveal an existing modularity problem. We illustrate this limitation in the light of a partial class diagram presented in Figure 62. This figure shows shadowed elements to highlight that they implement the Singleton and Façade patterns. Besides, it also presents the results of three concern-driven metrics for these patterns: Concern Diffusion over Components (CDC) (Section 5.3.1), Number of Concern Attributes (5.3.5) and Number of Concern Operations (NCO) (Section 5.3.5). Note that the values for NCA and NCO in Figure 62 represent the sum of the values for all the classes with the concerns.

Although Singleton has an average metric value lower than Façade, the former presents a crosscutting nature and the latter does not. Therefore, the measurement results in this example are not indicative to warn the developer

about the crosscutting phenomena relative to Singleton (Cacho et al, 2006a; Garcia et al., 2006b). Again, the application of concern-driven heuristics (Table 25) overcomes this measurement shortcoming and correctly classifies the Singleton pattern as black sheep (Section 5.4.3), which is a specialized category of crosscutting concern.



Pattern	CDC	NCA	NCO
Façade	1	5	6
Singleton	2	1	1

Figure 62: Concern-driven metrics for Façade and Singleton

Controversial outcomes from concern measures. A problem of this category occurs if the results of different metrics do not converge to the same outcome, hindering the designer's interpretation. We have identified some occurrences of this problem in our studies. For instance, applying the concern-driven metrics to adaptation concern in the Portalware system, the value for the Concern Diffusion over Components metric is 3 (indicating low scattering) while other concern metrics present high values (e.g. Number of Concern Attributes = 10 and Number of Concern Operations = 22). Hence, the concern metric results are contradictory in the sense that it is hard to conclude whether adaptation is or not a crosscutting concern. The concern heuristic rules address this category of shortcomings in a number of cases. For instance, although the adaptation concern has contradictory results for the concern metrics, the application of concern-driven heuristics (Table 25) has successful identified it as an octopus (Section 5.4.2), which also means that it is a crosscutting concern.

8.2. Detailed Design Metrics Study

The goal of this study is to evaluate the effectiveness of our concern-driven detailed design metrics to detect design flaws when they are applied by

independent subjects. In particular, the purpose of this study is to compare concern-driven and conventional measurement in order to learn which technique is the most effective to support the detection of two specific bad smells: shotgun surgery and divergent change (Fowler, 1999). These two bad smells are typically associated with crosscutting concerns (Monteiro & Fernandes, 2005). The first bad smell was the same used in our heuristics assessment study (Section 8.1).

Divergent Change occurs when one class is commonly changed in different ways for different reasons. Fowler (1999) say about Divergent Change: "If you look at a class and say, 'Well, I will have to change these three methods every time I get a new database; I have to change these four methods every time there is a new financial instrument', you likely have a situation in which two classes are better than one." On the other hand, *Shotgun Surgery* as explained in previous sections is encountered when every time you make a change to a class, you have to make a lot of little changes to a lot of different classes.

8.2.1. Study Format and Procedures

This study involved eight master students attending an Aspect-Oriented Software Development course at Lancaster University. The students were grouped in pairs. Each pair worked with different metrics in order to identify classes that were suspect of having one of the two bad smells in the object-oriented design of the Health Watcher system (Soares et al., 2002). Two groups worked only with conventional metrics, one group only with concern-driven metrics, and the forth group with both conventional and concern-driven metrics (hereafter referred as hybrid metrics group).

We estimated each student's relative ability from our previous knowledge of them and background questionnaire they answered regarding the scope of their previous experience, particularly with regards to object-oriented programming (in Java) and design, class diagram, and software metrics. Then the pairs were formed to balance abilities. All the students had previous experience with object-orientation and class diagrams in academy. Two of them had experience with these techniques in the industry context. None of them had previous experience with software metrics.

The conventional metrics group worked with the following metrics: Coupling Between Object Classes (CBO), Lack of Cohesion in Methods (LCOM), Weighted Methods per Class (WMC), described in (Section 2.4), as well as the metrics Number of Attributes (NOA) and Number of Methods (NOO), which merely count the number of attributes and methods of each class, respectively. The concern-driven metrics group worked with the metrics Concern Diffusion over Classes (CDC), Concern Diffusion over Operations (CDO) and Lack of Concern-based Cohesion (LCC) (Sections 5.3.1 and 5.3.3). The hybrid metrics group worked with all these metrics. The study was preceded by a training session in order to allow the participants to familiarize themselves with these metrics and the target bad smells.

At the beginning of the study execution, the participants were then given a document containing: (i) a partial view of the Health Watcher object-oriented design (class diagram), (ii) an introduction Health Watcher functionalities and non-functional requirements, (iii) a brief explanation of the design, and (iv) a brief description of the concerns involved in the Health Watcher design, namely graphical user interface (GUI), business, concurrency, distribution, exception handling, and persistence. This document also described steps and guidelines the students should follow to conduct the study, the questions they should answer and information they should register. Appendix B presents the mentioned document.

In addition, we provided the students with the results of the metrics application. Each group only had access to the results referent to the metrics they were assigned to work with. Appendix B presents tables of the results of all used metrics. Each group was asked to perform the following steps:

- Read the description of the Health Watcher design;
- Based on the metrics results, identify the classes with the highest probability of having the bad smell Divergent Change; and
- Based on the metrics results, identify the classes with the highest probability of having the bad smell Shotgun Surgery.

The time spent on each of these tasks was registered by each group. In order to identify the classes with bad smells, we asked them to reason about the metrics and identify which of them (one, some, or all) are relevant indicators based on the bad smell description. Also, we asked each group to explain which metrics they used for detecting the bad smell and which ones were not useful at all.

8.2.2. Hypothesis and Results

The hypothesis we wanted to test in this study was that the hybrid metrics suite is the most effective to support detection of design bad smells. The basic intuition behind this hypothesis is that the more metrics we can use, the more equipped you are to identify the design flaws. In order to test this hypothesis, we compared the actual instances of the bad smells with the classes identified by each group as the strongest candidates of having the bad smells. Before the study was executed, we played the oracle role to determine which classes were affected by the bad smells, based on our extensive knowledge of the HealthWatcher system and its releases. In order to do that, we checked out the source code of HealthWatcher and observed comments and changes made by real developers while refactoring the Java to AspectJ version of this system. We identified twelve classes affected by Divergent Change and eight by Shotgun Surgery.

Table 29 shows for each group and each bad smell: (i) the time spent on the identification of the bad smell, and (ii) the number and percentage of hits, and the number and percentage of false positives. A hit occurs when the group identified a class which was in our list of classes affected by the bad smell. A false positive occurs when the group identified a class which was not in our list. The percentage of hits is calculated dividing the number of hits by the number of classes in our list: 12 for Divergent Change, and 8 for Shotgun Surgery. The percentage of false positives is calculated dividing the number of false positives by the total number of classes identified by the group.

	Conventional Metrics	**Conventional Metrics**	**Concern-driven Metrics**	**Hybrid Metrics**
Divergent Change Identification				
Time (minutes)	9	10	21	31
Hits	2 (17%)	2 (17%)	12 (100%)	9 (75%)
False positives	1 (33%)	2 (50%)	7 (36%)	0 (0%)
Shotgun Surgery Identification				
Time (minutes)	6	10	13	35
Hits	1 (12%)	1 (12%)	6 (75%)	1 (12%)
False positives	3 (75%)	2 (33%)	11 (64%)	3 (75%)

Table 29: Results - identification of Divergent Change and Shotgun Surgery

As far as the identification of classes affected by Divergent Change is concerned, we can observe in Table 29 that the two groups with conventional metrics performed significantly worse than the others. Both only obtained two hits (17%). Besides, 33% and 50% of the classes they indicated as suspects were false positives. Both conventional metrics groups reported that the most useful metric for identifying Divergent Change was Lack of Cohesion in Methods (LCOM) (Section 2.4). As a consequence, this result is in line with studies in the software metrics literature which states that LCOM is a problematic metric sometimes because it leads to a number of false positives and false negatives (Fenton & Pfleeger, 1999). This metric presented high values for classes with no design problems.

Still regarding the Divergent Change bad smell, the group working with concern-driven metrics had 100% of hits, however 36% of false positives. In fact, we did not limit the number of classes to be listed by the groups. So the concern-driven metrics group indicated a high number of classes as having Divergent Change (19 classes). Nevertheless, as shown in the study guideline form (Appendix B), we asked the students to rank their list of classes with the ones with highest probability of having the bad smell coming first. The twelve first classes in the list of the concern-driven group were exactly the same of our list. This group reported that they used the Lack of Concern-based Cohesion (LCC) metric to identify Divergent Change. The group working with the hybrid suite of metrics also performed well. This group had 75% of hits and none false positive. Lack of Concern-based Cohesion (LCC) and Lack of Cohesion in Methods (LCOM) were the metrics considered useful by this group. However, in this case, the presence of LCC was efficient to minimize the limitations of LCOM.

We can also observe from Table 29 that the group working with conventional metrics did not perform well regarding the identification of the Shotgun Surgery bad smell either. They had just 12% of hits. Besides, one of the groups had 75% of false positives and the other 33%. Differently from the analysis of Divergent Change, the performance of the hybrid metrics group was not good for Shotgun Surgery identification: 12% of hits and 75% of false positives. Apparently the reason for the low performance of these three groups is the same: some conventional metrics (in general, size metrics) might have introduced “noise” in the design assessment. Metrics such as Number of

Operations (NOO), Number of Attributes (NOA) and Weighted Methods per Class (WMC) presented high values for classes not affected by the Shotgun Surgery bad smell. The group working with concern-driven metrics had again a high number of hits (75%), however an expressive number of false positives (65%). The reason for the high number of false positives was again the high number of listed classes. But again the correctly indentified classes were listed as having the highest probability of having the bad smell. This group reported that they used the Concern Diffusion over Components (CDC) metric to identify Shotgun Surgery.

The study results partially contradict our hypothesis that the hybrid metrics suite would be the most effective to support detection of design bad smells. This occurred mainly because of the results associated with the Shotgun Surgery bad smell. In spite of the high number of false positives, the concern-driven metrics suite was the most effective for identifying the assessed bad smells. Apparently the high number of metrics hindered the analysis made by the hybrid metrics group. As we can see in Table 29, the group working with these metrics took the longest time to finish their tasks. This might be because they spent too much time analyzing non-useful measures for the bad smells under assessment.

8.3. Study Constraints

This section discusses some constraints related to the studies regarding detailed design metrics and heuristic rules. Similarly to the studies about architectural metrics, the conclusions obtained here are restricted to the assessed software systems and analyzed concerns. Results regarding advantages and drawbacks in using concern-driven metrics and heuristic rules obtained in these studies should not be directly generalized to other contexts. However, these studies allowed us to make useful evaluations on whether the use of concern-driven metrics and heuristic rules for assessing design modularity would be worth studying further.

The studies in this section involve concern-driven metrics, thus they suffer from limitations related to the fact that the process of assigning concerns to design elements directly impacts on the measurement results. In order to make this

process more systematic, we took here the same measures we took in the studies with architectural metrics (Section 7.5): (i) “pair mapping”, where the assignment of concerns to design elements was done by two people assisting each other, (ii) consultation of the actual system developers, when possible, and (iii) we followed the “remove concern – remove design element” guideline (Section 7.1).

Other issue that limits the conclusions from the studies about detailed design metrics and heuristic rules is the fact that we played the role of the oracle while deciding which classes and aspects were affected by the design flaws. This could have biased the results mainly because the assessed concern-driven metrics were proposed by us. In order to minimize this issue, we consulted and observed comments of the real developers of the assessed systems as well as changes made by them to improve the design, especially during aspect-oriented refactoring.

The evaluation studies showed that concern-driven metrics and heuristic rules are promising means of modularity flaws detection, and are usable in practice. However, it is clear that the number of systems used in the study is by no means statistically relevant. Besides, we have not used rigorous statistical methods in the empirical evaluation. Nonetheless, we are considering the sample representative of the population due to the heterogeneity of systems and concerns involved in this study. We have recently replicated this study with four other groups of undergraduate and master students (total of 30 students) at the Lancaster University and the results were similar with the ones presented here. We also included other bad smells (e.g. God Class (Riel, 1996)) in these new studies.