

Evaluating the Efficacy of Concern-Driven Metrics: A Comparative Study

Claudio Sant’Anna

Computer Science Department
Federal University of Bahia (UFBA), Brazil
santanna@dcc.ufba.br

Alessandro Garcia

Computing Department
Lancaster University, UK
garciaa@comp.lancs.ac.uk

Carlos J. P. Lucena

Computer Science Department
PUC-Rio, Brazil
lucena@inf.puc-rio.br

Abstract

The inadequate modularization of driving software concerns degrades design modularity. Software metrics are traditionally key mechanisms for detecting modularity-related design flaws. Conventional design metrics are based on the module abstraction. With the emergence of new programming techniques targeting superior concern modularity, a number of concern-sensitive metrics have been proposed for assessing these techniques. These metrics are based on the concern abstraction. However, there is not much empirical evidence on the efficacy of concern-driven metrics. It is also unknown if they promote superior modularity assessment or a valuable complement to conventional metrics. We present an empirical study that compares the efficacy of specific sets of concern-driven and conventional metrics to detect two of Fowler’s bad smells. Our findings showed that the concern-driven metrics performed better than the conventional ones and are thus promising mechanisms for supporting modularity assessment.

Keywords software design; modularity; metrics, empirical software engineering

1. Introduction

The achievement of modular designs is far from being trivial as a multitude of widely-scoped concerns need to be simultaneously modularized. A concern is any important property or area of interest of a system that we want to treat in a modular way [9]. Business rules, distribution, persistence, security and caching are examples of concerns found in many software systems. Recent studies have shown that inadequate concern modularizations may lead to multiple design flaws [3, 11, 12, 15, 16]. For example, they can promote violations of important design principles, such as low coupling and narrow interfaces [3, 12, 15, 16].

The systematic assessment of modularity plays a pivotal role in the realm of software design. Software metrics are traditionally key mechanisms for assessing design modularity and identifying modularity-related design flaws [4, 17, 18]. Although typical modularity problems are related to

the inadequate modularization of concerns, most of the current design metrics do not explicitly consider concern as a measurement abstraction. To date, design assessment has been mostly rooted at module-based metrics [4, 10]. The object-oriented metrics community has consistently used notions of class coupling, cohesion and interface size to derive measures of modularity [1, 2, 4, 10].

The recognition that concern-based assessment is important through software design activities is not new. In fact, with the emergence of new design decomposition approaches, such as aspect-oriented software development (AOSD) and feature-oriented programming [13], there is a growing body of relevant work focusing on concern analysis techniques [20]. In particular, a number of suites of metrics that can be qualified as “concern-driven” have been recently proposed [5-7, 21, 22, 24]. In addition, several empirical studies involving concern-driven metrics have been carried out [3, 6, 11, 12, 15, 16]. Most of these studies focus on either applying concern metrics to assess aspect-oriented designs or validating them as predictors of defects. However, there is not much knowledge on the efficacy of concern-driven metrics for identifying design flaws in comparison with conventional module-based metrics.

This paper presents a first exploratory study that compares the efficacy of concern-driven and conventional module-based metrics (herein called conventional metrics) on the identification of design anomalies. Sets of concern-driven and conventional metrics were applied to the object-oriented source code of a Web-based information system. The values of the different sets of metrics were separately analyzed by distinct groups of design reviewers in order to identify specific design flaws. The goal was to compare the number of classes correctly identified as suspects of exhibiting a design flaw by using the different kinds of metrics. The study focused on two design flaws, namely the shotgun surgery and divergent change bad smells [14]. The notion of bad smells was proposed in Fowler’s book [14] to diagnose symptoms that may be indicative of something wrong in the design modularity. Our findings, although preliminary, suggest that concern-driven metrics are promising to

enhance modularity assessment. We also aim to learn some lessons from this study to support its future replication.

The remainder of this paper is organized as follows. Section 2 introduces the concept of concern-driven metrics. Section 3 describes the study procedures and configuration. Section 4 presents and discusses the results. The study constraints are addressed in Section 5. Section 6 describes related work. Section 7 concludes.

2. Concern-Driven Metrics

Concern-driven metrics are defined to capture modularity properties associated with the realization of concerns in software artifacts. This kind of metric allows the identification of specific design flaws or design degeneration caused by the poor modularization of concerns. Most of the existing concern-driven metrics [5, 6, 21, 22, 24] focus on quantifying the degree of concern scattering and tangling. Scattering is the degree to which a concern is spread over the design elements. Tangling represents the degree to which a concern is mingled with other concerns [9].

Concern-driven measurement approaches are based on a concern-to-design (or concern-to-code) mapping. This means that we have two domains related to each other through a mapping relationship. The source domain is a set of concerns and the target domain is a set of design elements, as illustrated in Figure 1 (inspired by a similar figure presented in [7]). The mapping consists of assigning a concern to the corresponding design elements that realize it. Therefore, before computing concern-driven metrics, it is necessary to identify the design elements responsible for implementing each concern in the system.

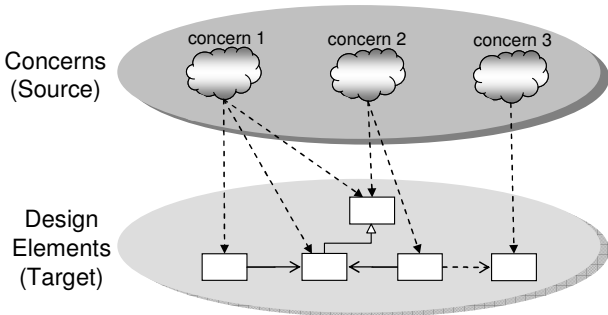


Figure 1. Mapping between concerns and design elements

In our empirical study, we evaluated the efficacy of three concern-driven metrics proposed by Sant’Anna and colleagues [21, 22]: Concern Diffusion over Components (CDC), Concern Diffusion over Operations (CDO) and Lack of Concern-based Cohesion (LCC). These metrics have been used in several studies with the goal of comparing object-oriented and aspect-oriented designs [3, 11, 12, 15, 16]. CDC and CDO measure the scattering of a given concern through the system components (classes and as-

pects) and operations, respectively. CDC counts the number of components that contribute to the realization of a given concern. CDO counts the number of operations that contribute to the realization of a given concern. The assumption behind these metrics is that a concern spread over a high number of design elements is detrimental to modularity. The understanding of a highly-spread concern demands the analysis of a large part of the design. In addition, a change related to that concern may affect a large number of design elements realizing the target concern.

LCC measures the cohesion of a given component in terms of the quantity of concerns addressed by it. Thus, it counts the number of concerns mapped to each component. The reasoning behind this metric is that a component that encompasses a large number of concerns is unstable. This is because it may suffer from modifications derived from change requests related to any of the concerns implemented by it.

To express the metrics unambiguously and facilitate the replication of our empirical study, we present the formal definition of the concern-driven metrics based on set theory. First, we present the terminology used on the formal definitions. Let S be a system, the classes and aspects of 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 $O(c)$. The set of all attributes and all operations in system S are represented as $A(S)$ and $O(S)$, respectively. In classes, operations are methods, and, in aspects, operations represent methods and pieces of advice. For each $c \in C(S)$, the set of concerns assigned to c is denoted as $Con(c)$. Let $o \in O(c)$ be an operation of c , the set of concerns assigned to o is denoted as $Con(o)$. Let $a \in A(c)$ be an attribute of c , the set of concerns assigned to a is denoted as $Con(a)$. For each concern con realized on the design of system S , the set of components, operations and attributes to which con is assigned is, respectively, denoted as:

$$\begin{aligned} C(con) &= \{c \mid c \in C(S) \wedge con \in Con(c)\}, \\ O(con) &= \{o \mid o \in O(S) \wedge con \in Con(o)\}, \text{ and} \\ A(con) &= \{a \mid a \in A(S) \wedge con \in Con(a)\}. \end{aligned}$$

The CDC, CDO and LCC metrics can now be defined as follows:

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

$$CDO(con) = |O(con)|, \text{ and}$$

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

3. Study Settings

The goal of this study is to evaluate the efficacy of a set of concern-driven metrics to detect design flaws. In particular, the purpose of this study is to compare concern-driven and conventional metrics in order to learn which technique is the most effective to support the detection of two specific bad smells: *Shotgun Surgery* and *Divergent Change* [14].

3.1 Bad smells

The Shotgun Surgery and Divergent Change bad smells have been chosen because they are traditionally identified by the use of conventional metrics, mainly coupling and cohesion metrics [17, 18]. However, these bad smells are also believed to be a symptom of design flaws caused by poor modularization of concerns [19]. According to Fowler [14], 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.

The Divergent Change bad smell occurs when one class is commonly changed in different ways for different reasons. When explaining the nature of divergent changes, Fowler [14] states: “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.”

3.2 Target System

Our study involved the object-oriented design of a Web-based information system called Health Watcher [23]. This system supports the registration and management of complaints to the public health system. The main concerns involved in the Health Watcher design are: graphical user interface (GUI), business rules, concurrency, distribution, exception handling, and persistence.

This system was selected because it met a number of relevant criteria for our intended evaluation. First, it is a real system with an existing Java implementation. The first Health Watcher release was deployed in 2001 by the Public Health System in Recife, Brazil [23]. Since then, a number of incremental and perfective changes have been addressed in later Health Watcher releases.

Second, this system has served as a kind of benchmark for the assessment of contemporary modularization techniques, such as AOSD [12, 16, 23]. Third, the modularization of the concerns in the Health Watcher design has already been extensively studied. Fourth and foremost, the set of modularity-related design flaws for all the Health Watcher modules is well-known and has been investigated through the last 7 years by different developers and re-

searchers. This means that we could rely on a reliable “oracle” when judging the metrics efficacy.

3.3 Subjects and Statistical Relevance

This study involved eight master students attending an Aspect-Oriented Software Development course at Lancaster University. It is important to highlight that we knew from the experiment design outset that the population was not representative and it would not allow us to gather statistically-relevant data. However, this study was a good opportunity to investigate if concern metrics are candidates to improve the state-of-the-art of design modularity assessment and, therefore, would call for additional research. It also allowed us and others to derive some lessons to replicate this study in the future.

The students were grouped in pairs. Each pair worked with different metrics in order to identify classes that were suspected of having one of the two bad smells in the object-oriented design of the Health Watcher system [23]. Two groups worked only with conventional metrics, one group only with concern-driven metrics, and the fourth 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 a background questionnaire they answered regarding the scope of their previous experience, particularly with regards to object-oriented programming and design, class diagrams, and software metrics. Then the pairs were formed to balance abilities. All the students had previous experience with object-orientation and class diagrams in academic contexts. Half of the students had 2 or more years of experience with professional software development. Two of them had experience with these techniques in the industry context. None of them had previous experience with software metrics.

3.4 Assessed Metrics

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), described in Section 2. The conventional metrics group relied on the following metrics: Coupling Between Object Classes (CBO), Lack of Cohesion in Methods (LCOM), Weighted Methods per Class (WMC). The original definition of these metrics can be found in [4], and the formal definition of CBO and LCOM can be found in [1] and [2], respectively. The metrics Number of Attributes (NOA) and Number of Operations (NOO) were also used by the conventional metrics group. These metrics merely count the number of attributes and methods of each class. The hybrid metrics group worked with all eight metrics.

3.5 Activities

The study was preceded by a training session to allow the participants to familiarize themselves with the evaluated 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 of Health Watcher functional 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. The document also described steps and guidelines the students should follow to conduct the study, the questions they should answer and information they should register.

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. Each group was asked to perform the following steps: (i) read the description of the Health Watcher design, (ii) based on the metrics results, identify the classes with the highest probability of having the bad smell Divergent Change, and (iii) 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. To identify the classes with bad smells, we asked them to reason about the metrics and identify which of them (one, some, or all) were 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.

3.6 Hypothesis

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 intuition behind this hypothesis is that the hybrid metrics group is better equipped to identify the design flaws because both sets of conventional and concern-driven metrics: (i) are intuitively suited for detecting the selected bad smells, and (ii) may be used in a complementary way. 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 made a systematic analysis of Health Watcher artifacts to determine which classes were affected by the bad smells. First, we have used our own extensive knowledge of the system design and its releases. Second, we also checked out the source code of Health Watcher and observed comments and changes made by real developers while both refactoring the Java code and reengineering it with AspectJ [16, 23]. We identified twelve classes affected by Divergent Change and eight by Shotgun Surgery.

4. Results and Discussion

Table 1 shows for each group and each bad smell: (i) the time spent on the identification of the bad smell, (ii) the number and percentage of hits, and (iii) the number and percentage of false positives. A hit occurs when the group identified a class which was in the previously-generated list of classes affected by the bad smell (Section 3.5). A false positive occurs when the group identified a class which was not in the 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.

As far as the identification of classes affected by Divergent Change is concerned, we can observe in Table 1 that the two groups with conventional metrics performed significantly worse than the others. Both obtained only 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). In fact, this metric presented high values for classes with no design anomaly. This metrics computes cohesion based on pairs of methods that access attributes in common. Because of this it erroneously classified classes with a high number of getters and setters methods as low cohesive, for instance.

Still regarding the Divergent Change bad smell, the group working with concern-driven metrics had 100% of hits, however 36% of false positives. This group reported that they used the Lack of Concern-based Cohesion (LCC) metric to identify Divergent Change. 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). This occurred because they listed all classes with $LCC \geq 2$. However, a class addressing two concerns, for instance, does not necessarily represent a Divergent Change. Nevertheless, in the study guidelines, 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 as the previously generated list. The group working with the hybrid suite of metrics also performed well. This group had 75% of hits and no 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 minimized the previously discussed limitations of LCOM. Note that we fairly provided cohesion metrics to every group.

We can also observe from Table 1 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, interface 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 a large 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 identified 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 1, 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.

5. Study Constraints

This section discusses some constraints and imperfections discovered in the design and execution of this empirical study. The conclusions obtained here are restricted to the involved metrics, bad smells and the target software system. As discussed in Section 3.3, results regarding advantages and drawbacks in using concern-driven metrics

obtained in this study should not be directly generalized to other contexts. However, this study allowed us to make useful evaluations on whether the use of concern-driven metrics for assessing design modularity would be worth studying further.

This study involves concern-driven metrics, thus they suffer from limitations related to the fact that the upfront process of assigning concerns to design elements directly impacts on the measurement results. Concern mappings and metrics collection were performed by the experiment designers (and not by the students). It was not our intention to assess the time spent on concern mappings, which it is matter to be addressed in future experiments. To make this process more systematic, we followed a number of procedures: (i) “pair mapping”, where the mapping of concerns to design elements was done by two people assisting each other, (ii) consultation of the actual system developers whenever it was possible, and (iii) we followed a guideline that states that a concern should be assigned to a design element if the complete removal of the concern requires the removal or modification of the element [7].

Other issue that limits our findings is the fact that we played a crucial role in the definition of the oracle list, i.e. while deciding which classes were affected by the bad smells. This could have biased the results mainly because the evaluated concern-driven metrics were proposed by the authors of the experiment in previous works [21, 22]. To minimize this issue, this is why we have consulted and observed comments of the real developers of Health Watcher as well as changes made by them to improve the design.

6. Related Work

To the best of our knowledge, no other empirical study that explicitly compares concern-driven and conventional metrics has been undertaken yet. Several studies involving the two kinds of metrics have been carried out [3, 6, 11, 12, 15, 16]. However, these studies use concern-driven and conventional metrics in a complementary way to assess the modularity of a number of systems. In fact, most of them are dedicated to compare the modularity of aspect-oriented and object-oriented designs and implementations.

More recently, Eaddy et al [6] have carried out three experiments involving concern-driven metrics, including two of the metrics used in our study, namely Concern Diffusion

	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 1. Results: identification of Divergent Change and Shotgun Surgery bad smells

over Components (CDC) and Concern Diffusion over Components (CDO). Their experiment aimed at testing the hypothesis that the more scattered a concern's implementation is, the more likely it is to have defects. They found a moderate to strong correlation between the evaluated metrics and defects for all three experiments, which suggested that scattering may cause or contribute to defects. Nevertheless, their experiments do not focus on the comparison between concern-driven and conventional metrics. In fact, their work only encompasses concern-driven metrics.

7. Final Remarks and Ongoing Work

The emergence of new modularization techniques that promise superior separation of concerns has brought the attention of software engineering researchers back to the importance of concern-sensitive assessment of software modularity. Hence, a number of concern-driven metrics have been defined. In addition, several empirical studies involving the use of concern-driven metrics have been undertaken. Most of these studies focus on assessing the benefits and drawbacks of emerging modularization techniques, such as aspect-oriented software development. However, there is not much empirical evidence on the efficacy of concern-driven measurement in order to assess design modularity.

This work represents a first stepping stone towards the evaluation of how concern-driven metrics enhance the process of detection of modularity-related design flaws. We presented an empirical study that compared the efficacy of concern-driven and conventional metrics. Our findings showed that concern-driven metrics are promising means for supporting the detection of modularity flaws. However, it was a preliminary study and it is clear that the number of involved subjects is by no means statistically relevant. Therefore, we have recently replicated this study with more undergraduate and master students (total of 30 students). We also included other bad smells in these new studies. We are still working on the analysis of the results, but at a first glance they seem similar to the ones presented here.

Acknowledgements

This work is supported in part by the European Commission grant IST-33710: Aspect-Oriented, Model-Driven Product Line Engineering (AMPLE).

References

- [1] Briand, L. et al. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1), pp. 91-121, 1999.
- [2] Briand, L. et al. A Unified Framework for Cohesion Measurement in Object-Oriented Systems, *Empirical Software Engineering - An International Journal*, 3(1), pp. 65-117, 1998.
- [3] Cacho, N. et al. Composing Design Patterns: A Scalability Study of Aspect-Oriented Programming. *Proc. of AOSD*, Bonn, Germany, 2006.
- [4] Chidamber, S., and Kemerer, C. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 1994, 476-493.
- [5] Ducasse, S. et al. Distribution Map. *Proc. of ICSM 2006*, Philadelphia, USA, 203-212.
- [6] Eaddy, M. et al. Do Crosscutting Concerns Cause Defects? *IEEE Transactions on Software Engineering*. (to appear), 2008, available at <http://www.columbia.edu/~me133/>.
- [7] Eaddy, M. et al. Identifying, Assigning, and Quantifying Crosscutting Concerns, *Proc. of ACoM'2007*, in conjunction with ICSE'07, 2007.
- [8] Workshop on AO Requirements Eng. and Arch. Design (Early Aspects), in conjunction with ICSE'2007. <http://www.early-aspects.net/>, 2007.
- [9] Elrad, T.; Filman, R.; Bader, A. Aspect-Oriented Programming, *Communication of the ACM*, 44 (10), pp. 29-32, 2001.
- [10] Fenton, N.; Pfleeger, S. *Software Metrics: A Rigorous and Practical Approach*, 2.ed. London: PWS, 1997.
- [11] Figueiredo, E. et al. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. *Proceedings of the 30th ICSE'08*, Germany, pp. 10-18, 2008.
- [12] Filho, F. et al. Exceptions and Aspects: The Devil is in the Details. *Proc. of Int'l Symposium on Foundations of Soft. Engineering (FSE)*, 2006.
- [13] Filman, R. et al. *Aspect-Oriented Software Development*. Addison-Wesley, 2005.
- [14] Fowler, M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading, MA, USA, 1999.
- [15] Garcia, A. et al. Modularizing Design Patterns with Aspects: A Quantitative Study. *Trans. on AOSD*, 1, 2006, 36-74.
- [16] Greenwood, P. et al. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. *Proc. of ECOOP'07*, Berlin, Germany, 2007.
- [17] Lanza, M., and Marinescu, R. *Object-Oriented Metrics in Practice*. Springer, 2006.
- [18] Marinescu, R. Detection Strategies: Metrics-Based Rules for Detecting Design Flaws. *Proc. of ICSM*, 2004, 350 - 359.
- [19] Monteiro, M., and Fernandes, J. Towards a Catalog of Aspect-Oriented Refactorings. *Proc. of the AOSD*, Chicago, 2005, 111-122.
- [20] Robillard, M; Murphy, G. Representing Concerns in Source Code. *ACM Transactions on Software Engineering and Methodology*, 16(1), Article 3, February 2007.
- [21] Sant'anna, C. et al. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework, *Proc. of the Brazilian Symposium on Soft. Engineering*, pp. 19-34, Brazil, 2003.
- [22] Sant'anna, C. et al. On the Modularity of Software Architectures: A Concern-Driven Measurement Framework, *Proc. of the 1st European Conference on Software Architecture*, Spain, 2007.
- [23] Soares, S. et al. Implementing Distribution and Persistence Aspects with AspectJ. *Proc. of OOPSLA'02*, pp. 174-190, 2002.
- [24] Wong, W.; Gokhale, S.; And Horgan, J. Quantifying the Closeness between Program Components and Features. *Journal of Systems and Software*, pp. 87-98, 2000.