# The Impact of Coupling on the Fault-Proneness of Aspect-Oriented Programs: An Empirical Study

Rachel Burrows*†, Fabiano C. Ferrari‡, Otávio A. L. Lemos§, Alessandro Garcia† and François Taïani*
*School of Computing and Communications – Lancaster University – UK
Email: r.burrows@comp.lancs.ac.uk, f.taiani@lancaster.ac.uk
†Informatics Department – Pontifical Catholic University of Rio de Janeiro – Brazil
Email: afgarcia@inf.puc-rio.br
‡Computer Systems Department – University of São Paulo – Brazil
Email: ferrari@icmc.usp.br
§Department of Science and Technology – Federal University of São Paulo – Brazil
Email: otavio.lemos@unifesp.br

***Abstract*—Coupling in software applications is often used as an indicator of external quality attributes such as *fault-proneness*. In fact, the correlation of coupling metrics and faults in object-oriented programs has been widely studied. However, there is very limited knowledge about which coupling properties in aspect-oriented programming (AOP) are effective indicators of faults in modules. Existing coupling metrics do not take into account the specificities of AOP mechanisms. As a result, these metrics are unlikely to provide optimal predictions of pivotal quality attributes such as fault-proneness. This impacts further by restraining the assessments of AOP empirical studies. To address these issues, this paper presents an empirical study to evaluate the impact of coupling sourced from AOP-specific mechanisms. We utilise a novel set of coupling metrics to predict fault occurrences in aspect-oriented programs. We also compare these new metrics against previously proposed metrics for AOP. More specifically, we analyse faults from several releases of three AspectJ applications and perform statistical analyses to reveal the effectiveness of these metrics when predicting faults. Our study shows that a particular set of fine-grained directed coupling metrics have the potential to help create better fault prediction models for AO programs.**

## I. Introduction

Effective approaches for fault-prone module detection in object-oriented (OO) programs are often based on *coupling* metrics [1, 2, 3], which quantify the degree of interdependency between modules [4]. However, the use of coupling measures to identify faulty modules in Aspect-Oriented Programming (AOP) [5] is still a daunting task. Even with the establishment of industry-strength AOP frameworks [6, 7], the detection of fault-prone modules has rarely been investigated in this context. The difficulty stems from the fact that core AOP mechanisms introduce new and intricate forms of inter-module dependencies. For instance, AOP supports modularisation of crosscutting code within *aspects* through method-like elements, called *advices*. Differently from methods, advices are implicitly invoked at specific points of the program execution (the *join points*), which are specified in *pointcut* expressions. AspectJ [8], an AOP language, also supports *intertype declarations* that allow to alter module structures, e.g. by introducing new members such as methods or fields.

Nowadays there is a better understanding of which coupling properties are good indicators of faults in OO programs. Therefore, prediction models using coupling metrics can be used to identify faulty modules early on. In fact, contemporary evidence suggests that faults are largely influenced by particularities of inter-module dependencies established by the underlying programming mechanisms [1, 2, 3, 9]. For example, industrial studies have pointed out that *export coupling* in method calls had stronger association with fault-proneness than *import coupling* in OO programs [1, 10].

We need to improve our understanding about the relationships of coupling properties in AOP and their probability of indicating faulty modules. The problem is that existing coupling metrics for AOP [11, 12] are direct extensions of the Chidamber and Kemerer [13] metrics for OO software. Given the particularities of AOP mechanisms, it is questionable if extensions of conventional coupling measures are effective fault-proneness indicators. Also, metrics for AOP [11, 12, 14, 15] are also criticised for being too coarse-grained and not taking into account subtle dimensions of class-aspect coupling [14, 16, 17]. They do not consider dependencies established by specific types of advice or intertype declarations. Moreover, influenced by previous studies in OO programming [1, 10], coupling metrics in AOP tend to only quantify export coupling. Even worse, there is a lack of empirical validation on the effectiveness of these metrics to indicate fault-prone modules.

In this context, this paper presents an empirical study that investigates the ability of conventional and novel coupling metrics (Section III) to identify faulty modules in AOP. The conventional metrics are extensions of CK metrics used in previous experimental AOP studies. For the purposes of our analysis, we have created 20 new coupling metrics that quantify specific coupling properties of AOP. These metrics were defined and classified based on Briand et al.'s coupling measurement framework [18] and its extension to AOP [19]. The objective of our fault-proneness analysis was mainly twofold: (i) evaluate the effectiveness of fine-grained coupling measures for specific AOP mechanisms, and (ii) understand if there is any difference on the performance of import and export coupling metrics as fault predictors.

We analysed four releases from three applications written in AspectJ, from which faults and metrics for each module were collected and documented (Section IV). To carry out a more precise evaluation (Section V), we used two extensively used statistical techniques: Spearman's rank correlation and logistic regression analysis. Our findings (Section VI) indicate that certain fine-grained metrics outperform coarse-grained ones because specific AOP mechanisms have clearly shown to be more fault-prone than others. For instance, import coupling metrics for after advice outperformed the majority of other fine-grained metrics. We also observed that, unlike in OO programming, import metrics had stronger association with fault-proneness than export metrics. This indicates that an aspect affecting many classes may be symptomatic of a high probability of latent faults. We also discuss our study limitations (Section VII), related work (Section VIII), and concluding remarks (Section IX).

## II. Background

Aspect-Oriented Programming(AOP) [5] aims to improve the modularity of crosscutting concerns. Such concerns include exception handling, concurrency and caching as they are often scattered and tangled across multiple modules in a software system. AOP aims to extract these concerns from the *base code* and modularise them into aspects.

In AOP, *advice* is a method-like construct that defines crosscutting behaviour. An advice runs at *join points* selected by a *pointcut*. In AO languages, there are five typical types of advice: (i) before: runs before the join point; (ii) around: runs in place of the join point; (iii) after returning: runs after the normal execution of the join point; (iv) after throwing: runs after the abnormal execution of the join point; and (v) after: runs after either the normal or the abnormal execution of the join point. While advices change the behaviour of modules they crosscut, the static structure of such modules are modified by AspectJ's instructions called *intertype declarations* (ITD) and other *declare*-like forms. ITDs allow the introduction of members (methods and attributes) into other types. *Declare Parents* is used to change the type hierarchy of a system. *Declare Soft*, on the other hand, specifies that an exception, if thrown at a join point, is converted to an unchecked exception.

A number of AOP-specific coupling metrics have already been proposed [11, 12, 14, 15] and successfully used in empirical studies of AOP [12, 20, 21]. However, they have been criticised [14, 16, 17] for not effectively capturing subtle coupling unique to AO programs that results from the use of the aforementioned constructs.

One critisism of existing AOP coupling metrics is that they are too coarse-grained. This is because they measure multiple sources of coupling at the same time creating a "high-level" measure. More specifically, they do not separate the coupling contributions of individual AOP mechanisms, even though the varied mechanisms might contribute differently to a module's overall fault-proneness [17, 14].

Metrics such as Coupling Between Modules (CBM) [11] demonstrates this. It has a *module* as its *unit of measurement* as well as its *granularity*. This can be understood by referring to a simplified definition – CBM is the number of *modules* possibly called or accessed from another *module*. Thus, CBM cannot distinguish between the individual internal mechanisms that cause coupling, neither to the varying coupling strength between modules. Note that, in this paper, we interpret coupling strength as the frequency of connections between two modules, despite different definitions that can be found elsewhere [18].

Secondly, existing coupling metrics for AOP tend to overlook the impact of different coupling directions (*locus of impact*). The influence of coupling direction is certainly not a new concept within the metrics community because there are important differences between import and export coupling metrics [4, 18]. For example, different faults can be quantified depending on the locus of impact of the metric; modules with high frequency of incoming coupling dependencies (import coupling) may have a higher probabilities of becoming less reusable [18] and also manifesting faults within themselves during evolution. However, modules with a high frequency of outgoing coupling dependencies (export coupling) may have a greater chance to cause surrounding dependent modules to become faulty [22]. Despite this, existing metrics for AOP tend to overlook the impact of different coupling directions i.e. most metrics only quantify export coupling [17].

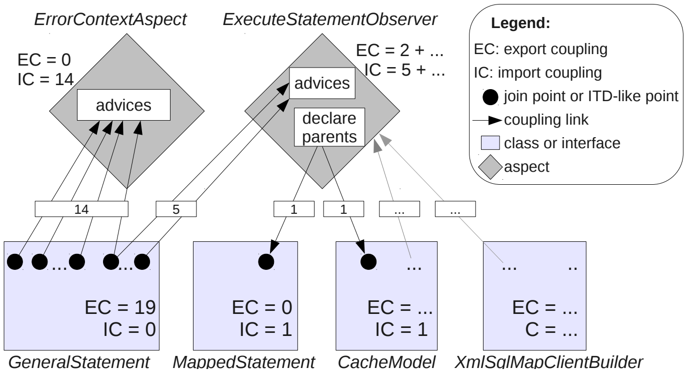

Fig. 1. Real example of import and export coupling between modules.

Figure 1 demonstrates the different impact that import and export coupling have in AspectJ programs. The `GeneralStatement` class is advised by two aspects; it has 19 export couplings as it has 19 join points that trigger advice in the aspects. However, it has an import coupling value of 0. We can also see in the `ErrorContextAspect` that import coupling is 14 as the advice inside the aspect is triggered 14 times from the base code; however export coupling is 0. Within the context of AOP, only utilising and developing export metrics in empirical studies of AOP is a dangerous trend that might be overlooking a wealth of important coupling information. Thus we evaluate the impact of both types of coupling to understand the behaviour of base-aspect dependencies in AOP.

Our previous work [23] compared the fault prediction ability of existing AOP metrics [11] with a novel, combined metric that takes into account the several coupling forms between aspects and the base code. Results showed that metrics that take into account fine-grained coupling connections created from aspect-base interactions outperformed the traditional metrics for predicting faults in AO systems. Such results motivate the metrics suites we introduce for our analysis in the next section.

## III. The Metrics

We propose 20 new metrics to measure coupling dimensions unique to AOP (Table II). To the best of our knowledge,

TABLE I
BASELINE METRICS

| Name | Description | Domain of Measure | Client Item | Locus of Impact |
|---|---|---|---|---|
| Depth of Inheritance Tree (DIT) | The length of the longest path from a given module to the class/aspect hierarchy root | class or aspect | class or aspect | n/a |
| Coupling on Advice Execution (CAE) | # of aspects containing advices possibly triggered by the execution of operations in a given module | class or aspect | aspect | export |
| Coupling between Modules (CBM) | # of modules or interfaces declaring methods or fields that are possibly called or accessed by a given module | class or aspect | class or aspect | export |

these metrics have not been suggested so far. They focus on eight AO-specific mechanisms: (i) the behavioural flows created by five different kinds of advice, and (ii) three types of structural changes that can be performed by aspects. For each mechanism, we consider two coupling directions: first, the coupling *induced* by an aspect on the rest of the code through this mechanism, yielding a set of *Aspect Coupling Metrics*; and, second, the coupling *received* by a module (class or aspect) through this mechanism from all aspects in the system, yielding a set of *Base Coupling Metrics*. We complement both groups of metrics with four summative metrics (see Table II), summarising either the behavioural (Base Behavioural Coupling and Aspect Behavioural Coupling) or structural coupling (Base Structural Coupling and Aspect Structural Coupling).

Finally, as a baseline to our study, we chose three commonly-used metrics that have been applied to AOP programs (Table I). In the following, we discuss all three groups of metrics (the baseline metrics in Section III-A, the new metrics in Section III-B), and provide a formal definition of the new metrics based on Briand et al.'s framework for coupling measures [18] (Table II and Section III-C).

### A. Baseline Metrics

The first two baseline metrics, Depth of Inheritance Tree (DIT) and Coupling Between Modules (CBM), are AO metrics adapted from traditional OO metrics and are frequently used in experimental studies of AOP languages [17]. They have presented a strong correlation with fault proneness in OO applications [1, 24], making them an interesting reference point against which to compare our set of new metrics. Coupling on Advice Execution (CAE) is also a popular metric applied in empirical studies on AOP. Unlike the first two, however, CAE only measures AO-specific coupling. Full definitions of these metrics are given in the original paper [11].

### B. New Metrics: Base and Aspect Coupling

The metrics we propose are designed to serve two goals: (i) assess the impact of individual AO mechanisms on fault proneness; and (ii) compare the respective impact of import and export coupling for each mechanism.

The first goal is served by considering specific metrics for each of the eight AOP mechanisms we targeted: five behavioural mechanisms covering different types of advices (before, after, after throwing, after returning and around); and three structural mechanisms (intertype declarations, declare parents statements, and declare soft statements). The first five behavioural mechanisms are widely found in AO languages, including AspectJ, which we used in our experimental study. The three structural mechanisms are less commonly found, and tend to be AspectJ-specific, but represent an important class of crosscutting structural changes. We also designed each metric to reflect coupling frequency, by counting every time a code is advised (possibly at the same joint point), rather than just summing up the number of modules involved in a coupling, as is often done for popular AO and OO coupling metrics.

The second goal is met by considering two coupling directions (import or export) for each of these mechanisms, yielding two groups of metrics, *Base Coupling*  and *Aspect Coupling*. Each metric in Base Coupling has a counterpart in Aspect Coupling that measures the opposite direction of coupling. As such, the domain of measure in Base Coupling is always a class or aspect whereas in Aspect Coupling it is always an aspect. Note that because of the semantic difference between inheritance-related metrics (measuring structural impact) and non-inheritance-related metrics (measuring behavioural impact), the locus or impact between structural and behavioural mechanisms is inverted in each group.

### C. Formal Description of the New Metrics

We classified the Base Coupling metrics and the Aspect Coupling metrics according to Briand et al.'s coupling framework [18]. Table II displays the metric name, description, and the 12 framework criteria. The values assigned to each metric are summarised in the bottom of Table II. Next we briefly discuss the most important points of this formalisation.

The *client* and *server item* (*Criteria 1* and *2*) indicate the direction of the coupling: from join points to advice for behavioural mechanisms, and from the structural declaration to the affected component for structural mechanisms. The coupling direction is indicated by the *locus of impact*: from the server to the client item for export coupling, and conversely for import coupling. For instance, Base Coupling via Before Advice measures the number of times a join point triggers advice in any of the aspects of the system. The direction of coupling is going out of the module being measured, and is therefore classified as *export*.

To better reflect coupling frequencies, as mentioned earlier, all metrics aggregate individual connections (ID) between client and server (*Criterion 6*) instead of counting the number of distinct items at the endpoints of the connection.

Regarding inheritance (*Criterion 11*), all metrics partially account for inherited members. More precisely, members of a module that are inherited and overridden contribute to the coupling metrics; however, members that are inherited but not overridden only contribute to coupling value of the implementing module. Note that this criteria does not apply to Base (and Aspect) Coupling via Declare Parents.

Finally *Criterion 12* classifies which members can contribute to coupling within a module. Base Coupling and Aspect Coupling metrics account for coupling between implemented members. The exception to this rule is Base and Aspect

TABLE II
BASE COUPLING AND ASPECT COUPLING METRICS SUITES.

## Base Coupling metrics suite

| Name | Description | 1. Client Item | 2. Server Item | 3. Phase of application | 4. Locus of Impact | 5. Domain of Measure | 6. How is Counted | 7. Server Stability | 8. Coupling Direction | 9. Inheritance-based | 10. Account for Polymorphism | 11. Inheriting owns inherited | 12. Element assignment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Base Couping via Before Advice** of module *m* | # of join point shadows in *m* advised via before advices | AD | JP | LLD, HLD | E | CA | ID | U | D | N | Y | P | Im |
| **Base Coupling via Around Advice** of module *m* | # of join point shadows in *m* advised via around advices | AD | JP | LLD, HLD | E | CA | ID | U | D | N | Y | P | Im |
| **Base Coupling via After Throwing Advice** of module *m* | # of join point shadows in *m* advised via after throwing advices | AD | JP | LLD, HLD | E | CA | ID | U | D | N | Y | P | Im |
| **Base Coupling via After Advice** of module *m* | # of join point shadows in *m* advised via after advices | AD | JP | LLD, HLD | E | CA | ID | U | D | N | Y | P | Im |
| **Base Coupling via After Returning Advice** of module *m* | # of join point shadows in *m* advised via after returning advices | AD | JP | LLD, HLD | E | CA | ID | U | D | N | Y | P | Im |
| **Base Coupling via Declare Parents** of module *m* | # of module hierarchy changes in *m* caused by declare parents statements | CA | DP | LLD, HLD | I | CA | ID | U | D | Y | n/a | n/a | Im |
| **Base Coupling via Intertype Declaration** of module *m* | # of module hierarchy changes in *m* caused by intertype declaration | CA | ITD | LLD, HLD | I | CA | ID | U | D | Y | Y | P | Im, Dec |
| **Base Coupling via Declare Soft** of module *m* | # of module hierarchy changes in *m* caused by declare soft statements | CA | DS | LLD, HLD | I | CA | ID | U | D | Y | Y | P | Im |
| **Base Behavioural Coupling (BBC)** of module *m* | The sum of Base Coupling via (i) Before Advice, (ii) Around Advice, (iii) After Throwing Advice, (iv) After Advice, and (v) After Returning Advice for *m* | AD | JP | LLD, HLD | E | CA | ID | U | D | N | Y | P | Im |
| **Base Structural Coupling (BSC)** of module *m* | The sum of Base Coupling via (i) Declare Parents, (ii) Intertype Declaration, and (ii) Declare Soft for *m* | CA | DP,<br>ITD,<br>DS | LLD, HLD | I | CA | ID | U | D | Y | Y | P | Im |

## Aspect Coupling metrics suite

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Aspect Coupling via Before Advice** of aspect *a* | # of join points shadows advised via a before advice from *a* | AD | JP | LLD, HLD | I | A | ID | U | D | N | Y | P | Im |
| **Aspect Coupling via Around Advice** of aspect *a* | # of join points shadows advised via an around advice from *a* | AD | JP | LLD, HLD | I | A | ID | U | D | N | Y | P | Im |
| **Aspect Coupling via After Throwing Advice** of aspect *a* | # of join point shadows advised via an after throwing advice from *a* | AD | JP | LLD, HLD | I | A | ID | U | D | N | Y | P | Im |
| **Aspect Coupling via After Advice** of aspect *a* | # of join points shadows advised via an after advice from *a* | AD | JP | LLD, HLD | I | A | ID | U | D | N | Y | P | Im |
| **Aspect Coupling via After Returning Advice** of aspect *a* | # of join points shadows advised via an after returning advice from *a* | AD | JP | LLD, HLD | I | A | ID | U | D | N | Y | P | Im |
| **Aspect Coupling via Declare Parents** of aspect *a* | # of module hierarchy changes caused by declare parents statements from *a* | CA | DP | LLD, HLD | E | A | ID | U | D | Y | n/a | n/a | Im |
| **Aspect Couping via Intertype Declaration** of aspect *a* | # of hierarchical changes caused by intertype declarations from *a* | CA | ITD | LLD, HLD | E | A | ID | U | D | Y | Y | P | Im, Dec |
| **Aspect Coupling via Declare Soft** of aspect *a* | # of module hierarchy changes caused by declare soft statements from *a* | CA | DS | LLD, HLD | E | A | ID | U | D | Y | Y | P | Im |
| **Aspect Behavioural Coupling (ABC)** of aspect *a* | The sum of Aspect Coupling via (i) Before Advice, (ii) Around Advice, (iii) After Throwing Advice, (iv) After Advice, and (v) After Returning Advice for *a* | AD | JP | LLD, HLD | I | A | ID | U | D | N | Y | P | Im |
| **Aspect Structural Coupling (ASC)** of aspect *a* | The sum of Aspect Coupling via (i) Declare Parents, (ii) Intertype Declaration, and (ii) Declare Soft for *a* | CA | DP,<br>ITD,<br>DS | LLD, HLD | E | A | ID | U | D | Y | Y | P | Im |

**Legend:**
1. Client Item: AD = advice; CA = class or aspect
2. Server Item: JP = join point; DP = declare parents statement;
ITD = intertype declaration; DS = declare soft statement
3. Phase of application: LLD = low-level design; HLD = high-level design
4. Locus of Impact: I = import; E = export
5. Domain of Measure: CA = class or aspect; A = aspect
6. How is Counted: ID = individual connections; DE = distinct endpoints
7. Server Stability: S = stable; U = unstable
8. Coupling direction: D = direct; I = indirect
9. Inheritance-base: Y = yes; N = no
10. Account for Polymorphism: Y = yes; N = no
11. Inheriting class owns inherited members: P = partially
12. Assigning items at connection endpoints:
Im = implemented attributes contribute to coupling;
Dec = declared attributes contribute to coupling

Coupling via Intertype Declarations as it is possible in AspectJ to insert a declared method or attribute into a module which contributes to this metric.

## IV. Study Setup

### A. Goal Statement and Research Hypotheses

Our goal is to investigate the impact of intricate coupling dependencies in AO programs upon software fault-proneness, as stated in Section I. This investigation develops in terms of two hypotheses H1 and H2, whose null (0) and alternative (1) definitions are as follows:

*a) H1 – coarse-grained versus fine-grained metrics:*

- H1-0: There is no difference between the effectiveness of coarse-grained and fine-grained coupling metrics when used as indicators of fault-proneness in AO programs.
- H1-1: Fine-grained coupling metrics are more effective than coarse-grained coupling metrics when used as indicators of fault-proneness in AO programs.

*b) H2 – import versus export coupling metrics:*

- H2-0: There is no difference between the effectiveness of import and export coupling metrics when used as indicators of fault-proneness in AO programs.
- H2-1: There is a difference between the effectiveness of import and export coupling metrics as indicators of fault-proneness in AO programs.

### B. The Target Systems

To achieve our goals, we evaluated three AOP systems from different application domains. The first system is *iBATIS* [25], a Java-based open source framework for object-relational data mapping. Over 60 Java releases of iBATIS are available at SourceForge.net[1] and Apache.org[2]. The second application is *HealthWatcher* (HW) [26], a typical Java web-based information system. HW was first released in 2002 in both Java and AspectJ versions, and allows citizens to register, update and query complaints about health issues through a web client. The third system is a software product line for mobile devices called *MobileMedia* (MM) [20]. MM was originally developed in 2005 to allow users to manipulate image files in mobile devices. It has then evolved to 10 Java and AspectJ releases that support the manipulation of additional media files, such as audio and video files. Table III shows general characteristics of the three target systems. For more information about each of them, the reader may refer to the respective placeholder websites or to previous reports of these systems [20, 21, 25].

Following recent studies [23, 27], we target four releases of each system in our evaluation. These releases are iBATIS 01, 01.3, 01.5 and 02[3], HealthWatcher 01, 04, 07 and 10 and MobileMedia 01, 02, 03 and 06. These releases were selected as their evolution included a wide range of different fine-grained and coarse-grained changes (e.g. refactorings and functionality increments or removals). Moreover, these systems are rich in different kinds of non-crosscutting and crosscutting concerns. Along their evolution, a subset of functional and non-functional crosscutting concerns were modularised within aspects. Therefore, such applications are suitable for the study at hand, since we are interested in faults related to aspectisation (rather than software faults in general), to better understand the effect of aspect coupling on fault-proneness. Note that, according to our experimental goals, we are concerned with analysing systems not necessarily large but instead with a considerable amount of aspects which present different forms of coupling with base code. Moreover, such systems have been recently used in other important AOP empirical studies [20, 21, 26, 27, 28].

TABLE III
Target applications.

| | iBATIS | HealthWatcher | MobileMedia |
|---|---|---|---|
| Application type | data mapper framework | health vigilance application | product line for mobile data |
| Code availability | Java/AspectJ | Java/AspectJ | Java/AspectJ |
| # of releases | 60 / 4 | 10 / 10 | 10 / 10 |
| Releases considered | 4 | 4 | 4 |
| Avg. LOC | 11,000 | 6,000 | 3,000 |
| Avg. # of modules* | 264 | 132 | 39 |
| Avg. # of aspects | 46 | 23 | 10 |
| Evaluation procedure | testing | testing | interference analysis |

*Interfaces, classes and aspects.

iBATIS is the most complex system among the analysed ones, which yielded the largest dataset and on which we draw most of our analyses. On the other hand, HW and MM have reached higher maturity levels as they have undergone more corrective and perfective changes. Hence, their stable implementations resulted in less faults than iBATIS. HW and MM have also been target of a number of previous studies focusing on other equally-important quality attributes [20, 21, 26, 28].

### C. The Fault Collection

We collected and documented faults from the three systems by applying varied approaches during the different development phases, according to the systems' characteristics and the available information. Note that faults were only documented when they were noticed in the AO version but not in the OO counterpart. That is, only faults introduced during the aspectisation of the systems were reported for further analysis. Table IV summarises the results per system.

TABLE IV
Fault distribution in all systems.

| | System | | | Total |
|---|---|---|---|---|
| | iBATIS | HW | MM | |
| Pre-release faults | 44 | n/a | n/a | 44 |
| Post-release faults | 28 | 13 | 9 | 50 |
| Total | 72 | 13 | 9 | 94 |

Figure 2 depicts the fault distribution according to the number of modules that contain a specific number of faults. It considers the total number of faults assigned to a particular module considering all releases. For example, in iBATIS a single module contains 17 faults when we sum up releases 01, 01.3, 01.5 and 02, while two modules contain 4 faults. For some cases in MM, more than one module have been assigned a common fault, therefore resulting in fault counts such as 2.5 and 0.5.

***iBATIS:*** iBATIS experienced two testing phases: pre-release and post-release testing. The former was performed by developers of AO releases and aimed to produce fault-free code

[1] *http://sourceforge.net/projects/ibatisdb/files/* - 23/05/10

[2] *http://archive.apache.org/dist/ibatis/binaries/ibatis.java* - 23/05/10

[3] Original SourceForge builds are #150, #174, #203 and #243, respectively.

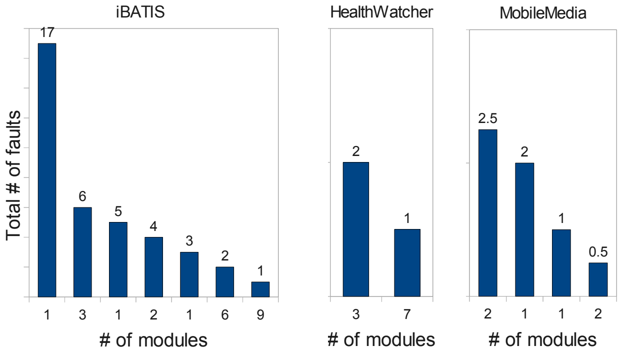

Fig. 2. Fault distribution per module in the target systems.

to be committed to a CVS repository. Developers used the original test sets from OO implementations as baselines in this phase. Any fault resulting in abnormal behaviour when regressively testing the AO implementations was readily reported. In the post-release phase, we extended the original test sets to increase the code coverage aiming to reveal faults that were not noticed by the developers. We focused on modules affected by the aspectisation process, achieving coverages of at least 80% of the related code [27].

***HW:*** Differently from iBATIS, HW releases experienced only post-release fault reporting. We then developed a full test set from scratch, based on the system specification and code documentation. In order to reduce test effort and avoid systematic bias during test creation, test cases were automatically generated with adequate tool support.

***MM:*** Post-release tests of MM revealed faults (mostly related to data input validation), which were also present in the OO counterparts. We also evaluated MM using the Composition Integrity Framework (CIF) [29]. CIF helped us identify faults caused from aspect interactions established either between aspects and base code or among aspects.

### D. Metrics Collection

All Base Coupling Metrics and Aspect Coupling Metrics were manually collected using the Cross Reference View in Eclipse[4] as it shows crosscutting information between the base and aspect code. We accounted for all potential matches exhibited by the IDE. The three metrics from the Baseline group were collected using the AOPMetrics tool [30].

### E. The Statistical Analysis

We conducted a Spearman's rank correlation test between metrics and fault counts per module. To evaluate the correlations, we used Hopkin's criteria[5] to judge the goodness of the coefficients: less than 0.1 means trivial, 0.1-0.3 means minor, 0.3-0.5 means moderate, 0.5-0.7 means large 0.7-0.9 means very large, and 0.9-1 means almost perfect.

The commonly accepted Pareto principle[6] states that 80% of the faults occur in 20% of modules in a system. For this reason, it is important to distinguish between modules that have (i) extremely high fault count (upper 20%), (ii) low number of faults, and (iii) no faults. Thus, fault data and each metric are classified into three groups. The group assignment was decided using the interquartile range of the dataset. The fault classification algorithm is shown in Figure 3, the same logic is also followed to classify each metric. We found no need for separate categories for the lower two quartiles as, due to the distribution of the data, over half the modules had 0 faults. In one or two cases where the upper metric quartile was 0 we followed lines 2-4 to divide the groups and avoid assigning each value into the same group.

```
1   If( 3rd Quartile = 0) {
2     If (#Faults = 0) Classification := 0
3     Else If (#Fault <= 1) Classification := 1
4     Else Classification := 2
5   }
6  Else {
7     If (#Faults >= 3rd Quartile) Classification := 2;
8     Else If (#Faults >= 2nd Quartile) Classification := 1;
9     Else Classification := 0
10  }
```

Fig. 3. Data classification algorithm.

To enhance our analysis, we also decided to employ binary logistic regression analysis to develop fault prediction models based on the studied metrics. We applied logistic regression instead of traditional linear regression analysis because of the lack of variability in the dependent variable (i.e. faults per module). This test requires the dependent variable to be binary so we classified the module as "0" for no faults and "1" for one or more faults. Logistic regression also does not require other assumptions required by some statistical techniques such as normally distributed variables or homoscedasticity [31]. Moreover, other studies have shown that logistic regression can provide suited models for fault-proneness prediction [1, 31].

Historically, significance levels of 0.01, 0.05 and 0.1 have been used for statistical tests [10, 32]. Although some of our tests were significant at levels as low as 0.01, we decided to adopt a significance level of 0.1, mainly motivated by our sample size, which was not very large. Other studies have also adopted such p-value threshold [33, 34]. Thus, our analyses consider p-values below 0.1 significant. For all statistical tests we used the R language and environment[7].

## V. Results

### A. Coupling Metrics

Table V shows descriptive statistics of each system. We collected the minimum (min), maximum (max), mean ($\mu$), standard deviation ($\sigma$) and median (med) for each metric collected from each system. Low $\mu$ values and higher $\sigma$ values indicate skewed measure distributions. This happens because a module is usually strongly coupled to only a few other modules [35]. It also explains why most medians are zero. The maximum and minimum values are not exact because we considered averages across all releases of each system.

### B. Correlation Rank and Regression Analysis

Table VI presents the Spearman's rank correlation results for all metrics. Correlations (i.e. $\geq 0.1$ according to Hopkin's criteria – see Section IV-E) are highlighted in grey. Note that in Table VI we used short names or acronyms for the metrics (e.g. *Before* in the Base Coupling group stands for Base Coupling via *Before* Advice).

[4] *http://www.eclipse.org/ajdt/* - 23/05/10

[5] *http://www.sportsci.org/resource/stats* - 23/05/10

[6] *http://www.gassner.co.il/pareto/* - 24/08/10

[7] *http://www.r-project.org/* - 23/05/10

TABLE V
DESCRIPTIVE STATISTICS RESULTS.

| Metric | iBATIS | | | | | HW | | | | | MM | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | max | $\mu$ | $\sigma$ | med | min | max | $\mu$ | $\sigma$ | med | min | max | $\mu$ | $\sigma$ | med |
| Baseline metrics | | | | | | | | | | | | | | | |
| Depth of Inheritance Tree (DIT) | 0 | 4.0 | 0.43 | 0.85 | 0.00 | 0 | 4.0 | 0.65 | 0.87 | 0.00 | 0 | 3.0 | 0.97 | 1.14 | 0.83 |
| Coupling between Modules (CBM) | 0 | 47.8 | 1.78 | 3.91 | 1.00 | 0 | 29.7 | 3.04 | 4.01 | 1.50 | 0 | 14.0 | 2.51 | 3.34 | 1.00 |
| Coupling on Advice Execution (CAE) | 0 | 2.3 | 0.27 | 0.53 | 0.00 | 0 | 4.5 | 0.62 | 0.86 | 0.00 | 0 | 5.0 | 0.55 | 1.03 | 0.00 |
| Base Coupling metrics | | | | | | | | | | | | | | | |
| Base Coupling via Before Advice | 0 | 15.3 | 0.12 | 1.09 | 0.00 | 0 | 19.8 | 0.16 | 1.58 | 0.00 | 0 | 1.0 | 0.05 | 0.19 | 0.00 |
| Base Coupling via Around Advice | 0 | 27.8 | 0.48 | 1.86 | 0.00 | 0 | 10.0 | 0.42 | 1.28 | 0.00 | 0 | 4.3 | 0.33 | 0.86 | 0.00 |
| Base Coupling via After Throwing Advice | 0 | 7.5 | 0.31 | 0.92 | 0.00 | 0 | 14.5 | 0.10 | 1.15 | 0.00 | 0 | 4.0 | 0.11 | 0.56 | 0.00 |
| Base Coupling via After Advice | 0 | 5.5 | 0.03 | 0.34 | 0.00 | 0 | 3.0 | 0.20 | 0.51 | 0.00 | 0 | 4.0 | 0.16 | 0.58 | 0.00 |
| Base Coupling via After Returning Advice | 0 | 2.5 | 0.02 | 0.20 | 0.00 | 0 | 18.5 | 0.12 | 1.47 | 0.00 | 0 | 2.0 | 0.10 | 0.39 | 0.00 |
| Base Coupling via Declare Parents | 0 | 2.0 | 0.04 | 0.20 | 0.00 | 0 | 1.8 | 0.17 | 0.42 | 0.00 | 0 | 0.0 | 0.00 | 0.00 | 0.00 |
| Base Coupling via ITD | 0 | 11.0 | 0.41 | 1.28 | 0.00 | 0 | 4.0 | 0.05 | 0.37 | 0.00 | 0 | 5.0 | 0.20 | 0.75 | 0.00 |
| Base Coupling via Declare Soft | 0 | 7.5 | 0.31 | 0.92 | 0.00 | 0 | 30.0 | 1.02 | 3.49 | 0.00 | 0 | 10.0 | 0.38 | 1.51 | 0.00 |
| Aspect Coupling metrics | | | | | | | | | | | | | | | |
| Aspect Coupling via Before Advice | 0 | 12.8 | 0.72 | 2.79 | 0.00 | 0 | 18.5 | 1.00 | 3.63 | 0.00 | 0 | 1.0 | 0.27 | 0.46 | 0.00 |
| Aspect Coupling via Around Advice | 0 | 49.0 | 4.06 | 9.62 | 0.00 | 0 | 24.0 | 3.11 | 6.18 | 0.00 | 0 | 4.5 | 1.15 | 1.45 | 1.00 |
| Aspect Coupling via After Throwing Advice | 0 | 6.8 | 1.47 | 1.70 | 0.75 | 0 | 14.5 | 0.59 | 2.84 | 0.00 | 0 | 4.8 | 0.33 | 1.12 | 0.00 |
| Aspect Coupling via After Advice | 0 | 5.5 | 0.18 | 0.77 | 0.00 | 0 | 8.7 | 1.24 | 1.17 | 0.00 | 0 | 3.0 | 0.64 | 0.95 | 0.00 |
| Aspect Coupling via After Returning Advice | 0 | 2.3 | 0.09 | 0.36 | 0.00 | 0 | 18.5 | 0.72 | 3.63 | 0.00 | 0 | 4.0 | 0.41 | 1.01 | 0.00 |
| Aspect Coupling via Declare Parents | 0 | 3.0 | 0.20 | 0.54 | 0.00 | 0 | 11.8 | 1.06 | 2.59 | 0.00 | 0 | 0.0 | 0.00 | 0.00 | 0.00 |
| Aspect Coupling via ITD | 0 | 45.5 | 2.87 | 8.86 | 0.00 | 0 | 4.0 | 0.33 | 0.91 | 0.00 | 0 | 10.0 | 1.23 | 2.41 | 0.00 |
| Aspect Coupling via Declare Soft | 0 | 21.8 | 2.36 | 3.78 | 1.00 | 0 | 50.8 | 6.04 | 12.65 | 0.00 | 0 | 16.5 | 1.11 | 3.62 | 0.00 |

Table VII shows the results from our regression analysis. The Coefficient (column *coeff*) is the estimated regression coefficient. The stronger the impact of the corresponding metric on the likelihood of a module presenting a fault, the larger the absolute value of the related coefficient (positive or negative according to its signal). The p-value (column *p-v*) provides the significance of the corresponding coefficient and is related to the statistical hypothesis. The (log) odds ratio (column *odds*) indicates the relative amount by which the odds of a module presenting a fault increase or decrease when the value of the metric is increased by a unit. Similarly to Table VI, we used short metric names in the first column. We also highlight results significant at 90% confidence level.

TABLE VI
SPEARMAN'S CORRELATION RANK RESULTS.

| Metric | iBATIS | | HW | | MM | |
|---|---|---|---|---|---|---|
| | p-v | coeff | p-v | coeff | p-v | coeff |
| Baseline metrics | | | | | | |
| DIT | 0.495 | -0.04 | 0.005 | -0.22 | 0.015 | -0.29 |
| CBM | **0.069** | **0.10** | 0.188 | 0.11 | **0.002** | **0.36** |
| CAE | 0.702 | -0.02 | 0.017 | -0.19 | 0.087 | -0.21 |
| Base Coupling metrics | | | | | | |
| Before | 0.174 | 0.08 | 0.487 | -0.06 | 0.485 | -0.08 |
| Around | 0.743 | -0.02 | 0.078 | -0.14 | 0.276 | -0.13 |
| After Throwing | 0.431 | 0.05 | 0.653 | -0.04 | 0.536 | -0.08 |
| After | 0.272 | 0.07 | 0.134 | -0.12 | 0.366 | -0.11 |
| After Returning | **0.000** | **0.29** | 0.715 | -0.03 | 0.485 | -0.09 |
| Declare Parents | 0.925 | -0.01 | 0.171 | -0.11 | n/a* | n/a* |
| ITD | 0.883 | -0.01 | 0.559 | -0.05 | 0.366 | -0.11 |
| Declare Soft | 0.334 | 0.06 | **0.000** | **0.28** | 0.366 | -0.11 |
| BBC | 0.687 | 0.02 | 0.030 | -0.17 | 0.127 | -0.18 |
| BSC | 0.754 | 0.02 | **0.067** | **0.15** | 0.252 | -0.14 |
| Aspect Coupling metrics | | | | | | |
| Before | 0.250 | 0.16 | 0.973 | 0.01 | 0.122 | 0.34 |
| Around | 0.375 | -0.12 | **0.066** | **0.37** | 0.706 | 0.09 |
| After Throwing | **0.005** | **0.37** | **0.022** | **0.45** | 0.621 | 0.11 |
| After | 0.253 | 0.16 | 0.543 | -0.13 | **0.041** | **0.44** |
| After Returning | **0.068** | **0.25** | **0.022** | **0.45** | **0.000** | **0.83** |
| Declare Parents | 0.278 | 0.15 | 0.044 | -0.40 | n/a* | n/a* |
| ITD | 0.411 | -0.11 | 0.966 | -0.01 | 0.211 | 0.28 |
| Declare Soft | **0.018** | **0.32** | **0.008** | **0.51** | 0.793 | -0.06 |
| ABC | **0.007** | **0.36** | **0.063** | **0.37** | **0.004** | **0.60** |
| ASC | **0.016** | **0.32** | 0.709 | 0.08 | 0.429 | 0.18 |

* Mobile Media had no Declare Parents occurrences

## VI. ANALYSIS

The following subsections discuss our two research hypotheses (Section IV-A) in the light of the results of Section V.

### A. Coarse- vs Fine-Grained Metrics (H1)

The proposed metrics for base coupling and aspect coupling are fine-grained in that (i) they measure coupling induced by individual mechanisms (excluding the four summative metrics Base Behavioural Coupling, Base Structural Coupling, Aspect Behavioural Coupling and Aspect Structural Coupling), and (ii) they take into account coupling frequency by counting each coupling connection. By contrast, the baseline metrics (Depth of Inheritance Tree (DIT), Coupling Between Modules (CBM)

TABLE VII
LOGISTIC REGRESSION RESULTS.

| Metric | iBATIS | | | HW | | | MM | | |
|---|---|---|---|---|---|---|---|---|---|
| | p-v | coeff | odds | p-v | coeff | odds | p-v | coeff | odds |
| Baseline metrics | | | | | | | | | |
| DIT | 0.50 | -0.38 | 0.68 | 0.99 | -9.24 | 0.00 | 0.99 | -17.42 | 0.00 |
| CBM | **0.06** | **0.49** | 1.63 | 0.35 | 0.36 | 1.43 | **0.02** | **2.30** | **9.94** |
| CAE | 0.75 | -0.17 | 0.84 | 0.06 | -1.38 | 0.25 | 1.00 | -15.25 | 0.00 |
| Base Coupling metrics | | | | | | | | | |
| Before | **0.05** | **0.95** | **2.58** | 0.99 | -14.36 | 0.00 | 1.00 | -15.22 | 0.00 |
| Around | 0.77 | -0.08 | 0.92 | 0.99 | -13.54 | 0.00 | 0.99 | -8.19 | 0.00 |
| After Throwing | 0.62 | 0.14 | 1.15 | 0.99 | -14.01 | 0.00 | 1.00 | -14.02 | 0.00 |
| After | 0.77 | -0.08 | 0.92 | 0.99 | -15.83 | 0.00 | 1.00 | -15.09 | 0.00 |
| After Returning | 0.99 | 9.04 | >10 | 1.00 | -13.54 | 0.00 | 1.00 | -8.14 | 0.00 |
| Declare Parents | 0.69 | 0.25 | 1.29 | 0.99 | -8.02 | 0.00 | n/a* | n/a* | n/a* |
| ITD | 0.86 | -0.06 | 0.94 | 0.99 | -13.39 | 0.00 | 1.00 | -15.59 | 0.00 |
| Declare Soft | 0.31 | 0.26 | 1.29 | **0.00** | **1.08** | **2.93** | 1.00 | -14.30 | 0.00 |
| BBC | 0.26 | 0.14 | 1.15 | 0.99 | -13.77 | 0.00 | 1.00 | -14.79 | 0.00 |
| BSC | 0.49 | 0.13 | 1.13 | 0.18 | 0.32 | 1.38 | 1.00 | -8.43 | 0.00 |
| Aspect Coupling metrics | | | | | | | | | |
| Before | 0.39 | 0.44 | 1.56 | 0.84 | -0.11 | 0.89 | 0.16 | 0.73 | 2.08 |
| Around | 0.73 | -0.13 | 0.88 | 0.10 | 0.82 | 2.28 | 0.96 | 0.03 | 2.45 |
| After Throwing | **0.02** | **0.89** | **1.15** | 1.00 | 18.14 | >10 | 0.47 | 0.55 | 1.73 |
| After | **0.04** | **0.97** | **2.63** | 0.55 | -0.31 | 0.74 | 0.09 | 0.90 | 0.00 |
| After Returning | 0.17 | 0.90 | 2.46 | 1.00 | 18.14 | >10 | 0.99 | 10.32 | >10 |
| Declare Parents | 0.15 | 0.63 | 1.88 | 0.99 | -9.12 | 0.00 | n/a* | n/a* | n/a* |
| ITD | 0.97 | 0.02 | 1.02 | 0.79 | -0.17 | 0.85 | 0.44 | 0.40 | 1.50 |
| Declare Soft | **0.04** | **0.74** | **2.10** | **0.02** | **1.18** | **3.27** | 0.91 | -0.07 | 0.93 |
| ABC | **0.01** | **0.61** | **1.83** | **0.08** | **0.77** | **2.15** | **0.02** | **0.69** | **1.99** |
| ASC | **0.01** | **0.96** | **2.60** | 0.91 | 0.04 | 1.04 | 0.54 | 0.31 | 1.36 |

* Mobile Media had no Declare Parents occurrences

and Coupling on Advice Execution (CAE)) are coarser as they aggregate individual modules.

Among the baseline metrics, CBM shows the highest correlation with faults, varying from minor in iBATIS to moderate in MobileMedia. CAE and DIT, however, do not show significant correlation. The lack of correlation from DIT is surprising as large inheritance trees in OO programs have shown to increase the likelihood of faults in a module [1, 2]. This might be because none of the target systems contain dangerously large inheritance hierarchies (max depth of 4), thus hiding the potential influence of high values of this metric.

Interestingly, the CAE metric shows no significant correlation in any of the applications. This contrasts with new metrics which also measure coupling between the base and aspect code, which in fact do show a correlation. If we consider new metrics such as Base Coupling via both After Returning Advice (Table VI) and Before Advice (Table VII) the correlation found here indicates that these AOP constructs may impact negatively on the fault-proneness of an application in the presence of high coupling. One explanation for this difference is that CAE counts the number of aspects involved, but remains insensitive to the numerous interactions that could potentially occur between a single aspect and a class.

To illustrate this point, iBATIS contains a class, `GeneralStatement`, that is advised 19 times by two aspects `ErrorContextAspect` and `ExecuteStatementObserver` as seen in Figure 1. Not surprisingly, this intense coupling between the `GeneralStatement` class and the two aspects caused complex interdependencies between the base and aspect code. This was reflected in the high fault count for these modules; In total, 23 faults were associated with them. The CAE metric, on the other hand, only aggregates 2 connections as it only counts the number of involved modules, thus not proportionately representing the fault-proneness in the value.

The Spearman's correlation presents some interesting patterns in iBATIS, the largest system in our experiment. Aspect Coupling via After, After Returning, After Throwing and Around Advice all show correlations with fault-proneness in at least one application. Within this group, logistic regression analysis confirms correlations of Aspect Coupling via After and After Throwing Advice. These metrics all measure advice with "after" semantics, potentially being more fault-prone than other types of advice due to additional complexity involved. For instance, Aspect Coupling via After Throwing and After Returning Advice show the strongest correlation with faults. Both mechanisms force developers to consider not only the advised method, but also ensure that the program semantics is correctly transferred to the next point of execution. This can be complex for methods with multiple implicit return locations. For instance, we found multiple scenarios where an exception handling aspect advised a single method at multiple join points. Other aspects that advised the same method, and even the same join point, added to the complexity.

Finally, among the summative metrics, Aspect Behavioural Coupling presents the strongest and most consistent levels of correlation ranging from 0.36 in iBATIS to 0.60 in MobileMedia. It shows significant results in both Spearman's correlation and logistic regression tests across all applications. This compound metric is the sum of all behavioural import metrics, four of which are advice with "after" semantics, which tend to confirm our above discussion.

Our results indicate that certain metrics from Base Coupling group and, even more so, Aspect Coupling group correlate better with faults than the baseline metrics. This may be because they are more proportionately representative of fault-proneness in a module than the baseline metrics. In particular, coupling imposed by after advice shows promising correlations. These results support the alternative hypothesis H1-1, which states that *finer-grained coupling metrics are more effective than coarse-grained coupling metrics when used as indicators of fault-proneness in AO programs.*

### B. *Import vs Export Metrics (H2)*

During the evolution of AO applications, both base and aspect modules are modified, and therefore faults are naturally distributed amongst both aspects and classes. If we only measure export coupling via advice, we are quantifying the number of advices invoked from the base code (or number of times a base module is advised by aspects). Of course, these types of metrics do not quantify the number of times an individual aspect advises the base code modules. In the following analysis we investigate both import and export coupling and note key differences with respect to AOP.

*1) Behavioural Metrics:* We first consider behavioural metrics as they show different results within this analysis. Import behavioural coupling metrics generally outperformed export metrics. For example, the import metric for Aspect Coupling via After Throwing Advice shows a moderate correlation (coefficient of 0.37) within iBATIS; however its export coupling counterpart (i.e. Base Coupling via After Throwing Advice) shows no significant correlation. This is supported by the logistic regression analysis (Table VII).

Combined metrics show similar results. As previously mentioned, (import) Aspect Behavioural Coupling metric shows significant correlations across all systems. However, when considering the equivalent export metric – i.e. Base Behavioural Coupling – Tables VI and VII do not show any significant correlation or fault prediction ability.

*2) Structural Metrics:* Regarding the 9 metrics that quantify structural coupling, Aspect Coupling via Declare Soft within Health Watcher results shows significant results when considering both Spearman's rank correlation (Table VI) and logistic regression analysis (Table VII), however no correlation can be found with the other two structural metrics. Despite this, previous research [27] has supported our findings on how certain structural mechanisms, such as Declare Soft, can be particularly fault prone.

Combining the structural metrics together do not increase the correlation. In fact, when combining the three structural *import* metrics the coefficient remains the same, and when combining the three structural *export* metrics the coefficient lessens. Despite this, it is important not to out-rule structural coupling as an indicator of faults. The community still has much more to learn about the impact of these constructs, and in particular with the use of Declare Soft.

The difference in effectiveness between import and export coupling metrics depends on key metric characteristics. Differences between import and export *structural* coupling were inconclusive as only Aspect Coupling via Declare Soft showed significant results. However, most import *behavioural* coupling metrics outperformed their export equivalent. Due to these variations we feel our results support the alternative hypothesis H2-1 that *there is a difference between the effectiveness of import and export coupling metrics as indicators of fault-proneness on AO programs*.

## VII. Experiment Limitations

Study limitations are discussed based on the four categories of validity threats described by Wohlin et al. [36]. For each category, we list all possible threats and the measures we took in order to reduce each risk when applicable.

***Conclusion validity:*** (i) *random heterogeneity of subjects* – evaluated systems come from different application domains; and (ii) *low statistical power* – the analysed data set might not be large enough to allow for deep statistical analyses. Both risks could not be totally avoided. However, the heterogeneity of applications helps to promote the external validity of the study. Regarding risk (ii), the current lack of fault-related historical data and the current few options of AO systems available for evaluation pose an obstacle for in-depth analyses.

***Internal validity:*** (i) *ambiguity about direction of causal influence* – the complexity of the aspectised concerns might have made a system release more faulty than the others; and (ii) *history and maturation* – HW and MM systems are well-known subjects, extensively evaluated and continuously improved through the last years. Risk (i) cannot be completely avoided since functionalities commonly differ in complexity. However, it was reduced since all systems were developed and revised by experienced programmers. Moreover, systematic regression testing helped developers preserve the semantics of the OO counterparts. To reduce risk (ii), we focused our analyses on iBATIS, which consists in the most recent from all target systems and yielded the largest data set to be analysed.

***Construct validity:*** We identified a risk related to the *confounding constructs and levels of constructs* – different maturity levels of the investigated systems impacted the number of revealed faults. This risk could not be avoided due to the few options of medium-sized AO systems available for evaluation to date. Such systems present different maturity levels, including varied types and numbers of faults.

***External validity:*** The major risk here is related to the *interaction of setting and treatment* – the evaluated systems might not be representative of the industrial practice. However, the heterogeneity of these systems helps to reduce this risk. They are implemented in AspectJ, which is one of the representative languages in the state of AOP practice. iBATIS is a widely-used framework. Even though HW and MM are smaller applications, they are also heavily based on industry-strength technologies and have been extensively used and evaluated in previous research [20, 21, 26, 28]. To conclude, the characteristics of the selected systems, when contrasted with the state of practice, represent a first step towards the generalisation of the achieved results.

## VIII. Related Work

Despite the little research into fault-proneness of AO programs, in this section we summarise pieces of work that we believe are mostly related to our own. They are distributed in two categories: (i) studies that investigate internal metrics as indicators of external quality attributes such as design stability and maintainability; and (ii) studies that investigate the ability of internal metrics in predicting faults.

***Studies on metrics for AO software***: Apart from our previous research on metrics evaluation with respect to fault-proneness (described in Section II), other work includes an evaluation of a subset of Ceccato and Tonella's metrics [11] performed by Bartsch and Harrison [16]. The evaluation was based on a framework for describing coupling mechanisms in AspectJ and another for theoretical validation of measures.

Shen and Zhao [37] also conducted an empirical study to evaluate 16 coupling metrics for AO programs. The study included all Ceccato and Tonella's metrics [11] and seven new metrics proposed by the authors. Their new metrics take into account AOP-specific properties and elements such as crosscutting degree caused by pointcuts and intertype declarations. Differently from the fine-grained metrics we propose in this paper, however, Shen and Zhao's metrics do not (i) take into account the frequency of coupling connections between base and aspectual modules (ii) measure coupling from different types of advice, declare soft or declare parents statements. Moreover, Shen and Zhao analysed the correlation between the metrics and other attributes such as LOCC (Lines of Class Code) and the age of an evaluated release, and maintainability [14] in an extension of their prior work.

Other studies applied coupling metrics to evaluate the design stability of AO systems in the presence of evolution, commonly used as a surrogate measure for fault-proneness. Greenwood et al. [21] assessed the stability of HW releases using metrics suites for modularity and change impact analysis, which comprised separation of concerns, coupling, cohesion and conciseness attributes. Similar metrics suites were used by Figueiredo et al. [20] to evaluate the stability of MM.

***Metrics as faults predictors***: As described in the previous category, we have recently performed a first empirical investigation of metrics as fault predictors in AO systems [23]. However, we were not able to identify any other initiative so far. Nonetheless, one can find a number of recent studies that investigate the fault-proneness of OO elements as well as the validation of OO metrics for fault prediction.

For example, Gyimothy et al. [2] applied statistical methods and machine learning techniques to evaluate the correlation between CK metrics [13] and fault-proneness of a large open source OO system (around one million LOC) . Also, Olague et al. [31] evaluated two other OO metrics suites in terms of fault predication in six releases of a medium-sized Java open source system. The subjects of study are six releases of a medium-sized Java open source system. Differently, however, Olague et al. performed only statistical analysis and built some statistical models for fault prediction on top of it. The results of both studies show that different metrics (e.g. CBO and LOC in [2], and RFC and WMC in [31]) stood out w.r.t. to faulty classes prediction in the evaluated systems.

## IX. Conclusions

The analysis of internal software properties as indicators of external quality attributes such as maintainability and fault-proneness has been a common practice in software engineering. In particular, *coupling* has shown to be an important indicator of faults in OO programs [1, 2, 3, 9]. However, little research has investigated the impact of coupling on the fault-proneness of AO programs. To overcome this, we introduced a novel suite of fine-grained metrics to analyse *structural*, *behavioural*, *import* and *export* coupling originated from the use of various AOP-specific constructs. The effectiveness of these metrics against existing metrics as fault indicators was also investigated based on the analysis of three AO systems.

Results showed initial evidence that specific fine-grained sources of AO-related coupling have a greater impact on fault-proneness which was not captured by coarse-grained metrics such as CAE and CBM [11]. Moreover, the proposed *Aspect Behavioural Coupling* metric – a combination of import advice-related metrics – showed that behavioural coupling sourced from advice mechanisms was the best fault-proneness indicator amongst all investigated metrics. Amongst the existing Baseline metrics, only Coupling Between Modules showed significant though minor correlations with the number of faults in the applications. One of the potential downfalls with existing metrics is that they do not take into account the new and intricate dependencies introduced by AOP mechanisms, and thus are less proportionately representative of the impact of coupling upon fault-proneness.

The contributions of this paper show how sources of AOP-specific coupling impacts on the external quality of AO software. We feel these results might aid programming language designers in future developments of AOP languages. These results also motivate future refinements of these metrics including a stronger controlled assessments. To broaden our analysis on this topic future work also includes the assessment of alternative measures (such as network analysis [38]).

## Acknowledgements

The authors received the following financial support: *Rachel Burrows*: UK EPSRC grant; *Fabiano Ferrari*: FAPESP (grant 05/55403-6); *Otávio Lemos*: FAPESP (grant 2008/10300-3); *Alessandro Garcia*: FAPERJ (distinguished scientist grant E-26/102.211/2009), CNPq (productivity grant 305526/2009-0 and Universal Project grant number 483882/2009-7), and PUC-Rio (productivity grant).

We would like to thank Andrew Camilleri from Lancaster University (UK), Eduardo Figueiredo from UFMG (Brazil) and Nélio Cacho from UFRN (Brazil) for their help while analysing the MobileMedia and HealthWatcher systems.

## References

[1] K. El Emam, W. L. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of Systems and Software*, vol. 56, no. 1, pp. 63–75, 2001.

[2] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Soft. Eng.*, vol. 31, no. 10, pp. 897–910, 2005.

[3] R. Subramanyam and M. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Trans. Soft. Eng.*, vol. 29, no. 4, pp. 297–310, 2003.

[4] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. PWS Publ. Co., 1998.

[5] G. Kiczales et al., "Aspect-oriented programming," in *ECOOP'97*. Springer, 1997, pp. 220–242 (LNCS 1241).

[6] "JBoss AOP," *http://www.jboss.org/jbossaop/docs/index* -23/05/10.

[7] R. Johnson et al., "Spring - Java/J2EE application framework," Interface21 Ltd., Reference Manual V. 2.0.6, 2007.

[8] G. Kiczales et al., "Getting started with AspectJ," *Communications of the ACM*, vol. 44, no. 10, pp. 59–65, 2001.

[9] A. B. Binkley and S. R. Schach, "Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures," in *ICSE'98*, 1998, pp. 452–455.

[10] L. Briand, P. Devanbu, and W. L. Melo, "An investigation into coupling measures for C++," in *ICSE'97*. ACM, 1997, pp. 412–421.

[11] M. Ceccato and P. Tonella, "Measuring the effects of software aspectization," in *WARE Workshop*, 2004.

[12] C. Sant'Anna et al., "On the reuse and maintenance of aspect-oriented software: An assessment framework," in *SBES'03*. Brazilian Computer Society, 2003, pp. 19–34.

[13] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Soft. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[14] H. Shen, S. Zhang, and J. Zhao, "An empirical study of maintainability in aspect-oriented system evolution using coupling metrics," in *TASE'08*. IEEE, 2008, pp. 233–236.

[15] J. Zhao, "Measuring coupling in aspect-oriented systems," in *METRICS'04 (Late Breaking Paper)*, 2004.

[16] M. Bartsch and R. Harrison, "An evaluation of coupling measures for AspectJ," in *LATE Workshop*. ACM, 2006.

[17] R. Burrows, A. Garcia, and F. Taïani, "Coupling metrics for aspect-oriented programs: A systematic review of maintainability studies," in *ENASE'09*. Springer, 2009.

[18] L. C. Briand, J. W. Daly, and J. Wüst, "A unified framework for coupling measurement in object-oriented systems," *IEEE Trans. Soft. Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[19] T. T. Bartolomei, A. Garcia, C. Sant'Anna, and E. Figueiredo, "Towards a unified coupling framework for measuring aspect-oriented programs," in *SOQUA'06*. ACM, 2006, pp. 46–53.

[20] E. Figueiredo et al., "Evolving software product lines with aspects: An empirical study on design stability," in *ICSE'08*. ACM, pp. 261–270.

[21] P. Greenwood et al., "On the impact of aspectual decompositions on design stability: An empirical study," in *ECOOP'07*. Springer, 2007, pp. 176–200 (LNCS 4609).

[22] S. Yacoub, H. Ammar, and T. Robinson, "Dynamic metrics for object oriented designs," in *METRICS'99*. IEEE, 1999, pp. 50–61.

[23] R. Burrows, F. Ferrari, A. Garcia, and F. Taïani, "An empirical evaluation of coupling metrics on aspect-oriented programs," in *ICSE WETSoM Workshop*, 2010, pp. 53–58.

[24] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Soft. Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[25] "iBATIS," *http://ibatis.apache.org/* - 23/05/10.

[26] S. Soares, E. Laureano, and P. Borba, "Implementing distribution and persistence aspects with AspectJ," in *OOPSLA'02*. ACM, 2002.

[27] F. C. Ferrari et al., "An exploratory study of fault-proneness in evolving aspect-oriented programs," in *ICSE'10*. ACM, 2010, pp. 65–74.

[28] R. Coelho et al., "Assessing the impact of aspects on exception flows: An exploratory study," in *ECOOP'08*. Springer, 2008, pp. 207–234.

[29] A. Camilleri, G. Coulson, and L. Blair, "CIF: A framework for managing integrity in aspect-oriented composition," in *TOOLS'09*. Springer, 2009, pp. 18–26 (LNBIP v.33).

[30] "AOP Metrics," *http://aopmetrics.tigris.org/* - 23/05/10.

[31] H. M. Olague et al., "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Trans. Soft. Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[32] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *ICSE'04*. IEEE, 2004, pp. 282–292.

[33] O. Laitenberger and J.-M. DeBaud, "Perspective-based reading of code documents at Robert Bosch GmbH," *Information and Software Technology*, vol. 39, no. 11, pp. 781–791, 1997.

[34] M. Zhao, C. Wohlin, N. Ohlsson, and M. Xie, "A comparison between software design and code metrics for the prediction of software fault content," *Inf. and Soft. Technology*, vol. 40, no. 14, pp. 801–809, 1998.

[35] L. C. Briand, J. Wüst, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in *ICSM'99*. IEEE, 1999, pp. 475–482.

[36] C. Wohlin et al., *Experimentation in Software Engineering: an Introduction*. Kluwer, 2000.

[37] H. Shen and J. Zhao, "An evaluation of coupling metrics for aspect-oriented software," Center for Soft. Eng., SJTU, Shanghai - China, Tech. Rep. SJTU-CSE-TR-07-04, 2007.

[38] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *ICSE'08*. ACM, pp. 531–540.