An Empirical Study on Refactoring Objects to Aspects

Fernando Brito e Abreu, Sérgio Bryton

QUASAR/CITI - Center of Informatics and Information Technology
Faculty of Sciences and Technology
Lisbon New University, Portugal
{fba, bryton}@di.fct.unl.pt
http://citi.di.fct.unl.pt/

Abstract. It has been proclaimed in the literature that AOP allows obtaining better modularized systems than those built with OOP, namely by reducing tangling and scattering. Very few quantitative studies sustaining that claim were published. None is known to contradict it. We briefly describe an assessment process that allows performing paradigm-independent modularity assessments, based on a metamodel driven approach, and a set of paradigm-independent metrics. We use this assessment process upon the 23 GoF design patterns, that were refactored from Java to AspectJ, and then perform a set of statistical tests to answer research questions on the effect of this refactoring on overall coupling, cohesion and modularity.

Keywords: AOP, OOP, Refactoring, Modularity, Coupling, Cohesion, Metrics, Meta-Model Driven Measurement.

1 Introduction

A programming paradigm is a fundamental style of computer programming. Well-known examples are Procedural or Structured Programming (PP), Functional Programming (FP), Object-Oriented Programming (OOP) and Aspect-Oriented Programming (AOP). Programming paradigms usually have their hype in time, until a new paradigm arrives and a paradigm shift is initiated. In this paper we are concerned on understanding the impact of the shift from OOP to AOP, which encompasses making comparisons among systems built with different paradigms. Performing this kind of assessments is a difficult exercise since the characterization and quantification of software assets is generally performed using paradigm-specific characteristics and metrics. When a programming paradigm is more an extension than a disruption with an existing one (like happens with AOP face to OOP), things may get even worse, since there is a natural compulsion to perform comparisons based only upon what paradigms have in common (i.e., the features of the older paradigm). This is even fostered if the representative language of the newer paradigm is a hybrid language that also supports the older paradigm, as happens in this paper where we compare software systems written in perhaps the most widely used programming languages of each of the paradigms under study: Java for OOP and AspectJ for AOP. In this paper we present the results of an empirical study where we compared two sets of functionally equivalent source code examples, the 23 GoF design patterns, that were refactored from Java to AspectJ [11]. Our main objectives were to provide the grounds to answer the following research questions (i) Does OOP to AOP refactoring produce a significant change on the overall coupling, cohesion and modularity? (ii) And what about if we only consider the core modular features (classes, interfaces and aspects)?

This paper is organized as follows: on section 2 we briefly introduce our methodological approach to paradigm and language independent assessments; section 3 describes our empirical study, from collected data to the interpretation of the statistical tests' results; related work is described in section 4; we draw our conclusions in section 5 and, finally, we outline our future work in this research thread on section 6. Due to space constraints we chose not to present the tables with the relevant results from the statistical tests used, which are available in [6].

2 Methodological approach

In a nutshell, our methodological approach is the following: (i) expressing the domain knowledge with the Paradigm Independent Metamodel (PIMETA)[7]; (ii) defining the Paradigm Independent Modularity Factors (PIFACTORS), a set of factors that express distinct coupling and cohesion characteristics that, in our perspective, are influential on modularity[6]; (iii) using the Metamodel Driven Measurement (M2DM)[1] technique to quantify those factors with the Object Constraint Language (OCL) [14] upon the PIMETA, thus obtaining the Paradigm Independent Metrics (PIMETRICS)[6]; (iv) developing an automated process for metrics collection that allowed to obtain the PIMETRICS from our sample in a format that can (v) be imported in a statistical tool where our hypotheses were tested. We now provide some details on each step.

2.1 PIMETA

The PIMETA is instantiated in two steps. In a first step, the metaclasses *Paradigm*, *Formalism*, *AtomicFeatureType*, *ModularFeatureType* and *DependencyType* are instantiated with the information that describes the formalisms under study (in this case, the types of features and dependencies that occur in Java and AspectJ). Examples of Features can be classes, methods, aspects, parameters, or instance variables. Modular Features are those that can contain other Features and Atomic Features are those that cannot. For instance, in AspectJ a Pointcut Definition is a Modular Feature, since it may contain Pointcuts and Parameters (the latter is an Atomic Feature). A Dependency Type in AspectJ, for instance, could be a Method using an Intertype Field or calling an Intertype Operation. In a second step, the metaclasses *AtomicFeature*, *ModularFeature* and *Dependency* must be instantiated with actual occurrences of the features and dependencies in a given software asset under study.

2.2 PIFACTORS

Modularity can be assessed using the concepts of coupling and cohesion which, in turn, are the result of the dependencies that exist among the features from a software system. Therefore, we should study the different manifestations of those dependencies in terms of how the corresponding features participate in them. Notice that dependencies are directional (they always have a source and a destination feature). PIFACTORS are a categorization of those dependencies, as follows:

Role This factor regards to the part a feature plays in each dependency where it is involved; a feature acting as a source must know about the features on which it depends, since those may affect its purpose; if acting as a target, it should be aware of its probable impact on the features which rely on it.

Coverage This factor distinguishes features based on how many other different features are involved within its dependencies; for evaluating coupling, from the source perspective, the more different features a feature uses, the more that feature is coupled; for evaluating cohesion, the higher the coverage of the feature which is being analyzed, the higher is its contribute to the cohesion of the modular feature to which it belongs.

Commitment This factor expresses how much of a module is committed to the dependencies in which is involved; for evaluating coupling, the more different features, within this modular feature, are involved as sources in dependencies towards external features (i.e., those features that are not within the same modular feature as the one containing the dependency source), the more this modular feature is coupled; for evaluating cohesion, the more different features, within this modular feature, are involved as sources in dependencies towards internal features, the more this modular feature is cohesive.

Clustering This factor distinguishes features based on how many disjoint clusters are involved within the dependencies at which they are involved; disjoint clusters are modular features that do not share any of the modular features that contains them (i.e., a system composed by two packages; each package is a cluster disjoint from the other); notice that clustering only makes sense when evaluating coupling, since that for cohesion, only the dependencies from (and to) the features within the same modular feature are considered; the more different disjoint clusters contain targets of dependencies, the more the feature under evaluation is coupled.

2.3 PIMETRICS

Metrics should be formally defined to avoid subjectivity in their interpretation and collection. For that purpose we have used the Goal Question Metric (GQM)[4] approach together with the M2DM technique. This technique was grounded on the PIMETA and the PIFACTORS, from which the PIMETRICS set was derived and formalized as OCL expressions. An example of a PIMETRIC definition and formalization, is presented in Table 1. The complete PIMETRICS catalogue can be found in[6]. In addition to the formality granted by the use of this constraint language, OCL expressions can be automatically evaluated upon the instantiated metamodel, using an OCL evaluator, like the USE [10] tool.

Goal (G1)	**Decrease Features Coupling**	
Question (Q1.1)	What is the coupling coverage of a feature as a source?	
Metric (M1.1.1)	Acronym	SCouCovS
	Name	Source Coupling Coverage Size
	Informal definition	Number of different external features on which a feature depends
	Formal definition	PIMETA::Feature SCouCovS(): Integer = externalDependeeFeatures()->size()

Table 1. PIMETRICS definition and formalization example

2.4 Automated collection process

We developed an automated collection process to collect the PIMETRICS values from source code in Java and AspectJ, composed by five steps. In this process (i) the source code is parsed into XML (.ajfml files) using a parser for AspectJ and Java, the AspectJ-Front parser [5]. Then (ii) the output from the AspectJ-Front parser is transformed into XML files (.pimetaml files) compliant with the PIMETA XML schema, using a XSLT file (AJFML2PIMETAML). From here, another transformation occurs (iii) for generating input (.cmd files), with another XSLT file (PIMETAML2USE), for the USE tool, which is then used to (iv) instantiate the PIMETA. Finally, (v) the PIMETRICS, expressed as OCL expressions, are calculated with the USE tool, upon the previously loaded PIMETA.

3 Empirical Study

3.1 Sample description

The subjects of our experiment are a set of 2003 features (either atomic or modular) and 2039 dependencies, between these features, distributed by both implementation versions (Java and AspectJ) of the 23 GoF Design Patterns, as can be seen in Figure 1.

The number of features varies a lot from pattern to pattern, as well as between the GoF implementation versions, as we can see in Figure 2. In most (18) patterns in the GoF AspectJ implementation version, we see an increase in the number of existing features. Regarding modular feature types, all the Java ones except *Package* and all the AspectJ ones, except *AdviceBefore*, were found in the sample. As for atomic feature types, only the *Enumeration* was not found in the sample for Java, while for AspectJ only *DeclareParents* was found within the sample.

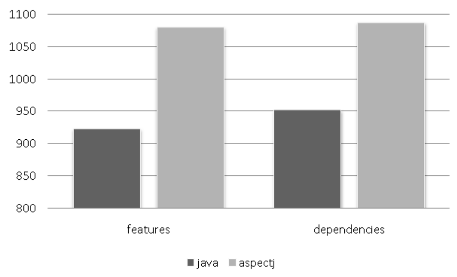

Fig. 1. Features and dependencies per GoF version

The number of dependencies also varies a lot from pattern to pattern, as well as between the GoF implementation versions, as we can see in Figure 3. Only in few (6) patterns in the GoF AspectJ implementation version, we see an increase in the number of existing dependencies. However, in the remainder 17 patterns, we hardly see a very pronounced decrease in the number of dependencies. Regarding the Java dependency types (see [6] for their full identification), 16 out of 23 possible ones had instances. As for the AspectJ dependency types, only 9 out of more than 70 possible ones (again see [6]) were found within the sample. It is worth mentioning that while the referred total number of dependency types were identified in the latest AspectJ documentation [3], the sample under study was published more than six years ago[11], when the features of the AspectJ version available at the time were less than today.

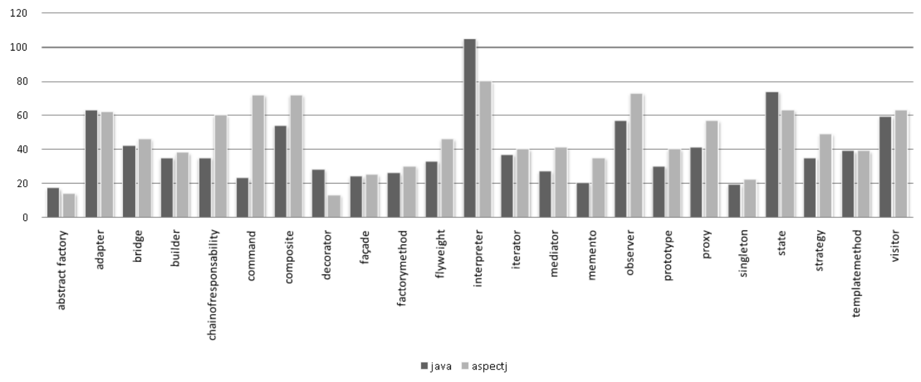

Fig. 2. Features per pattern per GoF version

Beyond the PIMETRICS, we also used the Potential Crosscutting Size (PCS) and the Potential Crosscutting Index (PCI) metrics, defined and formalized in [6], as well as the Average Module Membership (AMM), Intramodular Coupling Density (ICD) and the Modularization Merit Factor (MMF), defined and for-

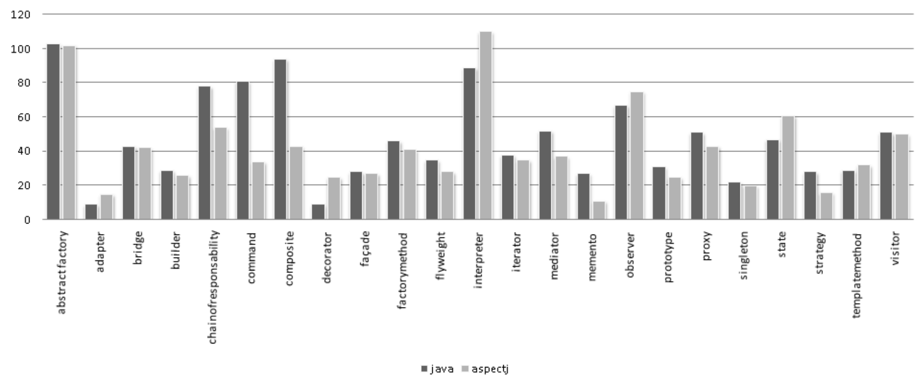

Fig. 3. Dependencies per pattern per GoF version

malized in [2]. All ratio metrics were found within the expected range [0, 1]. We have conducted an exploratory data analysis to eliminate (very few, in fact) extremes and outliers and to test for the Normal distribution adherence using the Kolmogorov-Smirnov one-sample test. None of the PIMETRICS could be considered as having a Normal distribution, so non-parametric tests had to be used.

3.2 Hypotheses identification and testing

– *Metrics discriminative power*

Research question Do the proposed metrics for coupling, cohesion and modularity vary significantly from pattern to pattern within the same implementation language? Although we can see in Figure 2 that the design pattern implementations, either on Java or AspectJ, differ significantly in size, we expect PIMETRICS to have some discriminative power. In other words, they are expected to show some kind of variance, depending on the considered design pattern. ***Hypotheses and statistical testing*** We want to know if each of the coupling, cohesion and modularity metrics, taken from the independent samples, corresponding to features of each of the design patterns, may be considered as drawn from the same population. In other words, the null hypothesis is that the values of those metrics do not vary significantly from design pattern to design pattern. We will perform two separate sets of tests, one for the Java patterns and another for the AspectJ patterns. Each individual test will refer to a given PIMETRIC. Since that none of the PIMETRICS has a normal distribution, we will use the Kruskal-Wallis one-way analysis of variance. In our case, the independent samples correspond to the 23 implementations of the GoF patterns for a given language. ***Results interpretation*** The Kruskal-Wallis H test statistic is distributed approximately as chi-square. Consulting a chi-square table with df=22 (degrees of freedom) and for a significance of α=0.05 (confidence level of

95%) we obtain a critical value of chi-square of 12.3. Since this value is less than the computed H values, we reject the null hypothesis that the samples do not differ on the criterion variable (the Design Pattern). In other words, given any of the metrics, we cannot sustain that the statistical distributions of the groups of modular features corresponding to each of the design patterns are the same. This means that we accept the alternative hypothesis that the design pattern has influence on the values of the proposed metrics. This confirms that the metrics have discriminative power.

– *Overall coupling changes*

Research question Does the refactoring of software systems produced with an OOP language to an AOP extension of the same language produces a significant change on the overall coupling? Regarding coupling, we expect that the AOP pattern versions are statistically different and, as claimed, exhibiting lower values for coupling from the OOP ones. ***Hypotheses and statistical testing*** If we consider that we have two independent groups of cases, one for the Java language and another for the AspectJ language, we can compare the averages for each of the testing variables in the two groups. In other words we want to test if there is a significant difference between the averages of the variables describing cohesion, for the two languages. The adequate test is called the Independent Samples t-Test for the equality of the means. If both groups have sizes above 30 cases (which is our case), then the distribution t with v degrees of freedom *t(v)*, approaches the Normal distribution and the test can be applied. **Results interpretation** Based upon the result of the Levene's test and an α=0.05, we reject the hypothesis of the equality of variances between the two groups, for all testing variables. Therefore we have to choose the rows signalized with "Equal variances not assumed". The positive value for the t statistic shows us that the values of the coupling metrics are, on the average, superior for Java than for AspectJ. Considering a test significance α=0.05 we reject the null hypotheses - the averages of all coupling metrics are the same between the two groups. In other words, we can say that there is statistical evidence that the coupling for the OOP versions is, on average, higher than the coupling for AOP module versions.

– *Core modular features coupling changes*

Research question Does the refactoring produced with an OOP language to an AOP extension of the same language produces a significant change on the coupling of the core modular features (classes and interfaces common to Java and AspectJ GoF versions)? We expect the AOP version to be statistically different and exhibiting the claimed lower coupling than the OOP version. With a rotation operation we were able to produce a paired list version of our data, so that for equivalent modular features, we get two variables within the same case, representing the values for the same variable in Java and in AspectJ, like for instance: TCouComI-Java and TCouComI-AJ. Notice that we get fewer cases than for independent tests, because only classes and interfaces are pairable. ***Hypotheses and statistical testing*** The hypotheses are the same as above in section 3.2. The test used was be the Paired Samples T-Test. ***Results interpretation*** The positive value for the t statistic shows us that the values of the coupling metrics are on the average superior for Java than for AspectJ. Considering a test significance α=0.10 we can conclude that for some paired variables we can reject the null hypotheses that the averages of both coupling variables are the same between the two groups, and for others we cannot. Specifically, we can say that: (i) for coupling coverage (SCouCovI, TCouCovI), for coupling clustering on the source perspective (SCouCluI) and for coupling commitment also on the source perspective (SCouComI) there is statistical evidence that the coupling for the OOP modules version is, on average, higher than the coupling of the corresponding AOP modules version; (ii) for coupling clustering on the target perspective (TCouCluI) and for coupling commitment also on the target perspective (TCouComI), and with a confidence level as low as 90%, there is no statistical evidence that the coupling for the OOP modules version is, on average, different from the coupling of the corresponding AOP modules version.

– *Overall cohesion changes*

Research question Does the refactoring of software systems developed with an OOP language to an AOP extension of the same language produces a significant change on the overall cohesion? We expect the AOP pattern versions to be statistically different and exhibiting the claimed higher cohesion than the OOP pattern versions. ***Hypotheses and statistical testing*** If we consider that we have two independent groups of cases, one for the Java and another for the AspectJ languages, we can compare the averages for each of the testing variables in the two groups. In other words, we want to test if there is a significant difference between the averages of the variables describing cohesion, for the two languages. We will use the Independent Samples t-Test. ***Results interpretation*** Again based upon the result of the Levene test for α=0.05, we rejected the hypothesis of the equality of variances between the two groups, for both testing variables. Again we choose the rows signalized with "Equal variances not assumed". The positive value for the t statistic shows us that the values of the cohesion metrics are on the average superior for Java than for AspectJ. Considering a test significance α=0.05, we reject the null hypotheses that the averages of both cohesion variables are the same between the two groups. In other words, there is statistical evidence that the cohesion for the OOP versions is, on average, higher than the cohesion for the AOP ones. These results do not corroborate the most common claims regarding cohesion of AOP systems. As explained earlier, we believe these results do not evidence the real cohesion differences but, instead, the differences between the *delusional* cohesion at the OOP implementations, due to the code tangling and the cohesion of their AOP equivalent versions.

– *Core modular features cohesion changes*

Research question Does the refactoring of software systems developed with an OOP language to an AOP extension of the same language produces a significant change on the cohesion of the core modular features? Again, with a rotation operation we get two variables within the same case, representing the values for the same variable in Java and in AspectJ, like for instance: TCohComI-Java and TCohComI-AJ. ***Hypotheses and statistical testing*** The hypotheses are the same as above in section 3.2. The test we have used was the Independent Samples T-Test. ***Results interpretation*** The positive value for the t statistic shows us that the values of the cohesion metrics are on the average superior for Java than for AspectJ. Considering a test significance α=0.10 we reject the null hypotheses that the averages of both cohesion variables are the same between the two groups. Again we confirm that there is statistical evidence that the cohesion for the OOP modules version is, on average, higher than the cohesion of the corresponding AOP modules version.

– *Overall modularity changes*

Research question Does the refactoring of software systems developed with an OOP language to an AOP extension of the same language produce a significant change on the overall modularity? We expect the AOP pattern versions to be statistically different and exhibiting the claimed higher modularity than the OOP pattern versions. ***Hypotheses and statistical testing*** If we consider that we have two independent groups of cases, one for the Java language and another for the AspectJ language, we can compare the averages for each of the testing variables in the two groups. In other words, we want to test if there is a significant difference between the averages of the variables describing modularity, for the two languages. We have used the Independent Samples t-Test. ***Results interpretation*** Based upon the result of the Levene's test and an α=0.05 we reject the hypothesis of the equality of variances between the two groups, for both testing variables. Therefore we choose the rows signalized with "Equal variances not assumed". The positive value for the t statistic shows us that the values of PCI is on the average superior for Java than for AspectJ. Considering a test significance α=0.05, we reject the null hypotheses (the averages of the potential crosscutting is the same between the two groups), meaning that there is statistical evidence that the potential crosscutting for the OOP versions is, on average, higher than the potential crosscutting for the AOP versions, which, as explained earlier, between functional equivalent systems, also means that modularity is higher on the AOP versions of those systems. The negative value for the t statistic shows us that the values of MMF is on the average inferior for Java than for AspectJ. Considering a test significance α=0.05, we reject the null hypotheses that the averages of the modularity merit factor is the same between the two groups, meaning that there is statistical evidence that the MMF for the OOP versions is, on average, lower than the MMF for the AOP versions or, in other words, that modularity is higher on the AOP versions of those systems. 3.3.

3.3 Threats to validity

The dependency type weight has been set to 1, for not being known, up to date, an appropriate scale which can be used for the dependencies of Java or

AspectJ. This implies the consent that all dependencies have the same effect on modularity, in spite of not being at all our conviction. The design patterns implementation in Java and AspectJ is a very interesting case study, yet small and not representative of a real world system. The sample size is very important to externally validate the metrics with, for instance, maintenance effort. To improve the reliability of the automated data collection and measurement process, more validation and enhancement is required. The AspectJ-Front parser evidenced the lack of some features that would be of value like, for instance, an xml schema for the generated xml files. The developed parser still has limitations like when dealing with different features with the same name (e.g., a method and a field).

4 Related Work

Hannemann and Kiczales presented a comparative study, where they claimed modularity improvements in 17 of the 23 design patterns [11]. Although being a qualitative study, it should be taken into consideration the empirical understanding of the paradigm effects on modularity, given the authors' expertise on the subject.

Proposals based on DSM claim [13] that metrics are not thought for assessing design options at certain decision points, but can be useful for a posteriori analysis. We do not subscribe to this opinion since the PIMETRICS can also be used for design evaluations, as long as the PIMETA is instantiated with the features from the design language being used. Lopes and Bajra [13] proposed the assessment of aspect modularizations by using the concepts of Design Structure Matrix (DSM) and Net Option Value (NOV), and present a case study where several OOP and AOP variants for a software application are compared and analyzed. They concluded that aspects can be beneficial as well as detrimental. This study was grounded on a small, however relatively complex example, which inhibited more ambitious conclusions beyond the scope of the system itself. Cai and Huynh [9] proposed an evolution model for software modularity assessment using a decision tree based assessment framework, grounded on augmented constraint networks. The authors claimed its independency of particular languages and paradigms and that the proposed model quantitatively corroborates the qualitative claims of Hannemann and Kiczales[11] about the modularity improvements on the Observer design pattern. However, the fact of only encompassing one simple example reduces the conclusions' external validity. These DSM-based approaches do not seem to provide a scale against which the results can be compared, to find out whether a design is good or bad. Their theoretical validity was not discussed and they reach slightly different conclusions. While in [9] the qualitative claims in [11] are corroborated, in [13] aspects are considered to be detrimental as well.

In [12], a quantitative study on the modularization of the design patterns with aspects is presented. By being grounded on the framework proposed in [15], which is not validated or formalized, there is a threat to the validity of those conclusions. Also, there is an apparent contradiction in [15], when the

Prototype, Command, Iterator, Chain of Responsibility, Strategy and Memento design patterns in AspectJ are ranked among those that present the highest separation of concerns (SoC) and, simultaneously, ranked among those that presented the best coupling and cohesion in their OOP versions, suggesting that to a decrease in modularity corresponds to an increase in the SoC, what sounds inadequate.

From all the above, we can conclude that most OOP versus AOP modularity evaluations made so far are grounded on poor metrics or processes meaning that the validity of their conclusions is threatened.

5 Conclusions

We have applied the M2DM technique, based upon a metamodel that allows paradigm independent representation of software systems (PIMETA) and OCL, to formally define and collect a set of paradigm-independent metrics (PIMETRICS) from the GoF design patterns functionally-equivalent implementations in Java and AspectJ[11]. Those metrics allow quantifying a set of paradigm independent coupling, cohesion and modularity factors, that we dubbed PIFACTORS. Our first conclusion is that is feasible in practice to measure coupling, regardless of paradigm or programming language, by using the PIMETRICS.

We have used statistical techniques to test a set of hypothesis on the modularity improvements of AOP upon OOP. Our main conclusions were the following: (i) The coupling of the coarser grained modules (classes and interfaces) in the OOP version is, on average, 11% higher than that of the coarser grained modules (aspects, classes and interfaces) of the AOP version; (ii) The cohesion of the coarser grained modules in the OOP version is, on average, 20% higher than the cohesion of the coarser grained modules of the AOP version. This decrease in the AOP version is due, in our opinion, to the fact of being compared with the cohesion in the modules of the OOP version influenced by the non-functional cohesion inherent to code tangling, which we named *delusional* cohesion; (iii) The potential crosscutting in the OOP version of the patterns is, on average, 16% higher than the potential crosscutting in the AOP version. This difference evidences a modularity improvement; (iv) The modularity merit factor in the OOP version is, on average, 16% lower than the modularity merit factor in the AOP version. As a whole, we conclude that modularity does improve when OOP systems are refactored to AOP, but that improvement seems to be far more moderate than claimed in related studies [12], [15].

6 Future Work

We aim at enlarging our sample of functionally equivalent systems with more and larger cases. We also are planning to define, at least at an ordinal scale, the dependency types on Java and AspectJ, based upon their impact on modularity. Indirect dependencies will also be studied, as well as the *delusional* cohesion phenomenon. Finnaly, we are developing a tool-supported process to suggest and implement cross-paradigm refactorings from OOP to AOP based on the PIMETRICS capability to identify code-smells [8].

7 Acknowledgments

The work presented herein was partly supported by the VALSE project of the CITI research center within the Department of Informatics at FCT/UNL in Portugal.

References

1. Brito e Abreu, F.: Using ocl to formalize object-oriented design metrics definitions. Technical Report ES007/2001, INESC, Lisbon, Portugal (May 2001)
2. Brito e Abreu, F., Goulão, M.: A merit factor driven approach to the modularization of software systems. L'Objet 7(4) (2001)
3. AspectJ5-Team: Aspectj 5 - quick reference guide (2008)
4. Basili, V., Caldiera, G., Rombach, H.: The goal question metric approach. In: Encyclopedia of Software Engineering, pp. 528–532. John Wiley and Sons (1994)
5. Bravenboer, M.: Aspectj-front parser for aspectj (2008)
6. Bryton, S.: Modularity Improvements with Aspect-Oriented Programming. Msc, Faculty of Sciences and Technology, UNL, Caparica, Portugal (2008), `http://ctp.di.fct.unl.pt/QUASAR/Resources/Papers/2008/bryton2008MSc.pdf`
7. Bryton, S., Brito e Abreu, F.: Pimetrics: A metrics set for paradigm independent modularity evaluation. Technical Report 2007-TR-003, QUASAR Group, Faculty of Sciences and Technology, UNL, Caparica, Portugal (December 2007 2007)
8. Bryton, S., Abreu, F.B.: Modularity-oriented refactoring. In: 12th European Conference on Software Maintenance and Reengineering (CSMR'2008). pp. 23–23. IEEE Computer Society Press, Athens, Greece (Jan 2008)
9. Cai, Y., Huynh, S.: An evolution model for software modularity assessment. In: Boehm, B., Chulani, S., Verner, J., Wong, B. (eds.) Fifth International Workshop on Software Quality (WoSQ'07) @ ICSE'2007. p. 3. Minneapolis, USA (2007)
10. Gogolla, M., Bohling, J., Richters, M.: Validating uml and ocl models in use by automatic snapshot generation. Software and System Modeling 4(4), 386–398 (2005)
11. Hannemann, J., Kiczales, G.: Design pattern implementation in java and aspectj. In: Aldrich, J., Kostadinov, V., Chambers, C. (eds.) OOPSLA02. pp. 161–173. ACM Press, Seattle, Washington, USA (2002)
12. Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., Staa, A.v., Lucena, C.: Quantifying the effects of aspect-oriented programming: A maintenance study. In: International Conference on Software Maintenance (ICSM'06). pp. 223–233 (2006)
13. Lopes, C.V., Bajracharya, S.K.: Assessing aspect modularizations using design structure matrix and net option value. In: Rashid, A., Aksit, M. (eds.) Transactions on Aspect-Oriented Software Development I, Lecture Notes in Computer Science, vol. 3880, pp. 1–35. Springer, Berlin / Heidelberg (2006)
14. OMG: Object constraint language (ocl) (May 2006)
15. Sant'anna, C., Garcia, A., Chavez, C., Lucena, C., v. von Staa, A.: On the reuse and maintenance of aspect-oriented software: An assessment framework. In: Proceedings XVII Brazilian Symposium on Software Engineering (2003), `http://twiki.im.ufba.br/pub/Aside/NossasPublicacoes/sbes2003-135.PDF`