DESIGN QUALITY OF ASPECT-ORIENTED AND OBJECT-ORIENTED PROGRAMS
An Empirical Comparison

Konstantina Georgieva, Ayaz Farooq and Reiner R. Dumke
Inst. of Distributed Sys., Faculty of Computer Science, OvG University Magdeburg
P.O.Box 4120, 39106 Magdeburg, Germany

Keywords: Aspect-oriented programming, Object-oriented programming, Software metrics, Java, AspectJ, Empirical analysis.

Abstract: The aspect-oriented programming introduces the next step in the software design approaches in the sense of overcoming the imperfections in the object-oriented paradigm by separating the cross-cutting concerns and manipulating them in a separate manner. While the use of aspect-orientation attempts to tackle some of the issues with object-oriented approach, it raises some others. How far is the design quality improved by the use of aspect-orientation over object-orientation. What are common design trends among AO programs in comparison with OO programs? A comparison among design metrics for both these kinds of programs can help find answers to these questions. We have used Chidamber & Kemerer metrics suite to empirically compare design properties of AO programs with OO programs. Results from Chidamber & Kemerer metrics for eleven AspectJ benchmark programs and those for dozens of Java libraries and programs are analyzed. The experiment reveals few interesting design trends and some limitations of the used metrics suite.

1 INTRODUCTION

As an another innovation to the hierarchy of programming technologies, aspect-oriented programming (AO) is aimed at enabling better software designs. It introduces clear responsibilities for individual modules, consistent implementation, improved reusability, and improved skill transfer (Laddad, 2003). The new aspect-oriented paradigm is aimed to improve designs over its well-known predecessor, object-oriented programming, by separating the cross-cutting concerns and in this way maintaining better modularity. In this new type of code it is believed that better cohesion, maintainability, reusability and quality are achieved.

Evaluation of aspect-oriented vs. object-oriented programming designs can be studied in different ways. One of these methods is the comparison between AO and OO programs for analyzing the benefits or improvements brought by either of the two technologies. Such comparisons can test validity of the claims that AO programming scores well on some of the quality attributes in comparison to OO programming. It has been observed that empirical research of this kind is yet at an early stage (Guyomarc'h and Guhneuc, 2005).

Within this context, this paper performs an empirical comparison between AO and OO program designs by using the well-known and widely used Chidamber & Kemerer (subsequently referred to as C&K) metrics. This metrics suite was initially defined for object-oriented systems (Chidamber and Kemerer, 1994) and has been extended for aspect-oriented programs (Ceccato and Tonella, 2004). The interpretation and measurement of these metrics for AO and OO programs is certainly slightly different. However, the comparisons between these metrics can still be meaningful since they measure the same property of design in both cases with the same scale. We have used measurement results from a very large source of object-oriented measurements (Farooq et al., 2005) which holds C&K metrics results for several Java standard libraries. The results are compared with the measurement of AO version of the same metrics calculated for a set of programs written in AspectJ.

This experiment's benefit is twofold. On one hand it will help us compare complexity, maintainability, usability and other quality attributes of typical AO and OO programs. Secondly, it will reveal the typical design and implementation trends for

Georgieva K., Farooq A. and R. Dumke R. (2009).
DESIGN QUALITY OF ASPECT-ORIENTED AND OBJECT-ORIENTED PROGRAMS - An Empirical Comparison.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 287-290
DOI: 10.5220/0002237102870290

aspect-oriented programs. This information can also be used for benchmarking among AO programs. A few other works have compared aspect-oriented and object-oriented programs (Tsang et al., 2004), (Guyomarc'h and Guhneuc, 2005), (Kulesza et al., 2006). However, our experiment is based on relatively very large set of measurements which gives strength to generalizability of our results.

2 ASPECT-ORIENTED AND OBJECT-ORIENTED DESIGN METRICS

As already mentioned above, several sets of metrics exist for aspect-oriented programs and even more available for object-oriented programs as well. To be able to make the planned comparisons, we needed a common and resembling set of metrics for both AO and OO programs. Chidamber & Kemerer metrics appeared an excellent choice as two variants of this metrics suite are available for aspect-oriented (Ceccato and Tonella, 2004) and object-oriented programs (Chidamber and Kemerer, 1994). C&K metrics for OO program used in this experiment are *Weighted Method per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Number of Children* (NOC), *Coupling between Objects* (CBO), *Response for a Class* (RFC), and *Lack of Cohesion in Methods* (LCOM). C&K metrics for AO programs used in this experiment are *Weighted Operations in Module* (WOM), *Depth of Inheritance Tree* (DIT), *Number Of Children* (NOC), *Response for a Module* (RFM), *Coupling between Modules* (CBM), and *Lack of Cohesion in Operations* (LCO).

3 THE EXPERIMENT

A very large collection of C&K metrics values for object-oriented programs is already provided by an online measurement repository called OOMJ (available at *http://donner.cs.uni-magdeburg.de:8080/oomj*). This resource is based on measurement of thousands of classes from several Java standard libraries and programs. To calculate C&K metrics for aspect-oriented programs we have used an open source tool called *aopmetrics* (available at *http://aopmetrics.tigris.org*). In our evaluation, we have measured the sample programs located under the *examples* folder of AspectJ 1.6.3 distribution, LoD (found at *http://www.ccs.neu.edu/home/lorenz/papers/aosd2003 lod/*), and TETRIS (found at *http://www.guzzzt.com/coding/aspecttetris.shtml*). The next section summarizes some initial measurements of the measured programs.

Table 1: Size metrics for measured programs.

	Number of classes	Number of methods	Avg. LOC per class
AOP	114	502	35
OOP	14372	140784	642

Table 2: Descriptive statistics: C&K metrics for AO programs.

	Min.	Max.	Mean	Std. Dev.
DIT	0	4	0.63	0.9
NOC	0	3	0.2	0.63
WOM	0	23	4.4	4.68
CBM	0	10	1.62	6.366
RFM	0	235	13.067	2.12
LCO	0	219	10.39	35.99

3.1 Size Metrics

Table 1 gives some size metrics for our experiment. The sample size of object-oriented programs is enormously large which provides stable trends of C&K metrics values for OO programs. Comparatively fewer AO programs could be measured. However, the sample is large enough for any meaningful and representative comparisons between both of the AO and OO paradigms.

4 RESULTS

4.1 Descriptive Statistics

Table 2 shows minimum, maximum, mean, and standard deviation values of C&K metrics for AO programs. A relatively high value of standard deviation for CBM and LCO indicates a high variation among the values of these metrics.

Table 3 compares values of mean and standard deviation for the similar C&K metrics among both types of OO and AO programs. The results show smaller averages for number of operations per class (WOM), response for class (RFM), and lack of cohesion (LCO) values for aspect-oriented programs. The rest of the metrics show almost similar trends. Furthermore, the smaller standard deviations for almost all of the AO

Table 3: Comparison between the C&K metrics for OO and AO programs

	Mean		Std. Deviation	
Metric	OO	AO	OO	AO
DIT	0.7	0.6	1.0	0.9
NOC	0.4	0.2	2.0	0.6
WMC/WOM	11.9	4.4	15.8	4.7
CBO/CBM	1.9	1.6	3.9	6.4
RFC	19.5	13.1	28.6	2.0
LCOM/LCO	139.4	10.4	802	36.0

Table 4: 95% Confidence interval: mean of C&K metrics for AO and OO programs.

Metric	OOP (lower/upper limit)	AOP (lower/upper limit)
DIT	0.57–0.61	0.47–0.80
NOC	0.14–0.16	0.09–0.33
WMC/WOM	8.54–8.85	3.54–5.26
CBO/CBM	2.92–3.09	1.23–2.01
RFC/RFM	14.45–15.10	4.47–7.20
LCOM/LCO	35.88–39.14	3.79–17.00

metrics make these averages more meaningful and reliable.

Sometimes, mean value of an entity may be misleading specially when there is a very large variation among the values. For a further deeper analysis of the mean values shown in table 3 we calculated confidence intervals for each of the metrics. Table 4 shows confidence intervals at 95% level for each of the metrics.

To give another orientation to the aforementioned statistics, figure 1 presents graphical view of the OO and AO metrics distributions. Metrics values are drawn on the horizontal axis while on the vertical axis are the frequencies of these values.

4.2 Measurement Analysis

General observations: The first comment is about the overall range of metric values. Table 3 indicates a smaller standard deviation for all of AO metrics as compared to their OO counterparts. This is due to the fact that most metrics values fall within a narrow range with very few outliers. Figure 1 shows frequency distributions for these metrics which corroborate this fact. This makes the results of table 4 more representative and reliable which is aimed to tell us a most probable value for each of these metrics.

WMC/WOM: A small WMC/WOM is considered a good design practice which is believed to reduce complexity. Separation of concerns in AO designs

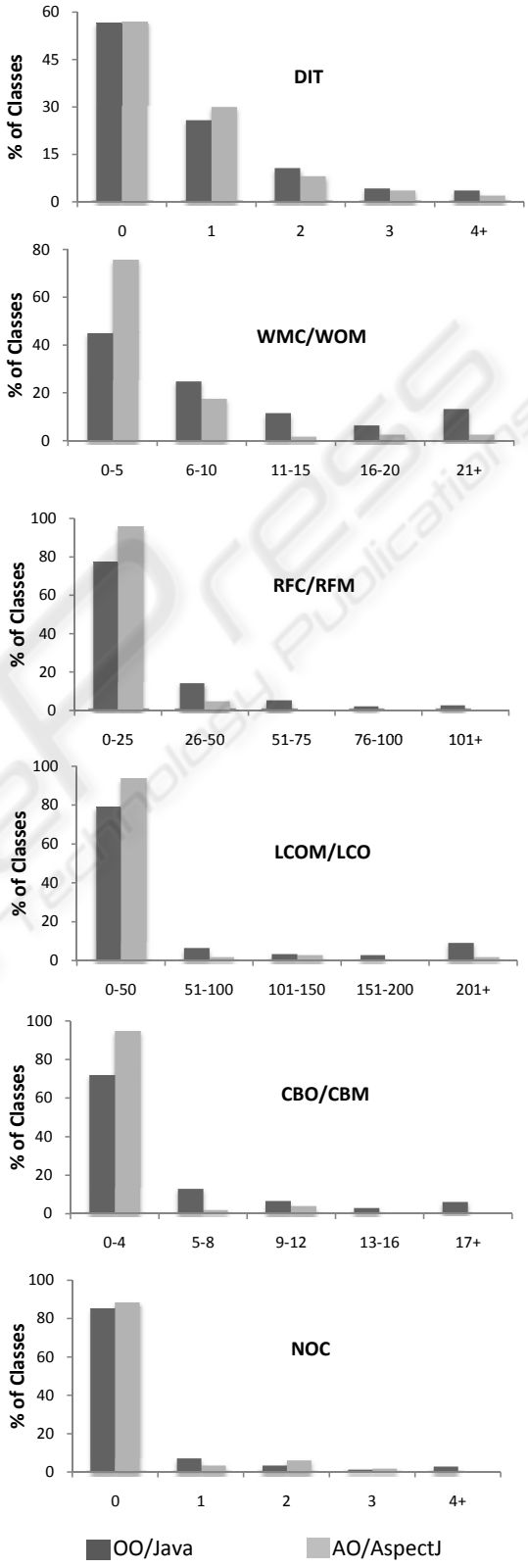

Figure 1: C&K metrics distributions for AO and OO programs.

should reduce the size of operations. This is clearly seen by a smaller value of this metric observed in the performed measurements.

DIT/NOC: Both these metrics as indicators of inheritance show slightly smaller values for AO programs vs. OO programs. Based on their characteristics, these metrics should not be much affected with the change of paradigm from OO to AO. Perhaps that is the reason that these metrics do not show any significant changes in program design strategies.

Coupling related measures, to be discussed next, are more important for analysis since aspect-orientation mainly concerns with this issue.

RFM: This metric, similar to RFC of C&K metrics suite, is a kind of coupling measure. Both the figure 1 and the tables 3 and 4 show that its value is between 0-25 in about 95% of the cases. This metric reflects reduced coupling for AO programs. It shows a considerable design improvement of the AOP over the OOP.

CBO/CBM: CBM of AO is the closest parallel to the CBO of OO programs. It is perhaps the most important measure characterizing the couplings which is the main motivation for the paradigm shift from OOP to AOP. The figure 1 and table 3 show considerably reduced coupling between modules. This seems to be a consequence of AO's separation of concerns and woven code.

LCO: In an OO or AO design, there should be high cohesion between methods/operations. Non-cohesive operations indicate a need for splitting classes/aspects. Despite being in the range of 0-50, a very high variation of this metric was observed for object-oriented programs. However, in case of AO, it shows a very small average with a very low value of standard deviation, see table 3. Table 4 shows that its value could be between 3 and 17. A higher majority of measured AO programs have showed a very high cohesion which reflects an improved design practice.

5 CONCLUSIONS

This paper presented a metrics-based comparison among object-oriented and aspect-oriented designs using several samples of OO/AO libraries and programs. Amplitude of the OO measurement data made this empirical study unique among the very few experiments of this kind ever performed. From the programs and libraries that we measured, it was found that both kinds of programs exhibited similar inheritance related characteristics. Small coupling and high cohesion was observed in the measured AO programs w.r.t OO programs. Based on these quality indicators we were able to confirm the thesis that aspect-orientation improves modularity and maintainability. Overall, AO reflected (usually considered) better design characteristics in contrast to OO programs.

A very large proportion of measured OO programs consisted of standard libraries, and not normal programs/applications. The study needs to be extended with a larger and diverse sample of both real OO and AO programs using more coupling related and other relevant metrics.

REFERENCES

Ceccato, M. and Tonella, P. (2004). Measuring the effects of software aspectization. In *Proceedings of 1st Workshop on Aspect Reverse Engineering*.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object-oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493.

Farooq, A., Braungarten, R., and Dumke, R. R. (2005). An empirical analysis of object-oriented metrics for Java technologies. In *INMIC 2005: Proceedings of the 9th IEEE International Multitopic Conference*, pages 548–553. IEEE Computer Society.

Guyomarc'h, J.-Y. and Guhneuc, Y.-G. (2005). On the impact of aspect-oriented programming on object-oriented metrics. In *Proceedings of the 9th ECOOP workshop on Quantitative Approaches in Object-Oriented Software Engineering*, pages 42–47. Springer-Verlag.

Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., von Arndt Staa, and Lucena, C. (2006). Quantifying the effects of aspect-oriented programming: A maintenance study. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 223–233. IEEE Computer Society.

Laddad, R. (2003). Aspect-oriented programming will improve quality. *Software, IEEE*, 20(6):90–91.

Tsang, S. L., Clarke, S., and Baniassad, E. (2004). Object metrics for aspect systems: Limiting empirical inference based on modularity. Technical report, Trinity College Dublin, Ireland. Available at http://www.cse.cuhk.edu.hk/ẽlisa/papers/OO-AOMetrics.pdf.