

The Effect of Design Patterns on Aspect Oriented Software Quality—An Empirical Evaluation

Sirbi Kotrappa^{1,*} and Kulkarni Jayant Prakash²

¹ Department of Computer Science & Engineering,
K L E's College of Engineering & Technology, Belgaum, India

² Department of Computer Science & Engineering,
Walchand College of Engineering, Sangli, India
kotrappa06@gmail.com, pjk_walchand@rediffmail.com

Abstract. In recent times, software engineers attempted to measure software quality using various approaches and techniques such as metric suites. Aspect oriented programming (AOP) a new technology addressing issues of scattering and tangling of code spread throughout the system. Today, Aspect oriented programming (AOP) is gaining wide attention both in industry and research. OO DPs (design patterns) have difficulties in implementation of crosscutting concerns because of lack of features in OO languages and crosscutting concerns affecting software quality. In this paper, we evaluate metrics of OO Vs AO DPs for separation of concern, size, coupling and cohesion metrics from the C & K metric suite, which was modified to AOP. This empirical evaluation provides a new knowledge about AOP software quality and software developer can adopt in a specific situation. We claim that the AOP has significance effect on design quality than OOP.

Keywords: Aspect oriented programming (AOP), design patterns (DPs), software metrics, quality model.

1 Introduction

When a novel method is planned, this has to provide evidences for its supremacy over presented participants. AOP emerged as a new technology to improve software quality attributes whose implementations would otherwise have been spread throughout the whole application because of the limited abstractions of the underlying programming languages. Since then several studies [3], [6], [7], [8], [9],[10], [11], [12] have suggested that AOP is successful in improving software quality crosscutting concerns. But these studies either provide strong evidences for better software quality offered by AOP or wrongly measure metrics of AO systems. We found in some cases in which an AO implementation was better quality than its OO DPs counterpart.

* Corresponding author.

Many OO DPs methods have been adapted to indicate and employ DPs effectively. But several DPs that impact system quality, when core objects are especially affect by the formation that the DPs needs .AOP AspectJ[24] will help us on separating few of the system’s DPs, mentioning and implementing those as a single units of abstraction. Here our objective is to show the OO Vs AO DPs implementation of observer pattern and its effect on software quality [11]. The software developer who wants to apply AOP for implementation of design patterns crosscutting concern will be benefited. Many researchers written at length on the nature of aspect oriented programming [4], [6], [7], [10],[11], [13], [14],[15],[16],[17],[18],[19],[20],[21]. AOP has evolved as a technology for combining separately created software components into working systems. It requires new assessment frameworks specifically tailored to measure the evolution, reusability, security and maintainability of aspect-oriented systems. Our results show that it is possible to use standard Object Oriented Programming (OOP) quality metrics to measure the advantages of AOP, even after adaptation [4]. But very few existing evaluations have been performed at qualitative and quantitative levels in AOP [8]. This paper presents an evaluation of effect of observer pattern OO DPs on AOP quality metrics, which is composed of three components: G-Q-M model, a suite of metrics and a quality model.

2 Measurement Process and a Quality Model

To provide comparison between OO DPs Vs AO DPs of software quality, we can apply the ISO/IEC 9126-1 quality model and Goal-Question-Metric (G-Q-M) [12]. G-Q-M defines a measurement system on three levels (Figure. 1) starting with a goal. The goal is refined in questions that break down the issue into quantifiable components. Each question is associated with metrics that, when measured, will provide information to answer the question. Our goal is to compare AOP and OOP systems with respect to software quality from the viewpoint of the software developer.

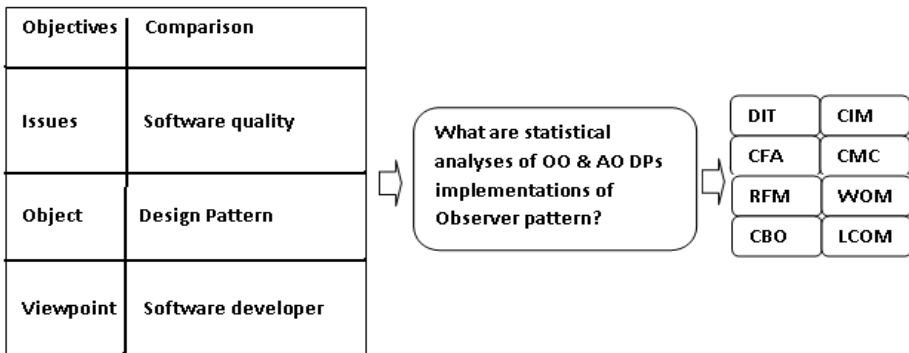


Fig. 1. G-Q-M Model

Our quality model defines a terminology and clarifies the relationships between the reusability, maintainability and the metrics suite. It is a useful tool for guiding software developers in data interpretation. The quality model has been built and refined using Basili's GQM methodology [12] (see Figure 2). The metrics are comparable because they both measure properties of concerns at the class and aspect level (see Table 1 and Figure 2).

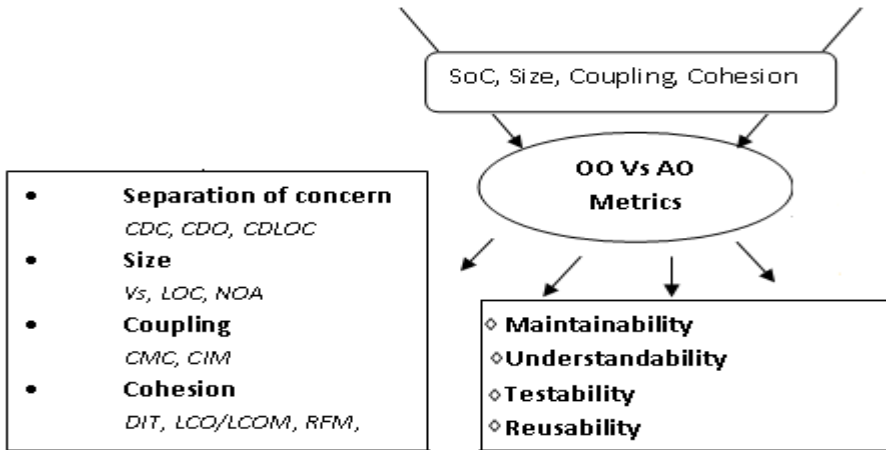


Fig. 2. Association of software quality attributes and their metrics

Table 1. Software quality metrics definitions

Metrics	Definition
RFC/ RFM	Number of methods and advices potentially executed in response to a message received by a given module
WOM/ WMC	Number of weighted operation in a given module
CBO/ CBM	Number of modules or interfaces declaring methods or fields that can be called or accessed by a given module
L COM/ LCO	Number of pairs of operations working on different class fields minus pairs of operations working on common fields
DIT	Length of the longest path from a given module to the class/aspect hierarchy root
CIM	Number of modules or interfaces explicitly named in the pointcuts of a given aspect
CFA	Number of interfaces or modules declaring fields that are necessary by a given module
CMC	Number of modules or interfaces declaring methods that can be called by a given module

3 Empirical Evaluation

This empirical evaluation uses implementations of the 23 GoF design patterns made freely available by Hannemann & Kiczales [9], [25]. In [9] every pattern explained with an example that uses pattern, which are implemented both in OO DPs (Java) and AO DPs (AspectJ). Observer pattern, known as Model-View is indented to “define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically”. OO DPs Observer pattern implementation generally add a field to every Subjects that stores a record of Observers attracted in that exact Subject. Whenever Subject report state change to its Observers, it calls its own *notify* method, which in turn calls an *update* method on all Observers in the record. The sample OO DPs Observer patterns shown in figure 3 and AspectJ [24] AO DPs shown in figure 4 in the perspective of a trivial figure example [9]. In such a system the Observer pattern is used to cause mutating operations to figure elements to update the screen. The code spread across the all classes in this pattern. This method includes code which is important to adopt example of such pattern. Every member i.e., Point and Line has to be familiar with about their responsibility in the pattern and accordingly have pattern code in them. Addition or deletion needs corresponding changes in that particular class [9]. Changing the notification mechanism requires changes in all participating classes. In the AspectJ version [9] all code pertaining to the relationship between Observers and Subjects is moved into an aspect, which changes the dependencies between the modules. Subject and Observer roles crosscut classes, and the changes of interest (the *subjectChange* pointcut) crosscuts methods in various classes. In this paper, we have decided to assess the implementation of observer design patterns in both Java and AspectJ. First, we applied the metrics in Hannemann and Kiczales original code [9]. Afterwards, we changed their implementation to add new participant classes to play pattern roles.

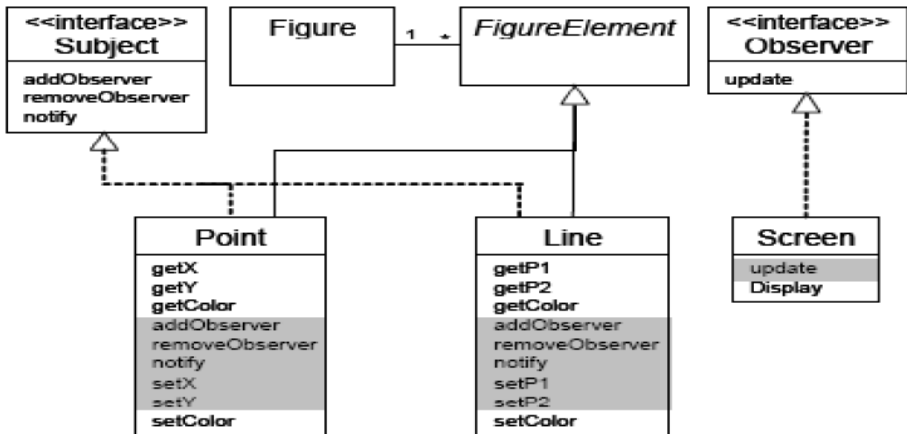


Fig. 3. Observer pattern-Java

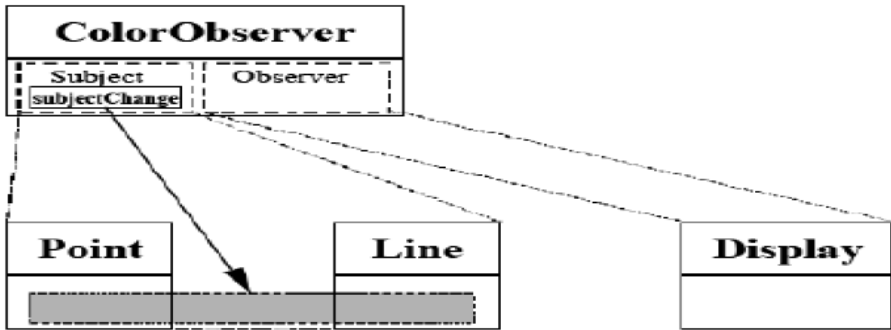


Fig. 4. Observer pattern-AspectJ

These changes were introduced because Hannemann and Kiczales' implementation encompasses few classes per role (in most cases only one) [9]. Hence we have decided to add more participant classes in order to investigate the pattern crosscutting structure. Finally, we have applied the selected metrics to the changed code. We analyzed the results after the changes, comparing with the results gathered from the original code (i.e. before the changes) [9].

4 Software Quality Assessments through OO and AO Metrics

In this paper authors have evaluated the popular C & K metrics [2] on effect of DPs on quality when we change system from OOP to AOP. For an empirical evaluation OO Vs AO metrics, we considered the simple observer patterns DPs to determine the effect of quality attributes on OO Vs AO DPs. In the evaluation of metrics authors have used the AOPMetrics tool [23] to measure the quality metrics related to separation of concern, size, coupling and cohesion. The goal of the AOPMetrics project is to provide a common metrics tool for the object-oriented and the aspect-oriented programming [24]. The project aims to provide CK metrics, Robert Martin's and Henry and Li metrics suite [23]. Table 2 gives total, mean, maximum, and minimum and standard deviation statistical values of OO DPs and AO observer pattern DPs C & K metrics.

5 Descriptive Statistics

Table 2 gives descriptive statistics of OOP and AOP C & K metrics for observer patterns. Also it compares values of total, mean, maximum, standard deviation for both OO and AO observer pattern. Here we present the measurement results for the observer design patterns; we focus on the presentation of results related to RFC, CBM, LCO and WMC from the C&K metric suite and their effects on software quality (see table 2). The relatively high value of standard deviation for CBM and LCO indicates a high variation among the values of these metrics. The results shows

that smaller average for number of operations per class (WOM (WMC)), response for class (RFM (RFC)), coupling between objects (CBO (CBM)) and lack of Cohesion (LCO (LCOM)) values for AOP observer patterns. The rest of the metrics shows almost same trends. Additionally, low standard deviation for almost all of the AOP metrics make these averages more meaningful and consistent. At times, mean value of an entity may be misleading particularly when there is a very large variation among the values.

Firstly general observation is that the overall quality of OO and AO metrics values. Table 2 indicates smaller variations of standard deviation for all of AO metrics as compared to their OO version. This is because of the reason that almost all metrics values fall in line a small range with very small outliers.

Table 2. OO Vs AO Observer pattern DPs Software Quality Metrics

	RFC/ RFM		WOM/ WMC		CBO/ CBM		LCOM/ LCO		DIT		CIM/ CAE		CFA		CMC	
	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO	OO	AO
Total	21	28	21	18	5	8	28	19	0	3	0	6	0	0	5	8
Average	4.2	3.1	4.2	2	1	0.9	5.6	2.1	0	0.3	0	0.6	0	0	1	0.8
Max	10	8	10	7	2	5	24	10	0	1	0	1	0	0	2	5
Min	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Std.Dev	4.2	2.7	3.8	2.4	1	1.6	10	4.2	0	0.5	0	0.5	0	0	1	1.6
Quality Attributes	Understandability		Testability		Complexity		Maintainability		Reusability		Maintainability		Complexity		Reusability	
	Testability		Testability		Complexity		Maintainability		Reusability		Testability		Reusability		Testability	
	Testability		Complexity		Maintainability		Reusability		Testability		Reusability		Reusability		Testability	
	Complexity		Maintainability		Reusability		Maintainability		Complexity		Reusability		Reusability		Testability	
	Maintainability		Reusability		Maintainability		Complexity		Reusability		Testability		Reusability		Reusability	
	Complexity		Maintainability		Reusability		Maintainability		Complexity		Reusability		Reusability		Testability	
	Maintainability		Reusability		Maintainability		Complexity		Reusability		Testability		Reusability		Reusability	
	Complexity		Maintainability		Reusability		Maintainability		Complexity		Reusability		Reusability		Testability	
	Maintainability		Reusability		Maintainability		Complexity		Reusability		Testability		Reusability		Reusability	

6 Results and Discussions

6.1 Results

Here we present the empirical results for the observer patterns; we focus on the presentation of results related to quality attributes metrics i.e., separation of concern, size, coupling and cohesion.

6.1.1 SoC: Separation of Concerns

The use of aspects clearly provided better support for separation of observer pattern concerns [4]. This result is supported by all separation of concern metrics. The results shows that CDC measurement identifies each observer pattern concern need many components for their implementation in the OO solution as compared to the AO

solution. In addition, all concerns required more operations (methods/advices) in the OO system than in the AO system (CDO metric). Finally, the CDLOC measures also pointed out that the AO solution was more effective in terms of modularizing the observer pattern concerns across the lines of code. The resulting metrics present the gathered data *before* and *after* the changes applied to the pattern implementation. These metrics support an analysis of how the introduction of new classes and aspects affect both solutions with respect to the selected metrics. Those changes also allow us to understand which solution is better to assist the modularization quality of concerns from the application and the pattern points of view. For separation of concerns, we have verified the separation of each role of the patterns on the basis of the three separations of concerns metrics [4], [6].

6.1.2 Coupling, Cohesion and Size

These results show that for the Observer patterns, the AO DPs implementation apparently has much more associated profit. As the changes were accomplished, the AO solution exhibited superior results with respect to DIT, RFC, CBM/CBO, LCO, CIM, CFA, CMC and WMC. The differences were typically higher in favor of the aspect-oriented solution for both OO and AO observer design patterns. The aspect-oriented project produced a more concise system according the number of lines of code. However, the use of aspects produced more complex operations, i.e. advices, than the use of the OO patterns (WOC metric) [4]. The AO system incorporated components with higher coupling (CBC metric). The OO DPs has led to the abuse of the inheritance mechanism, which was fundamental for establishing high inheritance coupling (DIT metric) [4], [6]. The LCOO metric detected some components of the OO system and produced better results in terms of cohesion than the components of the AO system. In the aspect-oriented implementation of this pattern, the major improvements were detected in the LOC, LCOO and NOA measures [4]. The use of aspects led to reduction of LOC in relation to the OO code. Thus aspects solve the problem of code replication related to the implementations of the method `notifyObservers()`. The cohesion in the AO implementation was mostly higher than the OO implementation because the latter incorporates a number of classes that play the Subject and Observer roles and, as a consequence, implement role-specific methods that in turn do not access the attributes of the classes. In the aspect-oriented design, these methods are localized in the aspects that implement the roles, increasing the cohesion of both classes and aspects. The tally of attributes in the OO implementation was respectively higher than in the AO code before and after the introduction of new components into the implementations [4]. In the OO solution, the “subject” classes need attributes to hold references to their “observer” classes; these attributes are not required in the aspect-oriented design [4].

6.2 Discussion

Based on the results, we have observed that the measures relative to quality metrics DIT, RFC, CBM/CBO, LCO, CIM, CFA, CMC and WMC. In general, the AO solutions were superior in terms of quality measures, since the use of AspectJ reduces

the overuse of inheritance mechanisms. However, as illustrated in table 2, most measures indicated that AspectJ implementations resulted in higher coupling (CBC) and more lines of code (LOC) than the respective Java implementations. The superiority of AspectJ is CDO, CDC, and CDLOC metrics measures were higher than OO DPs for observer pattern. The AO DPs solutions were better-quality in terms of DIT, RFC, CBM/CBO, LCO, CIM, CFA, CMC and WMC measures. However, in many measurements point to those AspectJ implementations provides in upper coupling (CBC) and extra lines of code (LOC) than particular java implementation. However, a careful analysis of the implementation show that these higher CBC and LOC values for AO solutions in general are related to presence of generic aspects in several AspectJ pattern implementations, which have the intension of making the solution more reusable. We claim that the AOP has significance effect on design quality than OOP.

7 Related Works

As new software development method evolved, it is essential that empirical studies are carried out to provide evidences about benefits to the software developer [10],[22]. Software metrics provide quality indicators of software development. The AOSD community has been developing significant work on quantitative and qualitative analysis of AOP software [22]. Sant'Anna et al. provided one of the exceptional research works on AOP metrics: a suite of metrics for quantifying modularity-related attributes, published in a SBES paper [14], [18], [20]. This metric suite includes coupling, cohesion and size metrics custom-made from existing OO metrics to deal with AOP quality measures [22]. A set of existing metrics has been used to evaluate the quality of different AOP implementations [6], [7], [14].The metric suite also includes novel concern-driven metrics, aimed at quantifying unusual aspect of separation of concerns [22]. Concern-driven metrics endorse the notion of concern as a measurement abstraction. This kind of metric relies on the identification and documentation of the pieces of source code that implement each concern of the system. Experimental studies [14], [15], [16] measured up to the quality of Java and AspectJ solutions of the 23 design patterns from the Gang of Four [22]. One more study methodically examined how AOP degrees up to treaty with modularization of 23 design patterns in the occurrence of pattern interfaces [26]. The investigator completed both qualitative and quantitative evaluation of 62 pair-wise compositions of patterns taken from 3 medium-sized systems implemented in Java and AspectJ programming languages. Kulesza et al. [17] presented an empirical study in which they quantified the effects of AOP in the maintainability of a web-based information system. Figueiredo et al. [13] carry out an experimental study for appraise whether AOP endorse better quality and changeability of product lines than conventional variability mechanisms, such as conditional compilation [22].

8 Concluding Remarks

In this paper, we investigate AOP and DPs effect on software quality. Our evaluation is based on measurements on parameters of software quality attributes of separation of concern, size, coupling and cohesion metrics from the C&K metric suite, which was modified to AOP. An approach to re-implement OO observer pattern DPs by AOP AspectJ is presented in this paper and analyzed for its software quality factors. In this paper, we were focused on the evaluation of OO Vs AO DPs Observer Pattern and software quality. We claim that the AOP has significance effect on design quality than OOP.

Acknowledgments. We place on records and wish to thank the authors Maria Luca Bernardi, Giuseppe Antonio Di Lucca, RCOST Research centre on Software Technology, University of Sannio, Palazzo ex Poste, Benevento, Italy for their valuable contributions to research on design pattern quality using Aspect orientation.

References

1. Fernando, C., Neilo, C., Eduardo, F.: On the modularization and reuse of exception handling with aspects. *Softw. Pract. Exper.* 39(17), 1377–1417 (2009)
2. Shyam, R.C., Chris, F.K.: A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.* 20(6), 476–493 (1994)
3. Lech, M., Lukasz, S.: Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study. *IET Software Journal* 1(5), 180–187 (2007)
4. Claudio, N.S.A., Alessandra, F.G., Uira, K.: Design patterns as aspects: a quantitative assessment. *Journal of the Brazilian Computer Society* 10(2) (November 2004) ISSN 0104-6500
5. Erich, G.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading (1995)
6. Adam, P.: An empirical assessment of the impact of AOP on software modularity. In: 5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2010), Athens, Greece (2010)
7. Adam, P.: What is wrong with AOP? In: 5th International Conference on Software and Data Technologies (ICSOF 2010), Athens, Greece (2010)
8. Avadhesh, K., Grover, P.S., Rajesh, K.: A quantitative evaluation of aspect-oriented software quality model AOSQUAMO. *ACM SIGSOFT Software* (2009); Workshop on Emerging Trends in Software Metrics, WETSoM (2010)
9. Jan, H., Gregor, K.: Design Pattern Implementation in Java and AspectJ. In: *Proceedings of OOPSLA 2002*, pp. 161–173 (November 2002)
10. Mariano, C., Paolo, T.: Measuring the Effects of Software Aspectization. In: 1st Workshop on Aspect Reverse Engineering, Delft, Netherlands (2004)
11. Mario, L.B., Giuseppe, A.D.L.: Improving Design Pattern Quality Using Aspect Orientation Software Technology and Engineering Practice. In: 13th IEEE International Workshop, September 24-25 (2005)

12. Victor, R.B., Gianluigi, C.H., Dieter, R.: The Goal Question Metric Approach. In: Encyclopedia of Soft. Eng., vol. 2, pp. 528–532. John Wiley & Sons, Inc. (1994)
13. Figueiredo, E., et al.: On the maintainability of aspect-oriented software: A concern-oriented measurement framework. In: CSMR, pp. 183–192 (2008)
14. Sant’Anna, C., et al.: Design Patterns as Aspects: A Quantitative Assessment. In: SBES (2004)
15. Garcia, A., et al.: Modularizing design patterns with aspects: a quantitative study. In: AOSD 2005, pp. 3–14 (2005)
16. Garcia, A., et al.: Modularizing design patterns with aspects: a quantitative study. Trans. on AOSD I, 36–74 (2006)
17. Kulesza, U., et al.: Quantifying the effects of aspect-oriented programming: A maintenance study. In: ICSM, pp. 223–233 (2006)
18. Sant’Anna, C.: On the modularity of aspect-oriented design: A concern-driven measurement approach, Ph.D. dissertation, PUC-Rio (2008)
19. Greenwood, P., Bartolomei, T., Figueiredo, E., Dosea, M., Garcia, A., Cacho, N., Sant’Anna, C., Soares, S., Borba, P., Kulesza, U., Rashid, A.: On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. In: Bateni, M. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 176–200. Springer, Heidelberg (2007)
20. Sant’Anna, C., et al.: On the modularity assessment of aspect-oriented multiagent architectures: a quantitative study. IJAOSE 2, 34–61 (2008)
21. Silva, B., et al.: Concern-based cohesion as change proneness indicator: an initial empirical study. In: ICSE WETSoM, pp. 52–58 (2011)
22. Chavez, C., Kuleszay, U., et al.: The AOSD Research Community in Brazil and its Crosscutting Impact, AOSD-BR (2011)
23. AOPMetrics Project home, <http://aopmetrics.tigris.org>
24. The AspectJ project, <http://www.eclipse.org/aspectj>
25. Jan, H.: Design Patterns, <http://hannemann.pbworks.com/Design-Patterns>