IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 233-237

Design Patterns Vs Aspect Oriented Programming – A Qualitative and a Quantitative Assessment

Kotrappa Sirbi[1] & Prakash Jayanth Kulkarni[2]

[1]Department of Computer Science & Engineering, K.L.E's Dr.M.S.Sheshagiri College of Engineering & Technology, Belgaum, India
[2]Department of Computer Science & Engineering, Walchand College of Engineering, Sangli, India
Email: [1]kotrappa06@gmail.com, [2]pjk_walchand@rediffmail.com

ABSTRACT

Design patterns offer flexible solutions to common problems in software development. Recent studies have shown that several design patterns involve crosscutting concerns. Unfortunately, object-oriented (OO) abstractions are often not able to modularize those crosscutting concerns, which in turn decrease the system reusability and maintainability. Hence, it is important to verifying whether aspect-oriented (AO) approaches support improved modularization of crosscutting concerns relative to design patterns. Ideally, quantitative studies should be performed to compare OO and AO implementations of classical patterns with respect to important software engineering attributes, such as coupling, cohesion and size. Our assessment is based on OO Design Patterns (DPs) Vs AO techniques in terms of a qualitative and a quantitative level, by identifying the additional design implications needed to perform the extension and evaluating the effect of the extension on several qualities attributes of the application. In this paper, we present a quantitative and a qualitative assessment of Design Patterns (DPs) and AOP implementations of the GoF patterns.

Keywords: Object-oriented (OO), Design Patterns (DPs), Aspect-Oriented Programming (AOP), Software Metrics

1. MOTIVATION AND BACKGROUND

1.1. Hannemann and Kiczales' Study

Several design patterns exhibit crosscutting concerns [9]. In this context, Hannemann and Kiczales (HK) have undertaken a study in which they have developed and compared Java [11] and AspectJ [2] implementations of the 23 GoF design patterns [5].They claim that programming languages affect pattern implementation. Hence it is natural to explore the effect of aspect- oriented programming techniques on the implementation of the GoF patterns. For each of the 23 GoF patterns they developed a representative example that makes use of the pattern, and implemented the example in both Java and AspectJ.Design patterns assign roles to their participants.A number of GoF patterns involve crosscutting structures in the relationship between roles and classes in each instance of the pattern [9].Two kinds of pattern roles are identified in the HK study, which are called defining and superimposed roles. A defining role defines a participant class completely. In other words, classes playing a defining role have no functionality outside the pattern. The unique role of the Façade pattern is an example of defining role. A superimposed role can be assigned to participant classes that have functionality outside of the pattern. An example of superimposed role is the Colleague role of the Mediator pattern, since a participant class playing this role usually has functionality not related to the pattern. These kinds of roles are used by the authors to analyze the crosscutting structure of design patterns. In the HK study, the goal of the AspectJ implementations is to modularize the pattern roles.

1.2. Background

One of the first case studies was conducted by Kersten and Murphy [12]. They have built a web-based learning system using AspectJ. In this study, they have discussed the effect of aspects on their object-oriented practices and described some rules and policies they employed to achieve their goals of modifiability and maintainability using aspects. Since several design patterns were used in the design of the system, they have considered which of them should be expressed as classes and which should be expressed as aspects. They have found that Builder, Composite, Façade, and Strategy patterns [13] were more easily expressed as classes, once these patterns were had little or no crosscutting properties. Soares et al [14] have reported their experience using AspectJ to implement distribution and persistence aspects in a web-based information system. They have implemented the system in Java and restructured it with AspectJ. They have argued that the AspectJ implementation of the system bring significant advantages with the corresponding pure Java implementation. Walker et al. [15] have conducted two exploratory experiments to study the increased

modularization provided by AspectJ. In these experiments, they have compared the performance of a small number of participants working on two common programming tasks: debugging and changing. However, these studies are qualitative assessments, which are not focused on the use of aspects for modularizing pattern-related concerns. Object Oriented Design Patterns provide the design of generic solutions to recurring problems [1]. The usage of DPs would increase the quality of OO design, but often developers miss the subtleties of the DP consequences [1] by applying/ implementing patterns with wrong variation points to add (unneeded) flexibility. In these cases the usage of such DPs negatively affects the system overall quality. However, also when the consequences are well taken into account, DPs may introduce in OO systems some problems related to reuse and maintainability that can be hard to solve within the object oriented paradigm. Usually, these problems are due to the indirection techniques forcing one or more key interfaces to be implemented introducing a greater overhead. This issue is dealt in [5] where is shown how AOP can help in restructuring GoF patterns to get more effective anticipation of future changes with a lower overhead. Recent researches have shown that many DPs involve Cross-cutting Concerns (CCCs).

That is mainly due to the poorness of the composition and quantification constructs in OOP languages that do not allow a good modularization of the concerns. Indeed, by using OO DPs, programmers are forced to add classes, interfaces, methods and attributes inside the code of the components (i.e. classes, packages, etc.) derived from the primary decomposition. The introduction of these elements will produce code scattering and tangling by reducing the quality of some DP attributes such as reusability, traceability, comprehensibility, maintainability.AOP constructs are able to better modularize DP concerns. In [2, 4, 6] AOP implementations of GoF [2] patterns are provided; they have better values for properties such as locality, (un)pluggability, composability and reusability. Zhao and Xu [17, 18] have proposed new cohesion measures that consider the peculiarities of the AO abstractions and mechanisms. Their metrics are based on a dependence model for AO software that consists of a group of dependence graphs. The authors have shown that their measures satisfy some properties that good measures should have. However, these metrics have not yet been validated or applied to the assessment of realistic AO systems.

2. AOP ASPECTJ

AspectJ is an aspect-oriented extension of Java. It introduces, like other aspect implementations, several new abstractions such as aspects, join points, point- cuts, advices, as well as introductions. These various mechanisms allow the expression of concerns that crosscut several basic modules. A join point represents well-defined points in the program flow, such as method calls and field sets. Pointcuts describe a set of join points and the context to expose. Advice is a method-like abstraction that defines code to be executed when a join point is reached. Pointcuts are used in the definition of an advice. An introduction defines how AspectJ modifies a program's static structure, namely, the members and the relationship between components. Pointcuts and advice dynamically affect program flow. Introduction statically affects a program's class hierarchy.

2.1. Quality of Aspect-Oriented Systems

Aspect-oriented software development introduces new abstractions to software engineering. For that reason, existing object metrics cannot be used directly. Presently, only few papers address aspect-oriented programs quality has been published in the literature. Zhao and Xu's approach [19, 20] is the first proposal in the field of aspect cohesion and coupling measurement.

Table 1
AO Metrics Suite

Attributes	Metrics	Definitions
Separation of Concerns	Concern Diffusion over Components (CDC)	Counts the number of classes and aspects whose main purpose is to contribute to the implementation of a concern and the number of other classes and aspects that access them.
	Concern Diffusion over Operations (CDO)	Counts the number of methods and advices whose main purpose is to contribute to the implementation of a concern and the number of other methods and advices that access them.
	Concern Diffusions over LOC (CDLOC)	Counts the number of transition points for each concern through the lines of code. Transition points are points in the code where there is a "concern switch".
Coupling	Coupling Between Components (CBC)	Counts the number of other classes and aspects to which a class or an aspect is coupled.
	Depth Inheritance Tree (DIT)	Counts how far down in the inheritance hierarchy a class or aspect is declared.
Cohesion	Lack of Cohesion in Operations (LCOO)	Measures the lack of cohesion of a class or an aspect in terms of the amount of method and advice pairs that do not access the same instance variable.
Size	Lines of Code (LOC)	Counts the lines of code.
	Number of Attributes(NOA)	Counts the number of attributes of each class or aspect.
	Weighted Operations per Component (WOC)	Counts the number of methods and advices of each class or aspect and the number of its parameters.

It is based on a dependency model for aspect-oriented software that consists of a group of dependency graphs. Ceccato and Tonella [21] proposed an aspect revision of Chidamber and Kemerer's metrics suite [22]. Sant'Anna et al. also proposed a metric suite for AOP in [23]. Zhang and Jacobsen [25] used the cyclomatic complexity number, size, weight of class, coupling between objects and response time for their evaluation. Tsang, Clarke and Baniassad [26] applied the C&K metrics suite in their evaluation of aspect oriented techniques. They focused on the quality factors understandability, maintainability, reusability and testability, and the C&K definitions of metrics suited for measuring these factors as mention in Table1.

2. 2. AOP Quality Metrics

Trying to define software quality is a challenging endeavor. For some, software quality is defined as strict conformance to requirements and absence of bugs, while for others; it is mainly an aesthetic issue. Over the past decades, software engineers attempted to measure quality using various approaches and techniques such as metric suites. They were motivated by the fact that you cannot control what you cannot measure. Aspect-oriented software development emerges as a promising new software engineering paradigm [30]. However, it has not reached yet its full maturity. Consequently, few rules or heuristics have been defined addressing quality assessment of aspect-oriented software. Existing object-oriented programming languages suffer from a serious limitation in modularizing adequately crosscutting concerns [29, 28]. Crosscutting goes beyond hierarchy as stated by Kiczales [27]. The code related to a crosscutting concern is generally duplicated within several classes in an object-oriented system. Consequently, these classes are difficult to understand, maintain and reuse. Aspect-Oriented Programming (AOP) deals with scattered and tangled code related to crosscutting concerns. It promotes improved separation of crosscutting concerns into single units called aspects. Even though this new programming technique seems to provide concrete answers to many object-oriented software defects, software engineering principles cannot be ignored. Aspect-oriented software quality has to be evaluated and better tools must be developed to support developers and validate the common intuition that aspect-oriented programming is a good solution. These tools must be developed in order to measure and control the new software complexity introduced by aspects.

2.3. Using AO to Improve DP Modularity

The modularity of DP implementation can be improved by re-designing DP by AO. The re-design can be driven by a set of appropriate metrics (to be identified) evaluating the crosscutting of DP concerns: DP implementations characterised by bad values of the metrics will highlight the DP to be re-designed.We are carrying out some preliminary studies on a suite of selected pattern implementations from different domains (structural decomposition, data access, communication, management and access-control patterns) to get and analyse quantitative data about crosscutting distribution and modularity properties. The DP to analyze were selected from a DP set wider than the GoF catalog. As a first result we noted that most of the analysed OO pattern implementations introduce CCC at different levels of degree, as well as that AOP-based reengineering is able to improve their quality.

Three main considerations derived from the first results:

- Patterns become language idiom in some cases AOP languages can implement directly the patterns with different degree of flexibility. In our experiments we referred to AspectJ language. Interesting comparisons among different AOP language models can be performed.
- AOP pattern re-implementation produces no benefits (but results in more overhead)

  Some pattern implementations are well handled by OO languages thus they gain no actual benefits by AOP constructs. In these cases AOP implementations can be even worse than the OO ones.
- Patterns involve crosscutting and role superimpositions.

In these cases pattern implementations introduce in OO systems scattered code while AOP implementations presents better modularity.

3. DESIGN PATTERNS (DPS) AND AOP-SOFTWARE METRICS ASSESSMENT

3.1. Case Study-AJHotDraw

In order to compare the two implementations of the patterns, the true value of metrics lies in their potential to assess large pieces of code that cannot easily be characterized by developers. However given the current state of AOP, large available applications are diûcult to ûnd. In order to present, we have taken 3rd party system AJHotDraw [31], an AspectJ implementation of JHotDraw [33]. The original JHotDraw project was developped by Erich Gamma and Thomas Eggenschwiler. It is a Java GUI framework for technical and structured graphics. It has been developed as a design exercise but it is quite powerful. Its design relies heavily on some well-known design patterns. The AJHotDraw program contains more than 400 elements

(classes, interfaces and aspects). To our best knowledge, there is no application of that size that has been carefully studied in the past regarding aspect-oriented quality.

3.2. AOP Metrics Framework

The metric framework that we use is an extension of the open-source aopmetrics tool developed by Michal Stochmialek [32]. The goal of the aopmetrics project is to provide a common metrics tool for the object-oriented and the aspect-oriented programming. The actual implementation is based on the AspectJ compiler and is written using the Java5 language (only run on Java5 VM). It currently provides the extension of the following metrics suites: Chidamber and Kemerer metrics suite [22] and Robert Martin's metrics suite (package dependencies metrics) [27].Metrics implemented by the aopmetrics project can be applied to classes and aspects.

The visualization tool can use the following metrics as an input:

- Weighted Operations in Module (WOM).
- Depth in Inheritance Tree (DIT).
- Number Of Children (NOC).
- Crosscutting Degree of an Aspect (CDA).
- Coupling on Advice Execution (CAE).
- Coupling on Method Call (CMC).
- Coupling on Field Access (CFA).
- Coupling between Modules (CBM).
- Respone For aModule (RFM).
- Lack of Cohesion in Operations (LCO).

4. DISCUSSION

It is well known in the software community that low coupling and high cohesion are regarded as being quality characteristics of software systems. During the assessment AJHotDraw, we remarked that the coupling was not significantly higher in the AO version, compared to the OO version. However, when comparing the two versions, the coupling was measured using the metric CBM, which does not take into account coupling induced by join points. Thus, there could be some hidden coupling that we could not visualize when comparing the two systems. Also, most aspects seemed rather cohesive, except for some parent aspects. This may be due to the fact that most aspects do not possess any attributes. We can mention that the AJHotDraw project contained very few aspects (less than 10% of the total number of entities).Also, the visualized aspects were linked to only few classes, usually the more complex ones. This could indicate that aspects are not so much reusable in practice. These results may be evidence that the aspect-oriented paradigm is still not well understood, or that aspects are hardly applicable on genuine software projects. Aspects might also offer benefits that were not directly assessed in our experiment, such as the separation of concerns or (un) pluggability.

5. CONCLUSIONS

This paper presented a quantitative and a qualitative study comparing the AO and OO implementations of the GoF patterns. The results have shown that most aspect-oriented implementations provided improved separation of concerns. However, some patterns resulted in higher coupled components, more complex operations, and more LOCs in the AO solutions. It must be analyzed in conjunction with other important factors, including coupling, cohesion, and size. Sometimes, the separation achieved with aspects can generate more complicated designs. However, since this is a first exploratory study, to further confirm the findings, other rigorous and controlled experiments are needed. However, the goal was to provide some evidence for a more general discussion of what benefits and dangers the use of AO abstractions might create, as well as what and when features of the AO paradigm might be useful for the modularization of classical design patterns.

REFERENCES

[1] Coplien, J. O. Software Design Patterns: Common Questions and Answers. In: Rising L., (Ed.), *The Patterns Handbook: Techniques, Strategies, and Applications*. Cambridge University Press, NY, January 1998, pp. 311-320.

[2] Gamma, E. et al., "Design Patterns – Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994.

[3] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwing, "J. Aspect-Oriented Programming", *Proceedings of ECOOP '97*, Springer Verlag, Pages 220-242, 1997.

[4] Norvig, *P. Design Patterns in Dynamic Programming*, In: Object World 96, Boston MA, May 1996.

[5] Sullivan, *G. T. Advanced Programming Language Features for Executable Design Patterns*, Lab Memo, MIT Artificial Intelligence Laboratory, Number AIM-2002-005, 2002.

[6] The AspectJ Web site. http://www.aspectj.org

[7] The Java Web site. http://www.java.sun.com

[8] AspectJ Team. The AspectJ Guide. http://eclipse.org/aspectj/

[9] Hannemann, J., Kiczales, *G. Design Pattern Implementation in Java and AspectJ. Proc. OOPSLA'02*, Nov 2002, 161-173.

[10] R. Martin, "Object Oriented Design Quality Metrics: An Analysis of Dependencies", *In ROAD*, **2**, 1995.

[11] Gamma, E. et al., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, 1995.

[12] Kersten, A., Murphy, G., "Atlas: A Case Study in Building a Web-based Learning Environment using Aspect-oriented Programming", *Proc. of OOPSLA'99*, November 1999.

[13] Gamma, E. et al. "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, 1995.

[14] Soares, S., Laureano, E., Borba, P., "Implementing Distribution and Persistence Aspects with AspectJ". *Proceedings of the OOPSLA'02*, pp. 174-190.

[15] Walker, R., Baniassad, E., Murphy, G. "An Initial Assessment of Aspect-oriented Programming". *Proceedings of ICSE'99*, Los Angeles, USA, May 1999, pp. 120-130.

[16] Garcia, A. et al, "Separation of Concerns in Multi-Agent Systems: An Empirical Study", In Software Engineering for Multi-Agent Systems II, Springer, LNCS 2940, January 2004.

[17] Zhao, J. Towards a Metrics Suite for Aspect-Oriented Software.TR SE200213625, *Inf. Proc. Society of Japan*, 2002.

[18] Zhao, J. and Xu, B. Measuring Aspect Cohesion. *Proc. Conf. on Fundamental Approaches to Software Engineering (FASE'04)*, LNCS 2984, Barcelona, March 2004, 54-68.

[19] J. Zhao, "Measuring Coupling in Aspect-oriented Systems", *In METRICS2004: Proceedings 10th International Software Metrics Symposium*, September 2004.

[20] J. Zhao and B. Xu, "Measuring Aspect Cohesion", *In FASE'2004: Proceeding of International Conference on Fundamental Approaches to Software Engineering*, March 2004.

[21] M. Ceccato and P. Tonella, "Measuring the Effects of Software Aspectization", In WCRE '04: 1st Workshop on Aspect Reverse Engineering, 2004.

[22] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Trans.Softw. Eng.*, **20(6)**:476–493, 1994.

[23] C. SantAnna, A. Garcia, C. Chavez, C. Lucena, and A. Von Staa, "On the Reuse and Maintenance of Aspect-oriented Software: An Assessment Framework", *In Proceedings of XVII Brazilian Symposium on Software Engineering*, October 2003.

[24] Y. Coady and G. Kiczales, "Back to the Future: a Retroactive Study of Aspect Evolution in Operating . . . . . *In AOSD '03: Proceedings of the 2nd International Conference on Aspect-oriented Software Development*, Pages 50–59, New York, NY, USA, 2003. ACM Press.

[25] C. Zhang and H.-A. Jacobsen, "Quantifying Aspects in Middleware Platforms", *In AOSD '03: Proceedings of the 2nd International Conference on Aspect-oriented Software Development*, Pages 130– 139, New York, NY, USA, 2003. ACM Press.

[26] S. L. Tsang, S. Clarke, and E. Baniassad, "Object Metrics for Aspect Systems: Limiting Empirical Inference based on Modularity", *Technical Report*, 2005.

[27] G. Kiczales, "It's the Crosscutting", *In Software Development Magazine*, February 2004.

[28] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of Aspectj", *In Proceedings of 15th European Conference on Object-oriented Programming*, June 2001.

[29] G. Kiczales, J. Lamping, A. Menhdekar, C. Maeda, C. Lopes, J. Loingties, and J. Irwin., "Aspect-oriented Programming", *In Proceedings of 11th European Conference on Object-oriented Programming*, June 1997.

[30] D. Sabbah, "From Promise to Reality", *In AOSD'04: Proceedings of the 3rd International Conference on Aspect-oriented Software Development*, March 2004.

[31] Ajhotdraw project. http://sourceforge.net/projects/ajhotdraw/.

[32] Aopmetrics project. http://aopmetrics.tigris.org/.

[33] Jhotdraw project. http://www.jhotdraw.org/.