On the Impact of Aspect-Oriented Programming on Object-Oriented Metrics

Jean-Yves Guyomarc'h and Yann-Gaël Guéhéneuc

GEODES - Group of Open and Distributed
Systems, Experimental Software Engineering
Department of Informatics and Operations Research
University of Montreal, Quebec, Canada
[guyomarj, guehene]@iro.umontreal.ca

Abstract. Aspect-oriented programming is a new paradigm designed to fulfill the limitations of object-oriented programming regarding separation of concerns. The advent of a new paradigm requires software engineers to define new metrics and quality models to measure the quality of programs in this paradigm. The close relationship of aspect-oriented programming and object-oriented languages drives us to wonder about the impact of this new paradigm over object-oriented languages, and especially over object metrics. In this position paper, we attempt to present an approach to study and to understand the impact of aspect-oriented programming on object-oriented metrics.

1 Introduction

Throughout the history of software development, new development techniques were developed to deal with the ever-growing complexity of software. Currently, most development uses the object-oriented paradigm that reifies concepts, or concerns, as objects and such an approach allows programmers to decompose high level requirements into a set of functional modules. The problem is that OOP does not offer an elegant way to recompose those modules and, at the same time, ensure modular continuity[1]. These limitations of object-oriented programming (OOP) gave birth to a new paradigm: the Aspect-oriented programming (AOP).

New development paradigm and associated design practices must be evaluated in term of quality. Right now, it is very difficult to determine good design and implementation decisions for aspect-oriented programs. The software engineering community has yet to develop models to assess the quality of design and implementation solutions in the AOP paradigm, and metrics to support these models.

Current researches are divided into two main streams: On the one hand some researchers attempt to build quality models and metrics for aspect-oriented

[1] According to Bertrand Meyer [1], a method satisfies Modular Continuity if, in the software architecture that it yields, a small change in a problem specification will trigger a change of just one module, or a small number of modules.

softwares, on the other hand, others focus on the relation between AOP and OOP, and its consequences on OOP. All these works are still at an early stage.

In this context, we propose a rigorous approach to evaluate the impact of aspect-oriented programming on object-oriented metrics. Due to its close relations to OOP, AOP has a certain impact on object-oriented languages and therefore on the metrics dedicated to these languages. We intend to evaluate this impact and determine if the analysis of this impact could enable us to assess design and implementation quality of aspect-oriented software based on the extensive knowledge on object-oriented quality.

This paper is organized as follow: Section 2 introduces basic concepts of Aspect-Oriented Programming and for one of its implementation, AspectJ. Section 3 presents currents work related to both AOP and metrics. Section 4 discusses our approach. Section 5 presents some concluding remarks and directions for future work.

2 Aspect-Oriented Programming

2.1 In a Nutshell

A software system can be seen as a set of structured modules, each module representing a concern *i.e.*, a functionality or a requirement. Current abstractions offered by the object-oriented paradigm (classes, objects, methods, and attributes) are insufficient to express fully all the concerns of a software system. For example, code handling concerns such as logging, tracing, or persistence tends to be scattered and tangled all across the objects of the system. These concerns are known as crosscutting concerns because they cut across other functionality of the program. As a consequence, pieces of software implemented with the object-oriented paradigm tend to have reduced comprehensibility, maintainability, and reusability.

A solution to address the problem of crosscutting concerns is the Aspect-Oriented paradigm [2]. This new paradigm offers a way to separate concerns and to ensure a good modularization.

AOP introduces a new abstraction called an *aspect*. Thanks to this abstraction, a developer is able to encapsulate a crosscutting concern. It also introduces the mechanisms to compose aspect and components (classes, objects, methods, and attributes). These mechanisms allow the aspect to *crosscut* object-oriented abstractions.

Because the AOP paradigm aims at overcoming the limitations of OOP, it must be implemented for each object-oriented language like C++ or Java.

2.2 AspectJ

The de facto standard of Aspect-Oriented Programming is AspectJ [3] which is an extension of the Java language. We will briefly go over the AOP abstraction implemented by AspectJ [4].

Join point: A join point is a well-defined *point* in the execution of a component. It can be a method call or execution, an access to an attribute, or the execution of a constructor.

Pointcut: A pointcut is the mechanism that encapsulate join points. It can be compose of one or more join point.

Advice: An advice specifies the action (*i.e.*, code) that must take place at a certain pointcut (*i.e.*, a group of join points). With both abstractions mentioned above, advice give developer the ability to implement crosscutting concerns. There are three types of advice:

- *before:* The code declared is executed before the join point.
- *after:* The code declared is executed after the join point.
- *around:* The code declared is executed instead of the one in the join point.

Inter-type declaration: This mechanism allow the developer to crosscut concerns in a static way. It permits alterations to classes and inheritance hierarchies from outside the original class definition. We enumerate below the types of possible changes through Inter-type declaration:

- Add members (methods, constructors, fields) to types (including other aspects).
- Add concrete implementation to interfaces.
- Declare that types extend new types or implement new interfaces.
- Declare aspect precedence.
- Declare custom compilation errors or warnings.
- Convert checked exceptions to unchecked.

Aspect: An aspect is the container for the encapsulation of pointcuts, advice code, and inter-type declaration. Acting like a Java classes, it can contain its own attributes and methods.

3 AOP and Metrics: State of the Art

Although AOP is a young paradigm, some research touching metrics and quality has already been conducted.

3.1 Metrics for AOP

In [5], Jianjun Zhao attempts to quantify the data flows in aspect-oriented program. He proposes a metric suite based on a dependence model for aspect-oriented software. This model consists of a group of dependence graphs representing various dependence relationships at different levels of an aspect-oriented programs. He claims that his metric suite can evaluate the complexity of aspect-oriented software.

Jianjun Zhao introduces in [6] a coupling measure for aspect-oriented programs. He expresses this metric as the degree of interdependence between aspects and classes. In his paper is defined the first coupling framework for AOP.

In [7], Jianjun Zhao and Baowen Xu propose an approach to assessing the aspect cohesion based on dependence analysis. They maintain that the cohesion for an aspect is mainly about how tightly the attributes and modules (methods and advices) of aspects cohere. This work has been criticized and refined in [8].

On the one hand, studying AOP and its implementations, like Jianjun Zhao did, is a necessity. Software engineers must define metrics and models to assess the quality of aspect-oriented system. On the other hand, the impact of AOP over object-oriented languages, such as Java must also be evaluated and understood.

3.2 Impact of Aspect on Object-Oriented Metrics

In [9], the Sable research group proposes a rigorous method for studying the dynamic behaviour of AspectJ programs. They defines a set of specialized metrics for this purpose and use tools such as a modified version of the AspectJ compiler, which tags bytecode instructions, and a modified version of the *J dynamic metrics collection tool. Their aim, principally, is to provide ways to understand runtime behaviour of AspectJ programs.

Claudio Sant'Anna et al. [10] present an assessment framework for aspect-oriented software development composed of a suite of metrics and a quality model. This framework intends to measure reusability and maintainability of aspect-oriented programs. Their suite of metrics is principally a refinement of well-known metrics, such as the CK metrics [11] and Lopes' metrics [12]. They take into account two abstractions of AOP, aspect and advice, and evaluate their impact on the metrics referred above. Thanks to this evaluation, they can refine these metrics and propose a quality model. The work of Claudio Sant'Anna et al. [10] on the impact of AOP on object-oriented metrics is the most substantial. They refined a set of well-known object metrics regarding the effects of AOP, but they took into account only a subset of the abstractions of AOP.

4 Our approach

4.1 Objectives

We intend on extending the work of Claudio Sant'Anna et al. [10], in particular, the part concerning the refinement of object-oriented metrics, because their evaluation of the impact of AOP on object-oriented metrics is incomplete.

We want to fully evaluate this impact *i.e.*, to evaluate the effect of all the abstractions introduced by AOP, on object-oriented languages. We believe that a mechanism such as inter-type declaration has an important incidence on object-oriented metrics. For example, adding members or declaring that types extend

new types through inter-type declaration affects metrics like *coupling*, *cohesion*, or *depth of inheritance*.

Unlike the work of the Sable research group [9], we plan on studying the source code, rather than the bytecode, in a static way. Thus, we stay at the design level and assess the quality of the source code written by aspect-oriented developers.

During our review of the literature, we also noticed the lack of a common agreed-upon example of AOP program and aspect. We believe that using a unique example among all the papers published on the subject could help both the authors and the readers. So we plan to develop and to use an example complete enough to satisfy as many needs as possible among the researchers working on AOP and quality.

4.2 Methodology

Below is the methodology we expect to follow, divided in 3 steps.

Step 1 We must study the aspect-oriented paradigm to understand its relations and the way it interacts with object-oriented programming fully. We will analyze AspectJ, which seems the most likely implementation to become a standard.

Step 2 Thanks to our understanding of AOP, we shall be able to evaluate the impact of AOP over a given set of object-oriented metrics, such as the CK metrics [11] and Lopes' metrics [12]. This analysis shall enable us to isolate the mechanisms of AOP in which we shall be interested.

Step 3 Finally, we will implement a framework to capture the impact of AOP over Java. To do so, we shall extend the meta-model PADL (Pattern and Abstract-level Description Language) [13], which describes the structure of object-oriented programs, so it could desribe aspect-oriented programs too. To compute metrics, we shall rely on POM (Primitives, Operators, Metrics), a library of software metrics, and modify it if necessary. To parse and analyze AOP source code, we shall depend on the AspectJ and Eclipse's plugin AJDT tools because we expect to integrate our framework to the Eclipse tool platform.

Thanks to this framework, we shall be able to virtually compose aspects and classes as the AspectJ compiler does (this processus is also called *weaving*), and compute metrics before or after composition.

5 Conclusion and Future Work

Assessing the quality of software has been the preoccupation of software engineers for two decades. The problem of separation of concerns led to the apparition of the aspect-oriented paradigm. This new paradigm raises questions about quality, due to its close relations with object-oriented programming.

Our work on the impact of AOP on object-oriented metrics and the implementation of a measurement framework shall enable us to build quality models and to assess the quality of aspect-oriented programs.

The proposed work shall be validated through empirical studies. In fact, our framework shall enable us to appraise the quality of an object-oriented program before and after the changes made by an AOP program.

Acknowledgements

We would like to thank Stéphane Vaucher and Gabriel Michaud for their valuable contributions to this work.

References

1. Meyer, B.: Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition). Prentice Hall PTR (2000)
2. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J. In: Aspect-Oriented Programming. Volume 1241. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242
3. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ. Lecture Notes in Computer Science **2072** (2001) 327–355
4. Gradecki, J.D., Lesiecki, N.: Mastering AspectJ: Aspect-Oriented Programming in Java. Wiley (2003)
5. Zhao, J.: Towards a metrics suite for aspect-oriented software. Technical Report SE-136-25, Information Processing Society of Japan (IPSJ) (2002)
6. Zhao, J.: Measuring coupling in aspect-oriented systems. Technical report, Information Processing Society of Japan (IPSJ) (2004)
7. Zhao, J., Xu, B.: Measuring aspect cohesion. In: Proceeding International Conference on Fundamental Approaches to Software Engineering. Volume 2984., Springer-Verlag (2004) 54–68
8. Gélinas, J.F., Badri, L., Badri, M.: Aspect cohesion measurement based on dependence analysis. In: Proceedings of the Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE). (2004) 121–125
9. Dufour, B., Goard, C., Hendren, L., Moor, O.d., Sittampalam, G., Verbrugge, C.: Measuring the dynamic behaviour of aspectj programs. In: OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications. Volume 39., New York, NY, USA, ACM Press (2004) 150–169
10. Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., Staa, A.v.: On the reuse and maintenance of aspect-oriented software: An assessment framework. In: Proceedings XVII Brazilian Symposium on Software Engineering. (2003)
11. Chidamber, S., Kemerer, C.: A metrics suite for object oriented design. Software Engineering, IEEE Transactions on **20** (1994) 476–493
12. Lopes, C.I.V.: D: A Language Framework For Distributed Programming. PhD thesis, College of Computer Science of Northeastern University (1997)
13. Guéhéneuc, Y.G.: Un cadre pour la traçabilité des motifs de conception. PhD thesis, École des Mines de Nantes et Université de Nantes (2003)