What is an *Aspect* in Aspect-oriented Requirements Engineering?

Hermann Kaindl

Vienna University of Technology, Gusshausstr. 27-29
A-1040 Vienna, Austria
kaindl@ict.tuwien.ac.at

Abstract. Addressing the issue of crosscutting concerns within a software system, the notion of an *aspect* has been introduced, first for so-called Aspect-Oriented Programming (AOP) and then, more generally, for Aspect-Oriented Software Development (AOSD). Unfortunately, this notion is used with two different meanings: one as a synonym for "crosscutting concern", and the other as a means to deal with a crosscutting concern within the software. Later, this notion has been carried over to so-called Aspect-Oriented Requirements Engineering (AORE). This raises questions about the meaning(s) of an aspect in this context, and about the relationship of this notion in AORE and the same notion in AOP. We try to answer these questions and argue to define an *aspect* as a means to deal with crosscutting concerns, and *not* as a synonym of "crosscutting concern". Most importantly, an aspect in AORE is not necessarily related to an aspect within the software.

Keywords: Crosscutting concerns, aspects, requirements.

1 Introduction

Decomposing a software system according to some given approach leads to a modular decomposition, where the resulting modules encapsulate certain concerns. Other concerns, however, are usually not encapsulated in such modules and they "end up being scattered across many modules and tangled with one another" [4]. Such concerns are known as *crosscutting concerns*.

This was the motivation for developing Aspect-Oriented Programming (AOP) first, and later generalizing the approach to Aspect-Oriented Software Development (AOSD). More recently, Aspect-Oriented Requirements Engineering (AORE) received some attention.

However, the term "aspect" as used in AOSD does not always match well the intuitive and usual meaning of "aspect" in natural language, and it is not uniformly defined even in the context of programming. And is it simply possible to carry the concepts and techniques over to the realm of requirements?

We try to pinpoint open issues and to clarify the notion of an aspect. In particular, we investigate its meaning in AORE and distinguish crosscutting concerns in a requirements representation from crosscutting concerns within the software.

The remainder of this paper is organized in the following manner. First, we provide some necessary background on AOSD for making this paper self-contained. Then we present a critical view of AORE approaches, with a focus on how the notion of an aspect is defined and used. We also highlight that crosscutting concerns in AORE are not necessarily the same as crosscutting concerns within the software.

2 Background on AOSD

Since we try to answer the question what an aspect is in AORE, we review first how this term is defined in AOP and AOSD. As it turns out, there are two different views, a problem view and a solution view. In contrast, the term aspect means in natural language a particular way in which something appears or may be regarded. So, it is synonymous to, e.g., appearance, look, or facet.

2.1 Problem View

The first paper on AOP that we are aware of is [8]. It provides the following definition (dots are inserted by us):

> "With respect to a system and its implementation using a GP-based language, a property that must be implemented is:
> …
> **An aspect, if it can not be cleanly encapsulated in a generalized procedure.** Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways. …"

We do not think that this text makes it really clear what an aspect would be, but it seems as though this term means here the *problem* of crosscutting concerns rather than a means to deal with it.

The following definition from the AOSD Wiki in the Web pages of the Aspect-Oriented Software Development community is much more explicit in this regard:
http://www.aosd.net/wiki/index.php?title=Glossary#aspect

> "Aspects are one kind of concern in software development. With respect to a primary or dominant decomposition aspects are concerns that crosscut that decomposition."

While it is useful to explicitly take crosscutting concerns into account, what is the point of calling them "aspects"? In particular, this does not even match the usual meaning of this notion in natural language.

2.2 Solution View

Interestingly, the very same Web pages contain a different definition of "aspect" in the context of the definition of "concern", which shows a different view (dots are inserted by us):
http://www.aosd.net/wiki/index.php?title=Glossary#concern

> "… There are many formulations used to capture concerns as well-identified separate units, aspects are one such mechanism, that are tuned to capturing crosscutting concerns."

We consider this a *solution* view, since it is about a "mechanism" to deal with crosscutting concerns, rather than crosscutting concerns themselves.

Also in the literature of Aspect-Oriented Programming (AOP) an aspect is defined, e.g., in [4] as

> "… a mechanism beyond subroutines and inheritance for localizing the expression of a crosscutting concern."

In AOSD, the term aspect is used, e.g., in [6]

> "for a module that is potentially able to encapsulate software or design artifacts treating an otherwise crosscutting concern".

In the same spirit, we transferred this aspect approach to architectures of general systems involving mechanical, electronic and software parts [1]. The key point is to bundle the part of the architectural design that deals with the crosscutting concern in an extra modular unit, even though it may crosscut mechanical, electronic and software parts.

In all three cases, an aspect is clearly something *within* the software system, more generally as a mechanism or more specifically as a module. We argue that it makes more sense to use the notion of an aspect for such a means to deal with crosscutting concerns rather than as a synonym of "crosscutting concern". While this use of the notion does also not match its usual meaning in natural language, it makes sense as a technical term for a technical solution.

2.3 "Early Aspects"

More recently, there was a trend to deal with aspects "earlier" in the life cycle, see, e.g., [3]:

> "*Early aspects* are concerns that crosscut an artifact's *dominant decomposition*, or base modules derived from the dominant separation-of-concerns criterion, in the early stages of the software life cycle. "Early" signifies occurring before implementation in any development iteration."

Here, requirements and software design are simply treated together, as opposed to implementation in AOP.

Also in [3], a straight-forward distinction between requirements and architecture can be found as follows:

> "An aspect in requirements is a concern that crosscuts requirements artifacts; an aspect in architecture is a concern that crosscuts architectural artifacts."

Still, how do crosscutting concerns of requirements artifacts relate to those of architectural artifacts? Can the mechanisms to capture them be the same? What is an aspect in the context of requirements?

3 A Critical View of AORE

Let us have a closer look into the literature of AORE, where aspects have been carried over to requirements engineering. Also in AORE, some approaches use this notion as a synonym of "crosscutting concern", while others propose mechanisms for dealing with such concerns. Both do not make clear, however, that crosscutting concerns in the context of requirements are not necessarily the same as crosscutting concerns inside the software. In this spirit, we provide a critical view of AORE.

3.1 Claim of Usefulness for Design and Implementation

Generally, AORE approaches claim that dealing with aspects (in either meaning) is useful for software development. See, e.g., [3] again:

> "Identifying and managing early aspects helps to improve modularity in the requirements and architecture design and to detect conflicting concerns early, when trade-offs can be resolved more economically."

This is, in effect, mixing requirements and software design, since modularity in the requirements and in the architecture design are not necessarily the same.

In [4], even a direct link is made from certain requirements to the implementation w.r.t. crosscutting concerns:

> "Aspectual requirements are concerns that (for common structural decompositions) introduce crosscutting in the implementation."

3.2 Hidden Assumption

The part of this quote "for common structural decompositions" may even indicate an otherwise well-hidden assumption. Most AORE approaches seem to tacitly assume that there is the *same* (or at least similar) dominant decomposition of the requirements (from outside the software) and of the software system itself!

Only under such an assumption, the *same* concerns might crosscut the requirements and the software design and implementation. So, is it usually fulfilled?

The requirements are often decomposed into functional and non-functional ones. The functional requirements are often grouped according to a functional decomposition of the overall functionality needed. Use cases are related to each other in use case diagrams. Such diagrams may be viewed as representing a functional decomposition according to use. W.r.t. to such a dominant decomposition, at least some of the non-functional requirements usually crosscut the functional requirements and/or use cases.

Is the software internally always decomposed according to these same principles? This is not necessarily the case, since certain architectural styles such as layers, distributed architecture, repository style etc. impose other decompositions, at least in addition. Ironically, the non-functional requirements are the ones that may indicate a certain software architecture following a different decomposition, e.g., security requirements may indicate a layered architecture.

3.3 Mix of Requirements and Software Design

So, there is a similar confusion of requirements and software design in AOSD as previously in object-oriented development methods [5]. Much as the wide-spread confusion of "analysis" and "design" especially in the object-oriented community, this confusion may create issues both in theory and in practical applications.

3.4 AORE Approaches Based on this Assumption

The following AORE approaches seem to be based on the hidden assumption of having the same decompositions for requirements and within the software:

- In [9], a systematic way of identifying crosscutting concerns using and w.r.t. *viewpoints* is presented. It leads to understanding crosscutting concerns in a representation of requirements that is organized according to viewpoints. We could not find there any indication about consequences for possible aspects within the software, however. Whether or not the same crosscutting concerns will exist inside the software will largely depend on the dominant decomposition of this software.
- In [11], a systematic way of identifying crosscutting concerns from goal models is presented (including "functional goals" and "soft-goals"). The dominant decomposition appears to be a functional decomposition. Also here we could not find any indication about consequences for possible aspects within the software. Whether or not the same crosscutting concerns will exist inside the software, will also here largely depend on the dominant decomposition of this software.
- In [2], a systematic way of identifying crosscutting concerns is presented, and modeling them in a software design language using their *Theme* approach. This seems to bridge requirements and design more consciously with regard to crosscutting concerns.

3.5 AORE Approaches Using Aspects as a Solution Concept

The following AORE approaches use the aspect notion as a solution concept for dealing with crosscutting concerns:

- In [7], *aspects* are basic "modules" of requirements specifications, represented using XML tagging. Such an aspect groups crosscutting requirements. This is in the spirit of the aspect definition of [6], but the hidden assumption of the same dominant decomposition of the requirements and of the software system itself seems to be behind this approach as well.
- In [10], a very appealing application of the aspects idea from programming in use cases is proposed. The program code of the former corresponds to the action sequences of the latter, both being behavior specifications. The former behavior is inside the software, while the latter is

behavior in the composite system (including the software system and the user). While this approach leads to untangling of the scenarios of these use cases, it is not clear what the consequences are for untangling of code in the implementation.

4 Conclusion

The very notion of an *aspect* in software development is a misnomer somehow, since it barely matches the intuitive and usual meaning of "aspect" in natural language. Even worse, it is ambiguously defined in AOP.

Therefore, we tried to clarify the different meanings of "aspect", with a focus on AORE. We argue to define it as a means to deal with crosscutting concerns, and not as a synonym of "crosscutting concern".

From a research point of view, it is important to have a conceptually clean view of these issues and concepts. Otherwise, there may be confusion in practical applications. In particular, the hidden assumption of the same dominant decomposition of the requirements and the software itself is dangerous. Whenever it is not fulfilled, a crosscutting concern identified in a requirements representation is not necessarily a crosscutting concern inside the software, i.e., in design and implementation.

Still, AORE is useful in its spirit of dealing with crosscutting concerns "early" and also in the context of requirements. This may lead to a better understanding of the requirements and of some of the issues involved in developing the system. However, it is necessary to consider the decompositions used in the requirements representation and in the software itself. An aspect identified in aspect-oriented requirements engineering is not necessarily related to an aspect within the software.

Acknowledgments

Edin Arnautiovic provided useful comments to an earlier version of this paper.

References

1. Arnautovic, E., Kaindl, H., *Aspects* for crosscutting concerns in systems architectures?, In *Proc. of the Second Annual Conference on Systems Engineering Research (CSER-04)*, 2004, pp. 1–10
2. Baniassad, E., Clarke, S., 2004. Theme: An Approach for Aspect-Oriented Analysis and Design. In *Proc. of the 26th International Conference on Software Engineering (ICSE'04)*, pp. 158–167
3. Baniassad; E., Clements, P.C., Araujo, J., Moreira, A., Rashid, A., Tekinerdogan, B., 2006. Discovering early aspects, *IEEE Software*, 23(1): 61–70
4. Elrad, T., Filman, E.R., Bader, A., 2001. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32

5. Kaindl, H., 1999. Difficulties in the Transition from OO Analysis to Design, *IEEE Software*, 16(5):94–102
6. Katara, M., Katz, S., 2003. Architectural Views of Aspects. In *Proc. of the Second International Conference on Aspect-Oriented Software Development (AOSD 2003)*, ACM Press, pp. 1–10
7. Katz, S., Rashid, A., 2004. From aspectual requirements to proof obligations for aspect-oriented systems. In *Proc. of the 12th IEEE International Requirements Engineering Conference (RE'04)*, pp. 48–57
8. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J., 1997. Aspect-Oriented Programming. In *Proc. ECOOP '97*, LNCS 1241, Springer-Verlag, pp. 220–242
9. Rashid, A., Sawyer, P., Moreira, A., Araujo, J., 2002. Early aspects: a model for aspect-oriented requirements engineering. In *Proc. of the IEEE Joint International Conference on Requirements Engineering (RE'02)*, pp. 199–202
10. Xu, D., Vivek Goel, Nygard, K., 2006. An Aspect-Oriented Approach to Security Requirements Analysis. In *Proc. of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, pp. 79–82
11. Yu, Y., Leite, J.C.S.P., Mylopoulos, J., 2004. From goals to aspects: discovering aspects from requirements goal models. In *Proc. of the 12th IEEE International Requirements Engineering Conference (RE'04)*, pp. 38–47