A RUP Based Analysis and Design with Aspects

Leonardo Cole[1]*, **Augusto Sampaio**[1]†

[1]Informatics Center
Federal University of Pernambuco
Caixa Postal 7851 - 50.732-970 Recife, PE

lcn@cin.ufpe.br ,acas@cin.ufpe.br

Abstract. *Aspect-Oriented Programming (AOP) is a new programming paradigm that comes to complement the Object-Oriented (OO) paradigm solving problems not adequately addressed by the later. Thus, it seems appropriate to adapt the software processes and life cycles to make use of its advantages. The Rational Unified Process (RUP) is one of the most widely used software development processes, however it has not been originally designed to use AOP. RUP must be adapted to incorporate AOP concepts. This work proposes adjustments to the most affected RUP discipline, Analysis and Design, in order to incorporate new concepts and techniques of AOP.*

1. Introduction

Object Oriented (OO) [Meyer, 1997, Booch, 1994] programming is a well established and widely used paradigm. However, it still can not solve some problems [Ossher et al., 1996, Ossher and Tarr, 1999] related to the implementation of some concerns that generally spreads over many different modules of the system (code scattering); this code is not related to the behavior implemented by the module (code tangling). Aspect Oriented Programming (AOP) [Elrad et al., 2001] is a new programming paradigm intended to complement OO, solving those problems.

AOP implements these crosscutting concerns as aspects. Each aspect defines some pointcuts and advices. The pointcuts are expressed in some way to define singular points on the system execution flow where an advice can execute; these points are called joinpoints. The advices specify the code to execute and when it should happen. The possibilities are before, after or instead of the specified joinpoint itself.

Rational Unified Process (RUP) [Jacobson et al., 1999] is one of the most widely used software development process. It defines disciplines, activities, roles and artifacts that control the software life cycle, as an attempt to contribute to the production of software with quality, on budget and within schedule.

As a new and promising programming paradigm, it seems appropriate that software processes and life cycles be adapted to make use of its advantages. This work proposes adjustments to the RUP Analysis and Design discipline in order to incorporate the new concepts and techniques of AOP.

*Supported by CAPES.
†Partially supported by CNPq, process number ...

In particular, this work can be regarded as an extension of [Piveta and Devegili, 2002], where Piveta and Devegili briefly suggest some new roles and activities to the Analysis and Design discipline in the presence of AOP. We developed his ideas to propose a more precise description of the necessary adaptations to the RUP Analysis and Design (A&D) discipline. We give an overview of each workflow detail, proposing changes to the related activities and even suggesting the creation of new activities.

Several of the RUP workflows are affected by the use of AOP as suggested in [Piveta and Devegili, 2002]. We focus on the A&D discipline. However, we first discuss some related work on the Requirements discipline which is the main input to the A&D discipline. We also present some suggestions and guidelines to future work on the Implementation discipline.

The reminder of this paper is organized as follows Section 2 gives an overview of RUP singling out the A&D discipline. Section 3 introduces the Requirements discipline making use of aspects; then we suggest the necessary adaptations to the A&D discipline in Section 4. Section 5 discusses our conclusions, related and future work.

2. The Rational Unified Process

”The Rational Unified Process or RUP product is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget” [Jacobson et al., 1999].

RUP is an object-oriented and Web-enabled program development methodology. It is like an online mentor that provides guidelines, templates, and examples for all aspects and stages of program development. RUP and similar products, such as Catalysis [D’Souza and Wills, 1998] and the OPEN Process [Firesmith and Henderson-Sellers, 2001], are comprehensive software engineering tools that combine the procedural aspects of development (such as defined stages, techniques, and practices) with other components of development (such as documents, models, manuals, code, and so on) within a unifying framework.

RUP establishes four phases of development, each of which is organized into a number of separate iterations that must satisfy defined criteria before the next phase is undertaken: in the inception phase, developers define the scope of the project and its business case; in the elaboration phase, developers analyze the project’s needs in greater detail and define its architectural foundation; in the construction phase, developers create the application design and source code; and in the transition phase, developers deliver the system to users. RUP provides a prototype at the completion of each iteration. The product supplies an HTML-based description of the unified process that an organization can customize for its own use.

There are four primary modelling elements: roles, activities, artifacts and workflows. Roles represents behaviors and responsibilities of the participants of the development team (i.e. system analyst, architect, designer). Activities are the work performed by the roles (i.e. find use cases and actors, review the design and execute a performance

test). Artifacts are information produced or used by a process (i.e. software architecture document, design model).

Workflows describe the sequence of activities that produce some value result and show the interactions among roles. There are seven workflows considered in RUP: Requirements, Analysis and Design, Implementation, Test, Configuration and Change Management, Project Management and Environment. Besides the last three, the artifacts produced in one workflow are the input to the next.

Requirements goals are to establish and maintain agreement with the customers and other stakeholders on what the system should do, to provide system developers with a better understanding of the system requirements, to define the boundaries of (delimit) the system, to provide a basis for planning the technical contents of iterations and for estimating cost and time to develop the system, and finally, to define a user-interface for the system, focusing on the needs and goals of the users.

Analysis and Design purpose is to transform the requirements into a design of the system-to-be, to evolve a robust architecture for the system and to adapt the design to match the implementation environment, designing it for performance. This workflow can be seen in Figure 1.

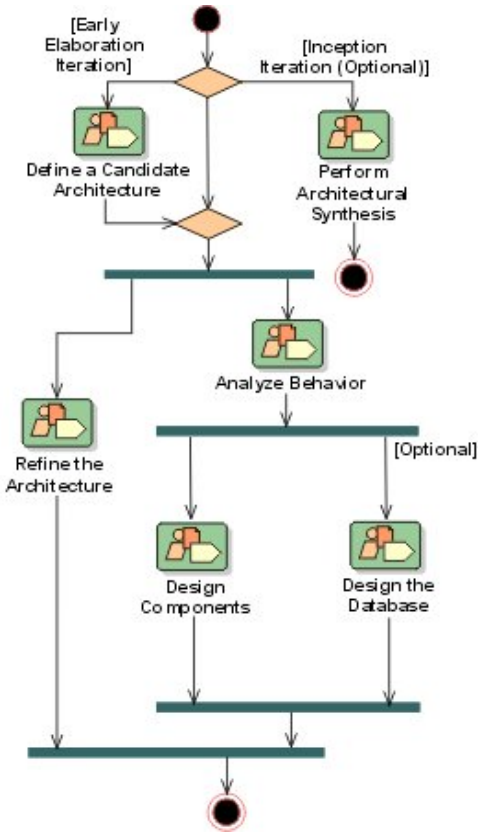

Figure 1: Analysis and Design workflow [Jacobson et al., 1999]

In the Inception Phase, Analysis and Design is concerned with establishing whether the system as envisioned is feasible, and with assessing potential technologies for the solution (in Perform Architectural Synthesis). If it is felt that little risk attaches to the development (because, for example, the domain is well understood, the system is not

novel, and so on) then this workflow detail may be omitted.

The early Elaboration Phase focuses on creating an initial architecture for the system (Define a Candidate Architecture) to provide a starting point for the main analysis work. If the architecture already exists (either because it was produced in previous iterations, in previous projects, or is obtained from an application framework), the focus of the work changes to refining the architecture (Refine the Architecture) and analyzing behavior and creating an initial set of elements which provide the appropriate behavior (Analyze Behavior).

After the initial elements are identified, they are further refined. Design Components produce a set of components which provide the appropriate behavior to satisfy the requirements on the system. If the system includes a database, then Design the Database occurs in parallel. The result is an initial set of components which are further refined in Implementation.

Implementation defines the organization of the code, in terms of implementation subsystems organized in layers. It implements the design elements in terms of implementation elements (source files, binaries, executables, and others). Tests the developed components as units to integrate the results produced by individual implementers (or teams), into an executable system.

This work focuses on the Analysis and Design workflow. However, it is important to consider the Requirements and the Implementation workflows as they, respectively, generate the input and use the output of the A&D workflow. The reminder of the workflows are beyond the scope of our work, details can be found elsewhere [Jacobson et al., 1999].

3. Requirements Assumptions

The phase where aspects should be introduced during the software development process is still discussed. In [Rashid et al., 2003, Rashid et al., 2002] they argue that early separation of crosscutting properties supports effective determination of their mapping and influence on latter development stages. As Requirements generates the input for the Analysis and Design workflow, we consider the work proposed by [Rashid et al., 2003, Rashid et al., 2002] and [Sousa et al., 2003].

The work developed in [Rashid et al., 2002, Rashid et al., 2003] proposes a generic model for Aspect Oriented Requirements Engineering (AORE), the resulting activities can be seen in Figure 2.

[Sousa et al., 2003] suggests an adaptation to the NFR-Framework(Non Functional Requirements-Framework) [Chung et al., 1999] in order to improve the mapping of crosscutting non-functional requirements onto artifacts at later development stages and to make better composition of those requirements with non-crosscutting ones.

They suggest it is more adequate to deal with NFR *operationalizations* in the context of Aspect Oriented Requirements Engineering because they better reflect how the crosscutting concern will be implemented and therefore improving the composition and the mapping of crosscutting requirements onto artifacts at later development stages. Nevertheless, those previous AORE models fail to address the issue of non-functional re-

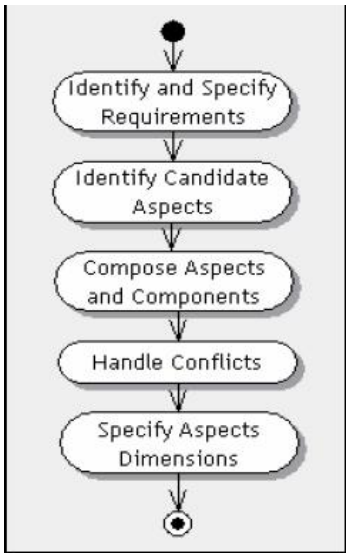

Figure 2: AORE Generic Model [Rashid et al., 2003]

quirements, specially because they deal with abstract NFRs instead of NFR *operationalizations*. Their proposed model is shown in Figure 3.

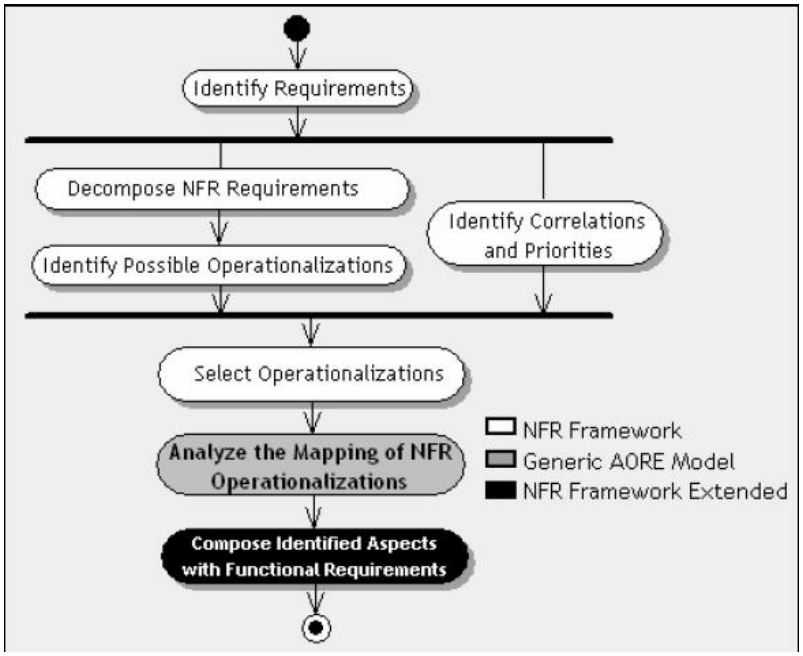

Figure 3: Proposed Adaptation of NFR Framework [Sousa et al., 2003]

Altogether, this work assumes that the Requirements workflow is executed considering aspects as described in [Sousa et al., 2003] and thus we assume the existence of an artifact containing the aspect information as an input for the Analysis and Design workflow. For instance, this artifact should describe the mapping of the NFR *operationalizations* with respect to artifacts to be generated at later stages. Figure 4 shows an example of this information representation.

NFR CONCERN		NFR *OPERATIONALIZATION*	MAPPING
SECURITY	ACCURACY	*Limited Value*	Aspect
		Firewall	Architectural Decision
	CONFIDENTIALITY	*Data Encryption*	Aspect
		Identification	Aspect
		Internet Password Request	Aspect
		Personal Data Validation	Aspect
	AVAILABILITY	*Duplication Server*	Architectural Decision
		Mirroring Database	Architectural Decision

Figure 4: Mappings of NFR *operationalizations* [Sousa et al., 2003]

4. Analysis and Design Adaptations

This section represents the main contribution of this work. It is structured similar to the way RUP explains its workflows details and activities. Only the activities affected by AOP are presented and its description includes only the information relevant to AOP. Some activities appear in more than one workflow, in these cases, we show only additional information and differences to the latter workflow, otherwise, if it is the same, the activity is omitted.

The A&D workflow is composed of six details: Perform Architectural Analysis, Define a Candidate architecture, Refine the Architecture, Analyze Behavior, Design Components and Design Database. Each detail is composed of some activities and describe what to be done, who is responsible and what are the artifacts produced or updated.

4.1. Perform Architectural Analysis

The intent of this workflow detail is to construct and assess an Architectural Proof-of-Concept to demonstrate that the system, as envisioned, is feasible. It is conducted during the inception phase and is responsibility of the Architect.

Architectural Analysis

The purpose of this activity is to define a candidate architecture for the system based on experience gained from similar systems or in similar problem domains and to define the architectural patterns, key mechanisms, and modelling conventions for the system.

- Survey Available Assets
  In addition to regular assets consideration, this step should also look for aspectual assets.
- Identify Key Abstractions
  The abstractions selected in this phase should include the ones necessary to uncover aspects, guiding the architect on the construction of the Architectural Proof-of-Concept without neglecting aspects.
- Develop Deployment Overview
  The geographical study on this step should consider the possibility of modelling distribution as an aspect.

Construct Architectural Proof-of-Concept

The purpose of this activity is to synthesize at least one solution (which may simply be conceptual) that meets the critical architectural requirements. It must incorporate critical aspects as they also need validation.

Assess Viability of Architectural Proof-of-Concept

The purpose of this activity is to evaluate the synthesized Architectural Proof-of-Concept to determine whether the critical architectural requirements are feasible and can be met (by this or any other solution). It is important to consider the critical aspects that influence the architecture.

4.2. Define Candidate Architecture

The purpose of this workflow detail is to create an initial sketch of the software architecture. Although modelling crosscutting concerns as aspects usually simplifies the architecture in general, most of the architectural modelling decisions must be aware of aspects because the architecture is directly affected by them. In addition, it is also important to work on subsystem and component interfaces, preparing them to offer the necessary joinpoints to the aspects. Thus, the architect should keep the aspects in mind even though he is modelling a simple architecture.

The architecture is also influenced by the goals identified during the requirements workflow that maps to an architecture decision (see Section 3).

Architectural Analysis

- Define the High-Level Organization of Subsystems
  This step should relate aspects to the layering organization and consider adaptations necessary to the patterns adopted.
- Identify Key Abstractions
  The goals identified during requirements which maps to an architecture decision should be considered in this activity. As the aspects are usually abstractions to the concerns it implements, this step should include aspects on relationships diagrams and descriptions. This step also define patterns used to refine the architecture. There are several patterns suggested to make better use of aspects that should be considered in this step. For instance, the PaDA [Soares and Borba, 2002] suggests a pattern for distribution aspects taking advantages from the use of AOP.
- Identify Analysis Mechanisms
  Decide which mechanisms maps to aspects and facilitate architecture comprehension. Almost all of the architectural mechanisms can be represented as aspects and be abstracted from the architecture design.

Use Case Analysis

- Find Analysis Aspects
  Considering the mapping suggested on the Requirement workflow(see Section 3), identify the model elements which will be capable of realizing the goals described on requirements.
- Distribute Behavior to Analysis Classes
  Include aspects on the collaborations, showing aspects dependencies and responsibilities.

- Describe Responsibilities
  To describe the responsibilities of aspects identified.
- Describe Attributes and Associations
  Define other classes and aspects on which the identified aspects depends. Define the classes affected by aspects and possibly determine a set of joinpoints used by the aspects.
- Qualify Analysis Mechanisms
  The mechanisms mapped to aspects where treated on previous activities, that is, this step considers only the mechanisms not modelled as aspects.

4.3. Refine Architecture

This workflow detail provides the natural transition from analysis activities to design activities, identifying: appropriate design elements from analysis elements and appropriate design mechanisms from related analysis mechanisms.It also describes the organization of the system's run-time and deployment architecture, organizes the implementation model so as to make the transition between design and implementation seamless, maintains the consistency and integrity of the architecture, ensuring that: new design elements identified for the current iteration are integrated with pre-existing design elements and maximal re-use of available components and design elements is achieved as early as possible in the design effort.

Identify Design Elements

- Identify Events and Signals
  This step should consider identifying the set os joinpoints required by the aspects.
- Identify Classes, Active Classes and Subsystems
  Refine the aspects identified during analysis and their relations to the new design elements. This step identifies the active classes(threads), therefore, concurrency should be considered during this activity. Concurrency is a common example of crosscutting concern as its effects spreads over the processes and consequently over different modules even though its behavior is modular. Hence, concurrency control is a strong candidate to be implemented as an aspect.
- Identify Subsystem Interfaces
  Subsystems affected by aspects should provide an interface with the necessary set of joinpoints to be used by the affecting aspects.

Identify Design Mechanisms

The mechanisms mapped to aspects where treated on previous activities, that is, this step considers only the mechanisms not implemented as aspects.

Incorporate Existing Design Elements

- Identify Reuse Opportunities
  Look for similarities and probably unification of interfaces affected by aspects. Another important consideration in this step is the identification of common aspect features as an indicative of an abstract aspect.

Describe Distribution

As a refinement to the Deployment Model created during Architectural Analysis, this activity is influenced by the use of AOP only if the distribution is modelled as aspects. In this case, this activity should refine de Deployment Model to reflect the current design, making it clear how the distribution aspects affect the deployment configuration.

Describe the Runtime Architecture

This activity is mainly focused on process and its organization, identifying them, how they communicate, their life cycle and finally how they share resources. Thus, new threads can be identified during this activity implying on changes to the concurrency policy. If concurrency is modelled by aspects, the aspects should be modified to control the new threads.

4.4. Analyze Behavior

The purpose of this workflow detail is to transform the behavioral descriptions provided by the requirements into a set of elements upon which the design can be based. It occurs in each iteration in which there are behavioral requirements to be analyzed and designed.

The activities concerning user interface are not affected by the use of AOP. The other activities specified on this workflow detail are affected and were already defined (see Section 4.3.

4.5. Design Components

The purpose of this workflow detail is to refine the design of the system. The activities described by this workflow detail are not directly affected by the use of AOP. However, there is a necessity for a new separate activity that we call Design Aspects.

Design Aspects

The purpose of this activity is to ensure that sufficient information is provided to unambiguously implement the aspects, refining the already identified joinpoint dependencies, and to identify the pointcuts and advices.

- Define Aspect Priorities and Precedence
  This decisions should be based on the conflict identification and resolution from Identify Design Mechanisms on the Refine Architecture workflow detail.
- Define Aspect Pointcuts
  This step is responsible for grouping the joinpoints on which the aspect depends, creating the necessary pointcuts and deciding the context that must be exposed to the advices. The grouping occurs to join all the joinpoints that are affected the same way and are able to expose the same context.
- Define Aspect Advices
  Specify only the kinds of advice(i.e. before, after and around) and the context exposure based on the identified pointcuts.
- Define Structure Influence
  This step is responsible for specifying how the aspects change the hierarchy of classes, identifying the new relationships. It is also responsible to determine how the aspects introduce behavior to the classes, identifying which attributes and methods must be introduced to which classes.

5. Discussion

We proposed adaptations to activities of the the Rational Unified Process(RUP) [Jacobson et al., 1999] Analysis and Design discipline in order to incorporate the new concepts and advantages of AOP.

In [Piveta and Devegili, 2002], Piveta and Devegili briefly suggests some changes to the roles, activities and artifacts of the Analysis and Design discipline in the presence of AOP. We developed his ideas and proposed a more precise description of the necessary adaptations to the RUP Analysis and Design discipline. We gave an overview of each workflow detail, proposing changes to the related activities and suggested the creation of a new activity, Design Aspects.

This work is not complete in the sense it only deals with one discipline defined by RUP. Although some disciplines are not affected by the use of AOP, there are disciplines, other than Analysis and Design, which need to be adapted.

Hence, a continuation of this work would deal with the adaptation of the Implementation workflow as it is strongly affected by the use of AOP and it follows the Analysis and Design workflow. One of the primary impacts is on the implementation language which generally is an extension to a known language or a framework implemented in one. Once the developer is used to the new language there are necessary adaptations to the way classes are implemented, to incorporate aspect implementations. In a similar way, the tests should be adapted to exploit the concerns implemented by aspects.

Besides [Piveta and Devegili, 2002], as far as we know, there are no other work that approaches this subject in a similar way to ours. However, there are related problems which were not discussed here and are treated elsewhere. The representation of crosscutting features is discussed in [Stein et al., b] and its representation in UML is treated in [Stein et al., a]. The representation of aspects during the design phase as an extension to UML allowing expression of aspects and its relationships to classes and other aspects is proposed in [Suzuki and Yamamoto, 1999]. Finally, [Stein et al., 2002] discusses the representation of joinpoints in UML.

RUP adaptations are necessarily common as it is a framework, an adaptable process that should be tailored to specific companies and projects. Nevertheless, there are general adaptations that maintain the general propose of the framework. [Kruchten, 2001] suggests adaptations to RUP with the objective of using it to evolve a legacy project, [Moraes, 2002] adapts RUP to incorporate concepts of the Architecture Based Development Process which aims to the architecture as a bridge between the requirements and the implementation. Thus, their adaptations are mainly on the Analysis and Design workflow.

There is also adjustments to use RUP in more specific domains, [Araújo, 2001, Souza, 2002] focused on adaptations to the Analysis and Design workflow to develop Web based applications allowing the software development process to increase product quality at lower costs.

References

[Araújo, 2001] Araújo, A. C. M. (2001). Framework de Análise e Projeto Baseado no RUP para o Desenvolvimento de Aplicações Web.

[Booch, 1994] Booch, G. (1994). *Object–Oriented Analysis and Design with Applications*. Benjamin/Cummings, second edition.

[Chung et al., 1999] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (1999). *Non-Functional Requirements in Software Engineering*, volume 5 of *The Kluwer International Series In Software Engineering*. Kluwer Academic Publishers.

[D'Souza and Wills, 1998] D'Souza, D. F. and Wills, A. C. (1998). *Objects, Components, and Frameworks with UML : The Catalysis(SM) Approach*. Addison-Wesley Pub Co.

[Elrad et al., 2001] Elrad, T., Filman, R. E., and Bader, A. (2001). Aspect–Oriented Programming. *Communications of the ACM*, 44(10):29–32.

[Firesmith and Henderson-Sellers, 2001] Firesmith, D. and Henderson-Sellers, B. (2001). *The OPEN Process Framework. An Introduction*. Addison-Wesley.

[Jacobson et al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison–Wesley.

[Kruchten, 2001] Kruchten, P. (2001). Using the RUP to Evolve a Legacy System.

[Meyer, 1997] Meyer, B. (1997). *Object–Oriented Software Construction*. Prentice–Hall, second edition.

[Moraes, 2002] Moraes, M. A. F. (2002). Um Framework de Análise e Projeto Baseado em Arquitetura de Software.

[Ossher et al., 1996] Ossher, H., Kaplan, M., Katz, A., Harrison, W., and Kruskal, V. (1996). Specifying suject–oriented composition. *TAPOS*, 2(3):179–202. Special Issue on Subjectivity in OO Systems.

[Ossher and Tarr, 1999] Ossher, H. and Tarr, P. (1999). Using subject–oriented programming to overcome common problems in object–oriented software development/evolution. In *International Conference on Software Engineering, ICSE'99*, pages 698–688. ACM.

[Piveta and Devegili, 2002] Piveta, E. K. and Devegili, A. J. (2002). Aspects in the Rational Unified Process Analysis and Design Workflow. In *Aspect Oriented Design Workshop, AOSD 2002*, Enschende, Netherlands.

[Rashid et al., 2003] Rashid, A., Moreira, A., and Araújo, J. (2003). Modularisation and composition of aspectual requirements. In *2nd International Conference on Aspect-Oriented Software Development*, pages 11–20. ACM.

[Rashid et al., 2002] Rashid, A., Sawyer, P., Moreira, A., and Araújo, J. (2002). Early aspects: A model for aspect-oriented requirements engineering. In *IEEE Joint International Conference on Requirements Engineering*, pages 199–202. IEEE, IEEE Computer Society Press.

[Soares and Borba, 2002] Soares, S. and Borba, P. (2002). PaDA: A Pattern for Distribution Aspects. In *Second Latin American Conference on Pattern Languages Programming — SugarLoafPLoP*, Itaipava, Rio de Janeiro, Brazil.

[Sousa et al., 2003] Sousa, G. M. C., Silva, I. G. L., and Castro, J. B. (2003). Adapting the NFR Framework to Aspect-Oriented Requirements Engineering. In *XVII Brazillian Symposium on Software Engineering - SBES*.

[Souza, 2002] Souza, R. A. C. (2002). Uma Extensão do Fluxo de Análise e Projeto do RUP para o Desenvolvimento de Aplicações Web.

[Stein et al., a] Stein, D., Hanenberg, S., and Unland, R. Designing aspect-oriented cross-cutting in UML.

[Stein et al., b] Stein, D., Hanenberg, S., and Unland, R. Position paper on aspect-oriented modeling: Issues on representing crosscutting features.

[Stein et al., 2002] Stein, D., Hanenberg, S., and Unland, R. (2002). On representing join points in the UML. In Kandé, M., Aldawud, O., Booch, G., and Harrison, B., editors, *Workshop on Aspect-Oriented Modeling with UML*.

[Suzuki and Yamamoto, 1999] Suzuki, J. and Yamamoto, Y. (1999). Extending UML with aspects: Aspect support in the design phase. In *ECOOP Workshops*, pages 299–300.