# Early Aspects Refactoring

Ricardo A. Ramos[1], Jaelson Castro[1], João Araújo[2], Ana Moreira[2], Fernanda Alencar[1] and Rosangela Penteado[3]

1 Centro de Informática - Universidade Federal de Pernambuco (UFPE)
{rar2, jbc}@cin.ufpe.br, frma@ufpe.br

2 Departamento de Informática, FCT, Universidade Nova de Lisboa (UNL)
{amm, ja}@di.fct.unl.pt

3 Universidade Federal de São Carlos (UFSCAR)
rosangel@dc.ufscar.br

**Abstract.** Typical problems of a requirements document, regarding its contents and organization, involve deficient modularization where requirements artifacts may deal with too much information, duplication of requirements, scattering requirements, tangled problems, among other problems. In this paper we describe how to improve requirements documents by removing duplication of information using aspect-oriented refactoring.

**Keywords:** Refactorings and Early Aspects.

## 1 Introduction

Several approaches are used to express the requirements of a software system and are extensively used in both academia and industry. These approaches describe the functionalities to be provided and the interactions between the users and the system. Requirements are structured and described using, for example, use cases, viewpoints, textual descriptions, activity diagrams or sequence diagrams [2].

Over the past years, we have observed that a set of typical issues seems to plague the requirements specifications, such as requirements that are abandoned and no longer relevant, descriptions that are unnecessarily long and complex, and information that is duplicated. Inspired in the code refactoring literature [9], we set off to identify a bad smell related to duplicated requirements that could indicate potential refactoring opportunities.

These bad smells decrease reuse, not only during implementation, but throughout the development process [1] and can be minimized by the identification of their symptoms and the removal of their causes. These symptoms may indicate potential problems with the software [2] and can be removed using appropriate refactoring transformations. The removal of these bad smells in the early stages of a software development process reduces the costs associated with software changes. This cost reductions could be three to six times bigger in later stages than during requirements activities [10].

Opdyke, in [9], initially coined the term refactoring as the process of improving the design of existing software using transformations, without changing its observable behaviour. The term is also used to refer to program restructuring operations aiming to support the design, evolution and reuse of object-oriented software. Refactorings express ideas of good style, which can have a significant impact on the maintainability and evolvability of code bases.

There are some tools for refactoring requirements documents [21] , [22], [16] but they focus on specific techniques, such as use case models and do not directly address textual descriptions or other mechanisms used to detail requirements. Moreover, these approaches do not provide any guidelines about how to identify the problems. As such, they say nothing about which refactoring can be used to address those issues.

In this paper we describe a generic approach to identify duplicated, tangled and scattered requirements as a refactoring opportunity in requirements descriptions, together with an associated early aspect refactoring. The approach can be instantiated with any of the existing requirements description techniques (e.g., use cases, viewpoints, goals). In this paper we will demonstrate our ideas using Multi-Dimensional Separation of Concerns [8] and use cases approaches. In summary, the main contributions of this paper are twofold:

i) The definition of the refactoring opportunity that helps to find duplicated, tangled and scattered requirements in requirements document. We describe (a) how to identify occurrences of these problems and (b) indicate how to use the refactoring that can to minimize the effects of the problem.

ii) The definition of the Extract Early Aspect refactoring. This refactoring contains the context that suggests the application of the refactoring, the type of solution provided by the refactoring, a motivation for the transformations, its mechanics (a set of well defined steps) and an example of a refactored description.

We are using early aspects for having had demonstrated a good alternative to deal with duplicated, tangled and scattered requirements. The Early Aspects [3] focuses on managing crosscutting properties at the early development stages of requirements engineering and architecture design using the aspect oriented paradigm [15].

This paper is structured as follows: Section 2 introduces the concept of refactoring opportunity. Section 3 presents the extract early aspect refactoring to manipulate requirement descriptions. Section 4 discusses related work. In Section 5, conclusions are discussed.

## 2. The Refactoring Opportunity

Fowler in [4] introduces bad smells as indications of deficiencies that can appear in existing software artifacts. The use of refactoring techniques might address the causes of these deficiencies. In this paper, we use the term refactoring opportunity to refer to the types of bad smells that can appear in requirements artifacts.

Refactoring opportunities should not be seen as exact rules to the automatic application of refactorings. The requirements engineer needs to decide about the

interest in changing the requirements and needs to choose which refactoring is more adequate for each opportunity.

In this section, we describe the duplicated, tangled and scattered requirements refactoring opportunity as well as associated refactoring. A brief textual description and an explanation of "when" to use the refactoring that could help in the solution of the identified problem are provided. The Extract Early Aspect refactoring is described in more detail in section 3.

### 2.1. Duplicated Requirements

A duplicated requirement is a situation that occurs when (i) the same requirement is duplicated in different places in a requirements document or (ii) the same requirements or the same pre-post conditions appear in several requirements structures.

A requirement that appears in more than one place offers an opportunity for refactoring. In the simplest case, the requirement is duplicated in one requirements structure. A possible solution is to use the Extract Early Aspect refactoring (Section 3.1), to remove the duplication and to create a new Early Aspect expressing this specific set of requirements.

Another common duplication problem occurs when an individual requirement appears in several requirements structures. This duplication could be removed using Extract Early Aspect refactoring as mentioned above. If the requirements are similar but not exactly the same, we may need to separate the duplicated piece from the rest of the requirement.

### 2.2. Tangled Requirements

A tangled requirement is a situation that occurs when a requirements unit contains descriptions of several properties or different functionalities [15].

A requirement unit that contains several different properties could be hard to understand. Thus it offers an opportunity for refactoring. A possible solution is to use the Extract Early Aspect refactoring (Section 3.1), to create a new Early Aspect that expresses specific and separated set of requirements. This solution needs to be analyzed with caution, because it may increase the number of the units.

Another possible solution is to use the Move Activity [14] (see more details in Section 4). With this solution requirement units are not created, they are just grouped in existing units that already express the same requirements. Thus, this second solution could be an alternative when the first, with aspects, were so expansive.

An example of Move Activity refactoring is showed in Figure 1. In the Order Processing System [19] described with use case, consider a Login use case that is concerned with user authentication and also with the functionalities selected by the user (two properties in the same unit). The shaded lines show the activities that could be moved to another use case, responsible for the functionality selection flow.

| **Main flow of events:** | … |
|---|---|
| 1. The use case starts when the user starts the application.<br>2. The system will display the Login Screen.<br>3. The user enters a username and password.<br>. . .<br>6. The system will display the Main Screen<br>7. The user will select a function<br>… | 8. While the user does not select Exit loop<br>9. If the user selects Place Order then<br>Use Place Order.<br>10. else if the user selects Cancel Order then<br>Use Cancel Order.<br>11. . . .<br>end if.<br>16. The user will select a function.<br>end loop.<br>17. The use case ends. |

**Fig. 1.** Login use case.

### 2.3. Scattered Requirements

A scattered requirement is a situation that occurs when the specification of one property is not encapsulated in a single requirements unit [15].

When the requirements that treat of the same goal are disperse by the requirement document the general localization is more difficult harder and in consequence difficult they reuse and maintenance, this situation could be solved with the Refactoring application. A possible solution is to use the Extract Early Aspect refactoring (Section 3.1) to create a new requirement unit that encapsulates a specific requirement that is scattered and then use the *Move Activity* (see more details in Section 4) refactoring [14] to put all scattered pieces in this new requirement unit.

## 3. Refactoring Requirements

In this section we define and describe the requirements refactoring Extract Early Aspect. This refactoring was mentioned in the previous section to provide solutions to: i) scattered requirements, ii) duplicated requirements and iii) tangled requirements.

For the refactoring we provide a context, a solution, the motivation for the application of the refactoring, a set of mechanics to apply the refactoring and an example illustrating the application of the refactoring. We use the format recommended by Fowler in [4] and we add some figures and a description of the solution for each mechanism steps.

The solutions described in this section are abstract, therefore we have the intention of being able uses them in any situation that occurs "bad smells". Thus, to a requirement engineer to uses them, first he already must have chosen the early aspect oriented technique to later applying the solutions.

The example used is described using the Multi-Dimensional Separation of Concerns approach [8]. The requirements are represented in XML format.

### 3.1 Extract Early Aspect

**Context**. A set of duplicated information is used in several places. This could be better modularized in an early aspect. This refactoring is also used when a requirement is too large or contains information related to a feature that is scattered across several requirements and is tangled with other requirements.
**Solution**. Extract that information to a new requirement and name it according to the context.
**Motivation**. This refactoring should be applied when there is duplicated information that can be split into two or more new requirements. The duplicated information could increase the costs of the requirements document maintenance and the potential for errors insertion. Every time that a change is needed, it is necessary to modify all the places where the duplicated requirements appear. [20].

The use of aspect-oriented requirements engineering [15] provides a good mechanism to modularize these requirements that are scattered among several places in a requirements document. The use of aspectual requirements [15], [5] promotes better modularization of the requirements artifacts.
**Mechanics**. The following steps should be performed:
**1.** Create a new early aspect and name it.

Figure 2 shows in detail how to create a new early aspect. Also it is considered when does not have to create a new early aspect.

```
Verify IF exist a <Early Aspect> that encapsulate the
same segment of information that will be encapsulate in
the structure that desire to create.
 IF Yes then
   Verify IF exist the possibility of to use the
   <Early Aspect> existent to encapsulate the new
   information.
     IF Yes then
       The new structure <Early Aspect> isn't created;
     END_IF;
     ELSE
       The new structure <Early Aspect> is created;
     END_ELSE;
 END_IF;
 ELSE
   The new structure <Early Aspect> is created;
 END_ELSE;
```

**Fig. 2**. The detailing mechanism to create a new structure <Early Aspect>.

**2.** Select in the original requirement the information you want to extract. An Early Aspect Mining Tool like the EA-Miner [17], [18] may be used in this step. Figure 3 illustrate an example with 10 requirements structures that only in 3 had been found the information desired.

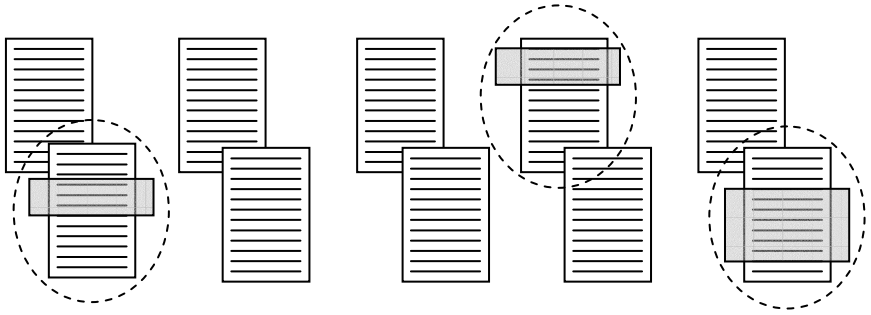

**Fig. 3**. Illustration of the requirements document and the selected segments.

Figure 4 shows in detail how to select in the original requirement the information you want to extract.

```
WHILE not be the end of the requirements document do
   To each structure of the requirement document:
     Verify IF exist the information you want to
     extract
     IF Yes then
        To select the segments that corresponds to the
        information desired;
        Go to the next structure;
     END_IF
     ELSE
        Go to the next structure;
     END_ELSE
END_WHILE.
```

**Fig. 4.** The detailing mechanism to select the information you want to extract.

**3**. Add the selected information to the new early aspect. Figure 5 illustrate the situation of the 3 selected information (in the previous step) to being added to the new early aspect.

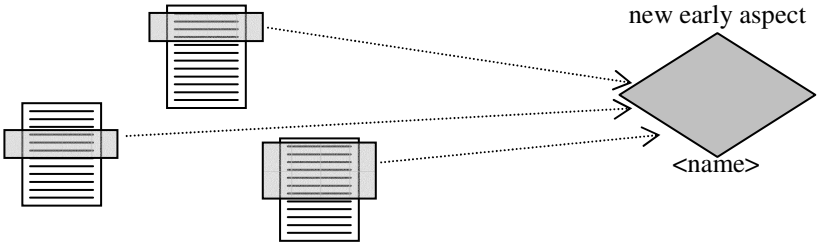


**Fig. 5.** Illustration of the addition of the selected information in the new early aspect.

To add the information in the new structure it is important to analyze the existence of a priority between the selected information. Figure 6 shows how to add the selected information in the new early aspect.

```
To all structures of the requirement document that
exists the information selected do:
   Verify IF exist a priority order:
   IF Yes then
      Order the selected information and do:
         To each structure of the requirement document
         that exists the information selected do:
            Verify IF the selected information already
            exists in the <new early aspect>
            IF Yes then
               Go to the next selected information;
            END_IF;
            ELSE
               Add (copy) the selected information to the
               new early aspect;
               Go to the next selected information;
            END_ELSE;
   END_IF;
   ELSE
      To each structure of the requirement document that
      exists the information selected do:
         Verify IF the selected information already
         exists in the <new early aspect>
            IF Yes then
               Go to the next selected information;
            END_IF
            ELSE
               Add (copy) the selected information to the
               new early aspect;
               Go to the next selected information;
            END_ELSE;
   END_ELSE;
```

**Fig. 6.** The detailing mechanism to add the selected information to the new early aspect.

**4**. Remove the selected information from the original requirement and collect information about the future composition. Figure 7 illustrate the selected information being removed of the original requirement and the collect of composition informations.

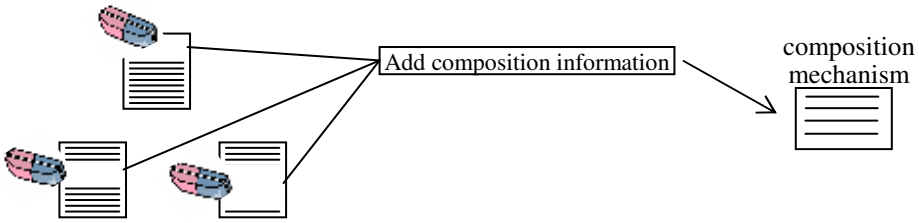


**Fig. 7.** Illustration of the removal of the selected information and the collects of composition guides.

All the information that need be collected is showed detailed in the Figure 8.

```
To each structures of the requirement document that
have information that was added to the new early aspect
do:
   To each selected information that was added to the
   new early aspect do:
      To make note of:
         - The name and/or the identification of the
         source structure (original requirement);
         - The localization inside of the structure (what
         it comes before and after);
         - The name of the destination structure (Early
         Aspect);
      Remove the select information;
```

**Fig. 8.** The detailing mechanism to removal the selected information of the original requirement.

**5.** Create the composition mechanism from the new early aspect to the original requirement(s). Using the collected information in the previous step is the composition mechanism is create. Figure 9 illustrate a situation in that the new early aspect has relationships with others 3 requirements structures.

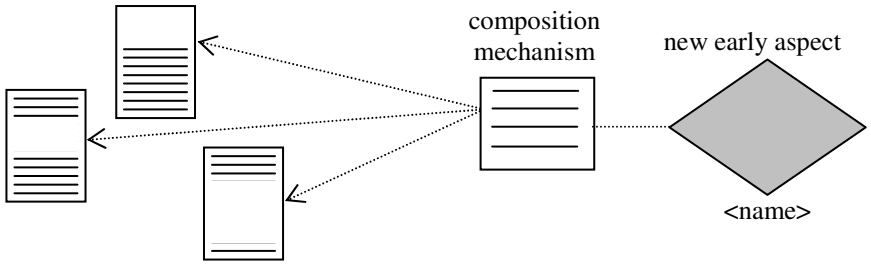


**Fig. 9**. Illustration of the creation of the composition mechanism.

```
In the composition mechanisms do:
  Provide pointers and/or rule with the following
  information:
    - The name and/or the identification of the source
    structure (original requirement);
    - Where the information will be introduced in the
    original requirement?
    - When the information will be introduced in the
    original requirement (before or after)?
```

**Fig. 10.** The detailing mechanism to create the composition mechanism.

**6**. Check if the original requirement is acceptable[1] without the removed information.
**7.** Update the references in dependent requirements to original requirements.
**Example**. The two excerpt requirements, shown in Figures 11 and 12, deal with the registration of a new employee and a new product, but we can see that the two initial requirements (1.1, 1.2 and 2.1, 2.2) in both figures are duplicated.

```
…
<Requirement id="1.1"> The employee provides the login
 and password. </Requirement>
<Requirement id="1.2"> The system validates the typed
 password. </Requirement>
<Requirement id="1.3"> The employee provides the
 following information about the new employee: Name,
 Login Password </Requirement>
<Requirement id="1.4"> The employee confirms the new
 employee register. </Requirement>
…
```

**Fig. 11.** Register new employee.

[1] Adequate for the original purpose.

```
…
<Requirement id="2.1"> The employee provides the login
 and password. </Requirement>
<Requirement id="2.2"> The system validates the typed
 password. </Requirement>
<Requirement id="2.3"> The employee provides the
 following information about the new product:
 Description, Size, Color, Location and Price.
 </Requirement>
<Requirement id="2.4"> The employee confirms the new
 product register. </Requirement>
…
```

**Fig. 12.** Register new product.

Applying the Extract Early Requirement to solve this duplicated requirements problem, we have the new Early Aspect shown in Figure 13 and the composition rules partially showed in the Figure 14.

```
…
<Requirement id="3.1"> The employee provides the login
 and password. </Requirement>
<Requirement id="3.2"> The system validates the typed
 password. </Requirement>
…
```

**Fig. 13.** Login early aspect.

Applying the step 4 (described in the refactoring mechanics) the duplicated information showed in the Figures 11 and 12 need to be removed. The composition Mechanism was created in agreement with the used approach MDSOC (Multi-Dimensional Separation Of Concerns [8]). Figure 14 shows the composition rules that maintain the original functionality to the Register New Employee and to Register New Product.

```
…
<Req.= "Register_New_Employee" id="1" children="include">
      <Constraint action="provide" operator="on">
           <Requirement concern="Login" id="3" children="include"/>
      </Constraint>
        <Outcome action="fulfilled"/>
  </Requirement>
…
<Req.= "Register_New_Product" id="2" children="include">
      <Constraint action="provide" operator="on">
           <Requirement concern="Login" id="3" children="include"/>
      </Constraint>
        <Outcome action="fulfilled"/>
  </Requirement>
…
```

**Fig. 14.** Composition rules (partial).

## 4. Related Work

Rui, Ren and Butler in [16] describe a meta-model for use case modelling and categorize a list of use case refactorings. Instead of simply listing the refactorings, we present a refactoring opportunity and a requirements refactoring definition, providing the context for the application of a given refactoring, a possible solution, motivation to perform the transformations and an example of the practical use of the described refactoring.

Yu, Li and Butler in [22] explain how refactoring can be applied in order to improve the organization of use case models. They focus on the decomposition of a use case and the reorganization of relationships between use cases. They also describe ten refactorings that could be used to improve the overall organization of use case models, such as inclusion or extension mechanisms introduction, use case deletion or refactorings that manipulate the inheritance tree. While Yu et al. focus on refactoring the use case models, we focus on refactoring requirements descriptions. As a consequence, our refactorings are finer grained than theirs. We also discuss in details the mechanics of the refactoring and possible refactoring opportunities in the context of requirements descriptions.

Xu, Yu, Rui and Butler in [21] present a tool that helps in refactoring of use case models and use it to conduct a case study using an ATM application. While they focus on the automation side of the use case refactoring process, we concentrate on providing definitions for deficiencies in requirements descriptions and refactorings in this context. Further research is needed to attain automatic identification of problems as proposed in this paper.

Ramos et al in [14] describes a collection of refactoring opportunities that might occur in requirements and provides a collection of refactorings that can be used to improve the quality of requirements where these opportunities appear. A collection of

examples for practical use for both refactorings and the identification of refactoring opportunities are described. In this work we are describing a refactoring in the early aspects context. Below (Figure 15) we are showing the Move Activity refactoring that it is cited in this article. Is important to note that the example of the application to this refactoring appears in the section 2.2, Figure 1.

**Move Activity**

**Context**. A set of activities is better accommodated in another requirement description.

**Solution**. Move the activity to the desired requirement. If the requirement does not exist yet, create a new requirement with the selected activities using the Extract Requirement refactoring.

**Motivation**. This refactoring is done to improve modularity and to ameliorate the balance of activities among the requirements. Activities are moved from one requirement to another also when a new requirement is created, either manually or by an Extract Requirement refactoring. This improvement in modularity could lead to a better understanding of the system in the long term [20].

**Mechanics**. The following activities should be performed:

1. Select the activities you want to move.
2. Move them to the desired requirement.
3. Update references to these activities if needed.

**Fig. 15.** The move activity refactoring [14].

## 5. Conclusions

This paper describes the refactoring opportunities that might occur in requirements and provides an Extract Early Aspect refactoring that can be used to improve the requirements where these opportunities appear. We describe an example for practical use for both refactoring and the identification of refactoring opportunity.

It is important to note that refactorings are usually described as two-way transformations and depending on a given situation one direction is more convenient than the other.

This work is part of a project whose objective is to evaluate the requirements document quality, finding pieces that can be improved with the application of requirements patterns and refactorings [11], [12], [13] and [14].

The refactoring opportunities that we have identified have the goal of making the requirements more understandable and to improve overall organization of the project. But depending on the system quality attributes considered (for example: reusability, maintainability, etc) the requirements engineer might take actions that slightly differ from the guidelines proposed here. Also, the expected granularity of the requirements descriptions might influence the use of the refactoring opportunity. The requirements engineer should take these issues into consideration.

## References


1. Boehm, B.W., Sullivan, K.J.: Software economics: a roadmap. In: ICSE - Future of SE Track. 319-343 (2000)
2. Elssamadisy, A., Schalliol, G.: Recognizing and responding to bad smells in extreme programming. In: Proceedings of the 24th International conference on Software Engineering (2002)
3. Early-Aspects.net. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. http://www.early-aspects.net/
4. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: improving the design of existing code. Object Technology Series. Addison-Wesley (2000)
5. Jacobson, I.: Use cases and aspects-Working seamlessly together. Journal of Object Technology 2(4) (2003)
6. Jacobson, I., Ng, P.W.: Aspect-Oriented Software Development with Use Cases. Addison-Wesley (2005)
7. Li, L.: A semi-automatic approach to translating use cases to sequence diagrams. In: TOOLS '99: Proceedings of the Technology of Object-Oriented Languages and Systems, Washington, DC, USA, IEEE Computer Society, 184 pg (1999)
8. Moreira, A., Araújo, J., Rashid., A.. A Model for Multi-Dimensional Separation of Concerns in Requirements Engineering. The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), 13-17 June 2005, Porto, Portugal. Lecture Notes in Computer Science, Volume 3520 (2005)
9. Opdyke, W.F.: Refactoring object-oriented frameworks. PhD thesis, University of Illinois at Urbana-Champaign (1992)
10. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw-Hill (2005)
11. Ramos, R.A., Araújo, J., Castro, J., Moreira, A., Alencar, F., Silva, C.: "Uma abordagem de instanciação de métricas para medir documentos de requisitos orientados a aspectos", in: 3º Brazilian Workshop on Aspect Oriented Software Development - WASP2006. Florianopolis, Brazil (2006)
12. Ramos, R. A.; Araújo, J. ; Moreira, A. ; Castro, J. ; Alencar, F. and Penteado, R.: "Um Padrão para Requisitos Duplicados", in: 6th Latin American Conference on Pattern Languages of Programming (SugarLoafPlop'2007), Porto de Galinhas, Recife, Pernambuco , Brazil (2007a)
13. Ramos, R. A. ; Alencar, F. ; Araújo, J. ; Moreira, A. ; Castro, J. and Penteado, R.: "i* with Aspects: Evaluating Understandability", in: Workshop on Requirements Engineering, 2007, Toronto, Canada (2007b)
14. Ramos, R., Piveta, E., Castro, J., Araújo, J., Moreira, A., Guerreiro, P., Pimenta, M., and Tom Price, R.. Improving the Quality of Requirements with Refactoring. In: VI Simpósio Brasileiro de Qualidade de Software – SBQS2007, Porto de Galinhas, Recife, Pernambuco, Brasil, Junho 27 – 30 (2007c)
15. Rashid, A., Moreira, A., Araújo, J.: Modularisation and composition of aspectual requirements. In Aksit, M., ed.: Proc. 2nd Int' Conf. on Aspect-Oriented Software Development (AOSD-2003), ACM Press 11-20 (2003)
16. Rui, K., Ren, S., Butler, G.: Refactoring use case models: A case study. In: International Conference on Enterprise Information Systems (ICEIS) (2003)
17. Sampaio, A., Loughran, N., Rashid, A., Rayson, P.: Mining Aspects in Requirements. In: Workshop on Early Aspects. Held with the 4th International Conf. on Aspect-Oriented Software Development AOSD 2005 (2005)

18. Sampaio, A., Rashid, A., Chitchyan, R., and Rayson, P. EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering. Transactions on Aspect-Oriented Software Development: Special Issue on Early Aspects (Accepted to Appear). Editor(s): J. Araujo, E. Baniasaad. Springer (2007)
19. Schneider, G., Winters, J.P.: "Applying Use Cases: A Practical Guide", in: Addison Wesley - Object Technology Series (1998)
20. Sommerville, I.: Software Engineering, 7th edition. Pearson Education (2004)
21. Xu, J., Yu, W., Rui, K., Butler, G.: Use case refactoring: a tool and a case study. In: Software Engineering Conference, 2004. 11th Asia-Pacific. 484-491 (2004)
22. Yu, W., Li, J., Butler, G.: Refactoring use case models on episodes. In: Automated Software Engineering, 2004. Proceedings. 19th International Conference on. 328-335 (2004)