

Traceability and AOSD: From Requirements to Aspects

Janet van der Linden, Robin Laney, Pete Thomas
Centre for Research in Computing
The Open University
{j.vanderlinden; r.c.laney; p.g.thomas}@open.ac.uk

Abstract

The traceability question is addressed through the development of a framework to go from requirements to aspects using the Design by Contract methodology. It is demonstrated that by starting at the requirements stage, and specifying an early aspect in the same semi-formal language as the system's existing requirements, we have the basis from which to design the aspects at the implementational stage. The language of pre- and post-conditions is shown to match closely that of aspects, in that pre-conditions match the aspect's pointcut, and the post-condition matches the advice part of the aspect. This thus gives us traceability from 'early aspects' to 'late aspects'. This approach will shed some light on the relationship between requirements and their refinements to pre- and post conditions, and the traceability of requirements in the face of reuse over time. The addition of a new crosscutting requirement is investigated in terms of the framework, demonstrating the relationship between early and late aspects and traceability. The framework promises to help with the design of the late aspects.

We propose the concept of relevancy: information in a requirement beyond its specification as pre- and post-conditions, as a way of identifying join points.

1. Introduction

In this position paper we work through a case study of an existing system containing legacy code. This is a legacy system which is well designed and where Design by Contract [10, 11] is used throughout. The requirements of the existing system are documented in the form of pre- and post-conditions.

We now want to add new functionality to the system. That is, a new requirement is identified. As it turns out, this new requirement crosscuts existing ones, and hence it makes sense to apply Aspect Oriented Software Development (AOSD) techniques.

Against the background of this case study, we will explore a technique of using pre- and post-conditions as a way of bridging the gap between existing requirements, additional crosscutting requirements and the design of aspects.

In this paper we will begin by setting out our position in the form of a framework for traceability from requirements to aspects, and from legacy requirements to new requirements. We will then discuss the related literature in this area. We will demonstrate our approach using the above mentioned case study, and conclude by reflecting on how our findings sit within the general discussion about early aspects, the traceability of aspects and the reuse of requirements.

Traceability can be defined as the degree to which a relationship can be established between two or more artefacts of the development process. The overall aim of traceability is to facilitate understanding by relating an artefact to its previous and next representation, augmented with information about design decisions taken [5].

To scope the remit of this paper we have only concerned ourselves with the introduction of new requirements that crosscut existing requirements. Furthermore, in terms of traceability we are particularly interested in the question of how an artifact is related to artifacts in the previous or next stage of development, but not with the question of how best to record additional information regarding the design decisions taken. Finally, when discussing 'aspects' we assume an AspectJ type language, and have not further investigated how this could be generalized to other representations.

2. Traceability from requirements to aspects.

As the starting point for our framework we assume we are dealing with a system with properties as shown in Figure 1.

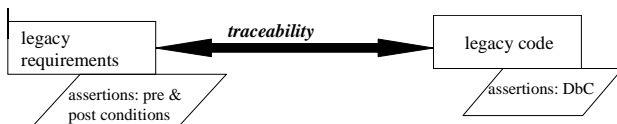


Figure 1 Traceability from legacy requirements to legacy implementation.

That is, we assume we are dealing with a legacy system, containing legacy requirements and legacy code. There is good traceability between these two sets of artefacts. The legacy requirements are described using assertions in the form of pre- and post-conditions. The legacy code contains assertions as we might expect in a system designed with the Design by Contract methodology. We also assume that it is clear how the assertions on the requirements' side are related to the assertions on the implementation side.

If the system needs to be augmented with additional functionality using aspects, it may be tempting to add such aspects directly to the legacy code. However, this could lead to the loss of traceability between requirements and code. We therefore propose the approach shown in Figure 2.

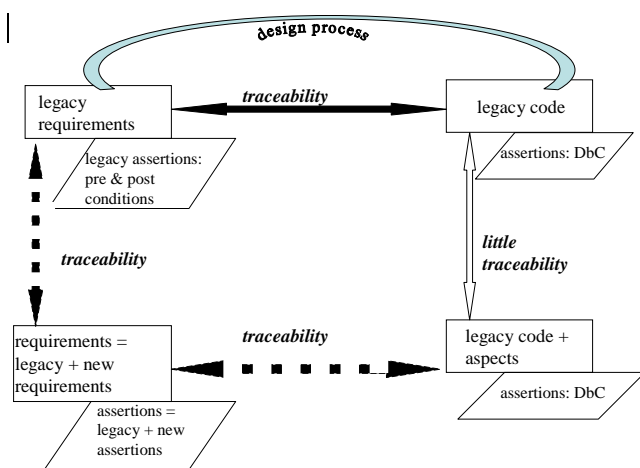


Figure 2 Traceability from requirements to aspects.

In Figure 2 it is shown how instead of going straight from legacy code to aspects (with loss of traceability)

we can start at the requirements side. The additional functionality is modeled as an early aspect specified in the same semi-formal language as the legacy requirements, using pre- and post-conditions. The new assertions for this early aspect then form the basis for the design of the late aspects that modify the legacy code.

The traceability from the early aspects to the implemented code comes about because there should be a close matching between the pre- and post-conditions for the requirement and the final realization of the aspect, where pre-conditions match closely the pointcut for the aspect, and the post-condition matches the advice part of the aspect.

As we shall explain, the early aspect defining the new requirement cannot specify completely the pointcuts for the late aspects, which depend upon the approach taken in the design of the legacy code. However, it may be the case that the traceability between legacy requirements and legacy code defined by the relationship between the two sets of assertions may help to bridge this gap.

3. Related literature

In the AOSD community the initial emphasis was purely on aspects at the language level, which we refer to as 'late aspects' to indicate that they are the aspects in the later stage of the software development process. More recently we see a move towards 'early aspects' descriptions, that is, the crosscutting that may occur at the level of requirements. There is an increasing level of interest in linking the two [2, 3], and to show how crosscutting requirements can be traced through other artefacts in the software development lifecycle to that of their actual realization as aspects. In their overview of early aspects approaches Bakker et al conclude that traceability remains very much an open research question [3].

Examples of early aspects approaches are AORE [2] and Theme/Doc [2]. The AORE approach is mainly concerned with the identification of crosscutting concerns at the requirements analysis level, and the specifying and evaluating of these concerns. Identification of such concerns at an early stage will help to detect conflicts between concerns, and identify trade-offs which can then be negotiated with the stakeholders early on in the software development life cycle. Theme/Doc provides support for the detection of early aspects in requirements documentation, by

identifying sets of actions that then help identify crosscutting behaviors.

There has been an interest in the Design by Contract approach in connection with aspects. In fact, the early paper by Kersten and Murphy, on the ATLAS project, has already identified an aspect ‘style rule’ for the introduction of aspects: the before and advice of an aspect should not alter the pre- and post conditions of the method into which it is introduced [7]. To this day, the issue of the interaction between code and aspects is a much discussed topic, often in terms of DbC [8].

CONA was developed as a tool to support DbC for objects and aspects, and enables the developer to classify aspects into *agnostic*, *obedient* or *rebellious*, depending on what their effects are on object contracts [9]. In [6] aspects are classified as *observers* or *assistants*, again depending on their effects on the contract in place. A module has to explicitly state whether or not it accepts the assistance of an assistant aspect. The overall aim behind this approach is to maintain modularity, since here the introduction of aspects does not lead to ‘whole-program analysis’ as is usually the case. A very different angle is found in those approaches where aspects are actually used in order to *ensure* DbC – i.e. by checking that pre-conditions are always checked.

A superimposition [13] is a module that can augment an underlying base program. It includes the specifications for both the assumptions about the basic systems to which the superimpositions can be applied, and the added properties of the resultant augmented program. These specifications are used to define proof obligations for correctness of superimpositions and to check feasibility of combining superimpositions to obtain new ones.

In this paper we look at the specification of requirements in terms of pre- and post-conditions. We introduce additional crosscutting requirements in the form of early aspects. These early aspects can also be specified using the formal language of pre- and post-conditions. By analyzing aspects in terms of pre- and post-conditions, we make a connection between requirements and aspects, or ‘early’ and ‘late’ aspects. This separation of requirements’ descriptions and the implementation of aspects is not dissimilar to the Join Point Designation Diagram approach which models pointcuts at the design level [14]. In this paper we also make a connection between old requirements and new requirements, thus bringing about traceability in both a horizontal and a vertical dimension as illustrated in Figure 2.

4. Case Study: Bookstore customers

The case study is of a system for dealing with the online ordering of books. Customers can browse the catalogue, place an order, go to the checkout to pay, or pay an outstanding bill. Customers are divided into three categories: *individuals*, *educational establishments* and *corporate clients* as shown in Figure 3.

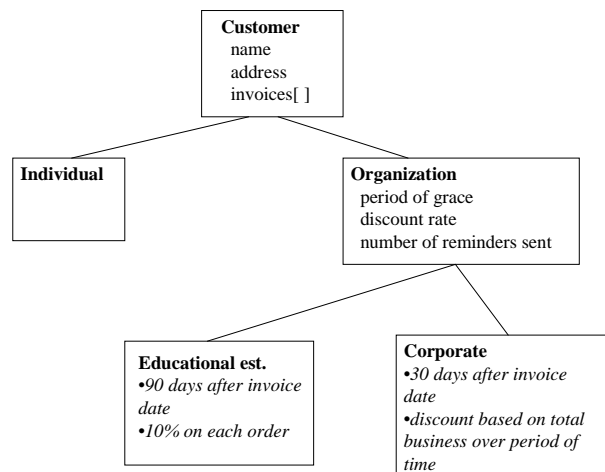


Figure 3 Different categories of customers

One of the important business events for the bookstore is that of a ‘received payment’ [12]. A number of business rules exist which reflect the store’s policies regarding the granting of *discount rates* and the length of the *period of grace* to settle payment. These policies differentiate between three categories of customers as follows:

1. Individuals pay by credit/debit card when ordering books, and only when payment has been received are books dispatched.
2. Educational establishments are given a 90 day period of grace to settle payment on orders received.
3. Educational establishments are given a 10% discount on each order.
4. Corporate customers are given a 30 day period of grace to settle payment for orders received.
5. The discount for corporate customers is based on the total amount of business over a period of time.
6. When no payment has been received and the period of grace has lapsed, a customer is sent a reminder.

- When payment has still not been received after an 'alert period', the credit department shall be alerted.

Most of the above requirements are related to the concern of 'non-receipt of payment', another business event that leads to a misuse case related to the 'received payment' event mentioned earlier [1]. That is, the bookstore has devised strategies to check that customers pay in time, and ways to raise the alert if they don't.

The informally described requirements have been brought together in the more formal description of the *raiseNonPaymentAlert* requirement, using the pre- and post-conditions notation for the misuse case that deals with the system's response to the 'non-receipt of payment' event:

```
raiseNonPaymentAlert
pre: (invoice has been sent &
      no payment received &
      invoice.date < (currentDate - periodOfGrace))

post: (!reminderSent &
       sendReminder())
      OR
      ((reminder.date < currentDate - alertPeriod)
       & alertCreditDepartment())
      OR
      true // no action to be taken
```

In other words, the concern that payment has not been made is only raised when an invoice has been sent, and no payment has been received – the period of grace has expired. This is the precondition for the misuse case. Once this condition is satisfied a whole plethora of activities occur to deal with the non-receipt of payment event as expressed in the post-condition. In the case where no reminder has been sent yet, the 'send-reminder' operation is put into action. If a reminder had already been sent, and the alert period has also expired, the credit department will be alerted. The rule that a reminder will be sent before the credit department is alerted, is part of the misuse case processing.

4.1. New requirement: good and bad customers.

We now suppose that a new initiative is launched as the store aims to improve the promptness with which customers settle payment for their orders. A number of incentive schemes as well as punitive measures are

introduced which aim to achieve this. As part of this goal a new requirement arises:

To be able to distinguish between 'good' and 'bad' customers.

Good customers are those customers who have always paid up promptly, whereas bad customers have given the store cause for concern as the credit department has had to become involved at some stage to extract payment.

Bad customers will see their discount rate lowered and period of grace shortened. The bookstore will also use a different type of invoice for the invoicing of such customers.

Corporate and educational clients can have their status changed back from bad to good, with different business rules for each category: corporate clients can regain the 'good' status after a positive check by the credit department, whereas for educational establishments the start of a new financial year is sufficient ground to change their status.

The distinction between good and bad customers is mainly relevant for the 'organization' customers, because the individual customers always pay by credit/debit and hence cannot get behind with their payment. However, the store would like to be able to reward the latter when they have placed a certain number of orders, by giving them a discount rate.

The different statuses granted to the original categorization of customers, can be shown in the form of a state chart where each new status is shown as a state, and business rules represent the transition rules to change from one state to another. This is shown in Figure 4.

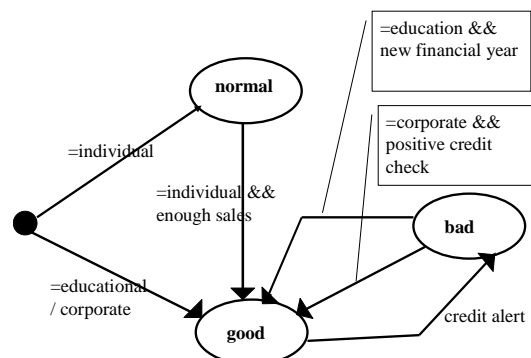


Figure 4 Good and bad customers: transitions.

Individuals would initially be granted the 'normal' status, and can climb to 'good' status if they have placed a certain number of orders. However, they cannot become 'bad'.

4.2. Crosscutting nature of new requirement.

The new requirement crosscuts the existing requirements, in a number of different ways:

- The original categorization into individual, corporate and educational customers is now crosscut with a new categorization of good, bad and normal. Note that the original categorization into 'individual /educational /corporate' is static and does not change over time, whereas the new categorization is dynamically changing.
- New discount arrangements crosscut with existing discount arrangements, and in particular, the individual customers can now acquire a discount rate which was not possible in the original scheme.
- The calculation of the lengths of the period of grace and the alert period, which was previously solely based on the categorization into educational or corporate, now also crosscuts with the rules associated with the good or bad status.

4.3. Some questions relating to crosscutting, traceability and Design by Contract.

The crosscutting nature of the new requirement fits the description of an 'early aspect': that is, it is an aspect at the requirements' stage, which is crosscutting existing requirements [2]. There is as yet no common formalism for describing such aspects and at this point we would normally be presented with the implementational view of the aspect, including its pointcut and advice.

One of our aims was to develop a technique that would take us from a cross-cutting requirements' description to the implementation in a traceable fashion. We are motivated to do this because we think that by starting from requirements, we avoid the problem of creating bad interactions between code and aspects.

Given that DbC is a well understood formalism for the design of software artifacts, we set out to explore DbC for the design of aspects, while asking the following question: Is it useful to describe aspects in terms of pre- and post conditions? Does such a description

deliver insight into the traceability between requirements and aspects? Is such a description able to guide us towards any join points with the original system? How closely does such a description match the final implementation of such aspects?

4.4. Early aspects – design.

The new requirement can be broken down into three early aspects:

- InitializeStatus aspect - in order to award the initial status to each category of good, bad and normal customers;
- ChangeStatus aspect – in order to be able to change the status of a customer from one status to another;
- RecognizeStatus aspect – in order to be able to treat customers in a different way given that there is a new categorization.

The specification of the changeStatusAspect aspect, using pre- and post-conditions is as follows:

ChangeStatus aspect

pre: (is individual customer
& has normal status
& made enough orders)
OR

(is organization customer
& has good status
& alertCreditDepartment())

post: (is individual customer
& has good status
& set DiscountRate to higher rate)
OR

(is organization customer
& has bad status
& reduce periodOfGrace)

etc.

The pre- and post-conditions for this aspect specify that an individual customer who has the 'normal' status but who has now made a certain number of orders, will have their status set to 'good' customer, and will be given a discount rate. If we are dealing with an 'organisation' customer, who currently holds a 'good' status, but has not settled their bill in time to the point that the credit department has had to be alerted, this customer will now be given the 'bad' status and their period of grace will be reduced.

Part of the aspect to recognize, and hence process, the fact that there is now a different categorization of customers is as follows:

RecognizeStatus aspect

pre : organisation has bad status

post : send special invoice

This aspect specifies that the pre-condition for the sending of a special type of invoice is that the customer has the 'bad' status.

5. Lessons learnt

In section 4.3 we formulated a number of questions that we hoped to reflect on during our work on the bookstore case study. Here we outline some of the lessons learnt.

5.1 Hypothesis about requirements, aspects, pre- & post-conditions and pointcuts.

An aspect at the implementational stage can be described as consisting of a pointcut and an advice. The pointcut specifies where in the base code we want the aspect to be applied, and the advice is the action to be undertaken at those specified points. It seems that the design for the aspect in terms of pre- and post-conditions comes quite close to this division into pointcut and advice: that is, the pre-condition closely resembles the pointcut, and the post-condition matches the advice part precisely.

In terms of DbC, an aspect's post-condition defines what the aspect is to achieve - provided the aspect's pre-condition is satisfied. The pre-condition is what must be satisfied if the aspect is to be executed.

Now the pointcut for the aspect defines a collection of join points within the legacy code where the aspect is to be executed. This implies that, at each join point, the aspect's pre-condition must be true. That is, the aspect's pre-condition specifies, in general, a number of points within the legacy code where the aspect could legitimately be executed. We shall call these 'potential join points'. However, not all such points would be appropriate. There needs to be some additional condition to discriminate between those potential join points where the aspect is or is not to be executed. We refer to this additional condition as a 'relevancy' condition.

For example, take the case of the requirement for the new style of invoicing of bad customers:

When dealing with 'bad' customers, they shall be sent different types of invoices. However,

we don't want to send such invoices continually, but rather we must make sure that this happens at appropriate places such as when we would have sent them an invoice under normal circumstances.

The pre-condition might indicate all the places in the code where we are dealing with such customers, and where we could 'potentially' decide to send them such invoices. The requirement specifies where we *want* it to take place.

So we could say that the *relevancy* or the *appropriateness* is expressed in the requirement, but not in the pre-condition. What's specified in the pre-condition is not specific enough.

5.2 Traceability and reuse of requirements.

In the beginning of this paper we outlined that the system we are dealing with is a legacy system, and contains legacy code. However, requirements themselves can also be viewed as legacy items and are therefore subject to the constraint that they should not be modified in an ad-hoc manner. The introduction of a new requirement in the way we did, in the form of early aspects are a way of partitioning a change from the legacy requirements. In other words, describing requirements in a 'formal' way (e.g. through pre- and post-conditions) enabling changes to be specified in a robust/clear/accessible way through a mechanism that is also 'formal' (i.e. aspects defined by pre- and post-conditions) has to be a 'good' thing. Effectively, this is a mechanism for partitioning - keeping the changes separate from the legacy requirements in a useful manner. Potentially, we then have traceability of both the legacy requirements and the changes (given as early aspects) through to the legacy code and the implementation aspects.

However, there is also traceability from the legacy requirements to the changes.

The technique we developed may also form a contribution to other problems associated with reuse, as discussed in [4]. When requirements are reused (and thus, by implication become part of the specification for a new product) they often lose any association with the original requirements. This means that, if the new requirements are changed (specifically the reused requirements are changed) because of some 'error', the original requirements specification may not be amended because of the lack of traceability.

With our new technique, any change to reused requirements could (should?) be specified via early aspects and could therefore be applied easily to the original requirements. Whilst this does not directly apply to keeping track of reused requirements, it does help in maintaining requirements that are replicated. Also, it is not just 'errors' that cause this need - one may, for example, have a product line where requirements are reused many times and a change to one line might be required in other lines.

6. Conclusion

We have outlined an approach based on design by contract by which cross-cutting requirements can be documented early in the life-cycle of a system development. By mapping these early aspects onto later, design aspects we ensure traceability between requirements, design, and implementation, as a system evolves. Furthermore, we can trace evolution through the requirements of a system. This ability to trace artifacts horizontally and vertically promises to improve the tractability with which aspects are applied.

In future work, we will concentrate on exploring the concept of relevancy as a way of tying join point definition to requirements, rather than code.

7. References

- [1] Alexander, I., "Misuse cases help to elicit non-functional requirements". *Computing and Control Engineering*, Feb 2003, pp 40-45.
- [2] Araujo J., Baniassad E., Clements P., Moreira A., Rashid A., Tekinerdogan B., "Early Aspects: The Current Landscape", 2005 [www.early-aspects.net]
- [3] Bakker J., Tekinerdogan B., Aksit M., "Characterization of Early Aspects Approaches", workshop on Early Aspects, AOSD 2005.
- [4] Bush, D. and Finkelstein, A. "Reuse of Safety Case Claims – An Initial Investigation." London Communications Symposium, University College, London, 2001.

www.ee.ucl.ac.uk/lcs/prog01/LCS035.pdf

(Accessed 15 Jan 06)

- [5] Chitchyan R. et al, "Survey of Aspect-Oriented Analysis and Design". AOSD-Europe Project Deliverable No: AOSD-Europe-ULANC-9. Editor(s): R. Chitchyan, A. Rashid, 2005.
- [6] Clifton C., Leavens G.T., "Observers and Assistants: A Proposal for Modular Aspect-oriented Reasoning". Technical Report 02-04a, Iowa State University, Department of Computer Science, April 2002.
- [7] Kersten M.A., Murphy G.C. "Atlas: A Case Study in Building a Web-based Learning Environment using Aspect-oriented Programming". Technical Report Number TR-99-04, Department of Computer Science, University of British Columbia, Canada, 1999.
- [8] Kiczales, G., and others, discussion thread "Aspect Interactions", discuss@aosd.net, Nov 2005
- [9] Lorenz & Skotiniotis, "Extending Design by Contract for Aspect-Oriented Programming", Technical Report NU-CCIS-04-14, College of Computer & Information Science, North Eastern University, Boston, 2005.
- [10] Meyer, B., *Object-oriented Software Construction*, 2nd ed., Prentice Hall, 1997.
- [11] Mitchell, R., McKim, J., *Design by Contract, by Example*, Addison Wesley, Boston, 2002.
- [12] Robertson, S. and Robertson, J. *Mastering the Requirements Process*. Addison Wesley. 1999, ISBN 0 201 36046 2.
- [13] Sihman M., Katz S., "Superimpositions and Aspect-oriented Programming", *The Computer Journal*, 46(5), 2003, pp 529-541.
- [14] Stein D., Hanenberg S., Unland R., "Modeling Pointcuts", in Workshop on Early Aspects: Aspect oriented Requirement Engineering and Architecture Design, held in connection with AOSD conference, 2004.