Requirements Engineering and Aspects

Yijun Yu¹, Nan Niu², Bruno González-Baixauli³, John Mylopoulos², Steve Easterbrook², and Julio Cesar Sampaio do Prado Leite⁴

¹ Department of Computing, The Open University, UK
y.yu@open.ac.uk
http://mcs.open.ac.uk/yy66
² University of Toronto, Canada
³ University of Valladolid, Spain
⁴ PUC-Rio, Brazil

Abstract. A fundamental problem with requirements engineering (RE) is to validate that a design does satisfy stakeholder requirements. Some requirements can be fulfilled locally by designed modules, where others must be accommodated globally by multiple modules together. These global requirements often crosscut with other local requirements and as such lead to scattered concerns. We explore the possibility of borrowing concepts from aspect-oriented programming (AOP) to tackle these problems in early requirements. In order to validate the design against such early aspects, we propose a framework to trace them into coding and testing aspects. We demonstrate the approach using an open-source e-commerce platform. In the conclusion of this work, we reflect on the lessons learnt from the case study on how to fit RE and AOP research together.

1 Introduction

Aspects are features of a software system that cut across architectural boundaries and impact on the design of multiple system components. Examples of aspects include data persistence and logging, security, performance, usability, maintainability, monitorability, testability, etc. Aspect-oriented programming (AOP) modularizes crosscutting concerns in software systems so that their evolution is less problematic [1, 2]. One may apply AOP to reengineer a legacy software system, thereby obtain a library of aspects from an implementation that can be further configured and reused. Thus, uncovering aspects can be supported by program analysis techniques such as aspect mining [3], refactoring [4], and program visualization [5]. A common limitation to these techniques is that it is hard to validate identified and modularized code aspects: Were they really required or designed to be there?

Early aspects – the concerns that crosscut during the early stages of a software system’s life cycle [6] are useful to study and analyze early on, before they are transformed into myriads of details that clutter code artifacts [7, 8, 9, 10, 11, 12].

Although various approaches have been proposed to discover and modularize aspects at the requirements level [13], few address validation and traceability issues of identified early aspects. Specifically, it is of great interest to trace how early aspects manifest

themselves and evolve throughout the software life cycle, and to validate the resulting system in light of stakeholder concerns that crosscut the problem domain.

In our previous work [9][11], we developed a framework for identifying and weaving candidate aspects during goal-oriented requirements analysis [13]. Aspects in goal models are captured as the operationalizations of softgoals and the relations to functional goals. These may be implemented as code aspects, but developers may choose other means to address these crosscutting concerns. Even in the latter case, it is desirable to keep early aspects modularized so that one does not have to recover them from the code at a later stage.

In this work, we investigate how to trace and validate identified early requirement aspects resulted from [9] through implementation and testing. Early aspects are either naturally mapped to code aspects, or recorded as issues to directly advise testing. Aspect testing is therefore guided by stakeholder goals. In our approach, aspects are intended to enhance system qualities by interacting with multiple system units, while preserving the functionalities defined by hard goals. The benefits of leveraging our approach are twofold. By separating crosscutting concerns throughout requirements, implementation, and testing phases, we achieve a high degree of modularity and traceability in software development. By validating implementation against stakeholder concerns, we achieve a high level of software quality and user satisfaction.

Our aim for this work is to explore the extent to which system implementation helps validate early aspects, and to propose a systematic means to trace the refinement of requirement aspects throughout the software life cycle. Our proposed approach to handling aspects is very general and does not depend on any specific programming language or particular type of applications. To demonstrate the idea, we present a study showing the approach’s application to an open-source e-commerce platform written in PHP – osCommerce[2]. Our study effectively validates broadly scoped quality concerns (e.g., usability) against stakeholder requirements, systematically enables the tracing of aspects throughout the software life cycle, and empirically verifies improved system modularity claimed by aspect orientation.

The remainder of the chapter is structured as follows. Section 2 explains generally why stakeholder requirements contain early aspects, and Section 3 introduces one form of early aspects systematically discovered from goal-oriented requirements engineering. Section 4 shows some existing solution to the problem and their relation to us. Section 5 explains our approach in detail, by showing the syntax of our early aspect language and the PhpAspect language[3], explaining how to validate the traces between early aspects and code aspects using testing. Section 6 illustrates the application of the approach using the osCommerce case study extended from that of [9]. Reflecting on the lessons learnt, Section 7 concludes the paper and suggests future work.

---

1. Yijun Yu et al.
2 Why Aspects in Requirements?

This section aims to address the research question: why do we need aspects in requirements? It situates our research within existing literature in Requirements Engineering (RE) and Aspect-Oriented Software Development (AOSD).

A requirement aspect is a stakeholder concern that cuts across other requirements concerns or artifacts [14]. It is broadly scoped in that it is found in and has an implicit or explicit impact on more than one requirements artifact [6].

Requirement aspects convey domain properties of interest to stakeholders, interact with base requirements dictate additional services, and need to be traced to stakeholder interests. For example, a performance requirement cuts across all functional requirements to deliver a “fast enough” computation. With respect to different functionalities, however, such performance requirement can be operationalized differently. For example, a “high-performance sort” function is required to have $O(n \log n)$ time complexity where the size of the problem, $n$, is large. On the other hand, a high-performance “enter keyword” function is not required to finish in milli-seconds: Most of elapsed time the computer is waiting for user input, thus more tolerable to computational complexity. Nonetheless, the overall system performance requirement depends on all functionality to be fast enough. Therefore we can regard the performance concern even a globally scoped aspect where all functional requirement artifacts are affected. Other requirements, such as security, usability, etc., that share the same characteristics are all requirement aspects. Once woven, requirement aspects may also become base requirements for other requirement aspects. For example, a usability requirement aspect can crosscut the previous example “high-performance sort”, requiring a progress bar to assure end-users about how much progress the algorithm is moving forward. Base requirements include functional requirements as well as those woven with other requirement aspects, hence an inevitable problem is to validate them. The effectiveness of a requirement aspect can only be validated by considering base requirements that they are woven. For example, one may compare one sort algorithm with another sort algorithm in terms of measurable performance, rather than comparing one performance problem to do with sorting with another problem to do with entering password. It is therefore useful to instantiate the validation to the places where one can trace to the woven requirements.

Although in AOP (aspectJ-like), aspects are typically considered to be solutions that crosscut existing base solutions. In RE, such is not always the case. This is due to the fact that a base problem does not always exist before the crosscutting ones. Reflecting different viewpoints of a problem, early aspects can, in the extreme, crosscut each other symmetrically. In that sense, viewpoint merge/matching techniques in RE [15] can also be considered an early aspect weaving technique.

To interpret the notions involved in aspect orientation from an RE perspective, we follow the metaphor that every requirement aspect acts as a service provider to some base modules [16].

- **Advice** defines the content of the service that a specific requirement aspect provides. It describes what the service is about.

- **Join points** are points in the base which a requirement aspect interacts with. They describe where the service is provided.
- **Pointcut** represents a set of join points. It describes the *situational patterns* of the service that an aspect provides.
- **Weaving** is the process of coordinating service providers (requirement aspects) and consumers (base requirements). It describes *when* and *how* the service takes place.

In addition, the purpose of a requirement aspect describes *why* the service is needed in the first place. Since it is hard, and usually counterproductive [14], to carry out aspect-oriented RE activities on requirements documents that are ill-structured, existing approaches fall into two categories.

The first category is based on linguistic clues where the scattered occurrences of terms in the document are considered as crosscutting concerns, and their relations to the terms in the context are considered as join points [6]. Information retrieval techniques [17] and natural language processing techniques [18] have been proposed to identify these scattered occurrences and associate them to the non-functional requirements (NFRs) as aspects.

The second category is based on structured requirements models, such as goal graphs [9] or problem frames [19]. The crosscutting concerns on these structures are often derived from the different nature of functional and non-functional domains. For example, a “higher performance” task in the “performance tuning” domain tends to cut across other functions in the system, leading to a natural crosscut to system functions. Therefore, the structures of the models used by the goal-oriented RE can be exploited to explain how early aspects are discovered from the stakeholders, rather than mined from the code.

To sum up, Table 1 classifies some approaches on early aspects with respect to the concepts of AOP.

<table>
<thead>
<tr>
<th>Approach</th>
<th>candidate aspects</th>
<th>advices</th>
<th>join points</th>
<th>pointcuts</th>
<th>weaving</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8, 14, 15]</td>
<td>concerns</td>
<td>requirements</td>
<td>viewpoints</td>
<td>contribution table</td>
<td>resolving conflicts</td>
</tr>
<tr>
<td>[9, 10, 11]</td>
<td>softgoals</td>
<td>advising tasks</td>
<td>goals and tasks</td>
<td>contribution links</td>
<td>composing algorithm</td>
</tr>
<tr>
<td>[19]</td>
<td>security</td>
<td>vulnerabilities</td>
<td>functional req.</td>
<td>threat descriptions</td>
<td>composition process</td>
</tr>
<tr>
<td>[6, 17]</td>
<td>non-functional terms</td>
<td>occurrences</td>
<td>req. statements</td>
<td>indicator terms</td>
<td>information retrieval</td>
</tr>
</tbody>
</table>

In relation to the themes of this book, requirement aspects “interwine requirements with contexts” in the sense that requirements are allowed to be crosscutted by global concepts such as viewpoints, concerns, softgoals. Moreover, all these must be localised with business, organisational and community situations through contextual joinpoints. Requirement aspects also help “evolving designs and ecologies” because selectively

---

4 Please refer to the early aspects landscape reports (available at http://www.early-aspects.net) for more detailed surveys of aspect-oriented RE approaches.
freezing certain aspects while changing other aspect is realised easier by modularising
the crosscutting concerns among the requirements that can significantly reduce the
coupling of similar advices. Requirement aspects support “fluidity of designs, archi-
tecture, visualisation and representation” because a core architecture can be cleaned by
separating messy crosscutting concerns, which can greatly help visualising and repre-
senting the base system. Last but not least, requirement aspects fit the theme of “com-
plexity, business process focus, architecture” by hiding the complexity caused by tan-
gled requirements. Although requirement aspects make it much easier to manage cross-
cutting concerns among requirements, common to these themes, there is still a research
challenge to trace them into system implementations and validate them against stake-
holder goals. In the remainder of the chapter, we will focus on demonstrating a possible
answer to this challenge.

3 Discovering and Testing Aspects in Goals

This section introduces the terminology and concepts involved in our approach to dis-
covering aspects in goal models and testing their validity.

Organizational goals lead to requirements. Goals justify and explain the presence
of requirements, while providing the baseline for validating stakeholder concerns [13].
Goal modeling shifts the emphasis in requirements analysis to the actors within an or-
ganization, their goals, and the interdependencies between those goals, rather than fo-
cusing on processes and objects. This helps us understand why a new system is needed,
and allows us to effectively link software solutions to business needs.

Goals provide basic information for detecting and resolving conflicts that arise from
multiple viewpoints [20]. Goal analysis facilitates the discovery of trade-offs and the
search of the full space of alternatives, rather than a subset. Goal modeling frameworks
distinguish between hard (functional) goals – states that actors can attain – and soft-
goals, which can be satisfied only to certain degrees. System qualities, such as reli-
ability, efficiency, and portability, are typically expressed as softgoals to suggest that
intended software is expected to satisfy these NFRs within acceptable limits, rather
than absolutely [21].

Aspects in goal models can be discovered using the correlations from hard goals to
softgoals along with a goal eliciting and refinement process of a V-shape goal graph [8].

Figure 1 illustrates the process of separating early requirement aspects as a result of
goal elicitation. A V-shape goal graph is a simplified metamodel where only functional
and non-functional requirements are initially elicited as goals/softgoals respectively.
Then through vertical refinements, they are operationalized into tasks. The horizontal
contribution links between goals and softgoals provide a clue to separate tasks that
operationalize softgoals from the tasks that operationalize hard goals.

The formal requirement aspect elicitation process can be briefly explained as fol-
lows. Initially, the stakeholders’ high-level concerns are elicited as abstract goals. The
functional ones are represented by hard goals and the non-functional ones are repre-
sented by softgoals. Relations are also elicited as abstract contribution (resp. correla-
tion) links from the functional hard goals to the non-functional softgoals that must be
fulfilled by the prescribed system-to-be.
During the refinement process, these abstract goals are recursively decomposed into more concrete ones through AND/OR decomposition rules. As a result, several hierarchies of goal trees are derived. These are inter-related through abstract contribution (resp. correlation) links that represent semantic inter-dependencies between these goal trees.

At the end of the refinements, all abstract goals are decomposed into a set of goals that need no further decompositions. These leaf-level goals can be fulfilled by tasks that can be carried out by the system-to-be or external actors. The set of tasks are further categorized into functional ones and non-functional ones depending on whether they are at the bottom of the decomposition hierarchy of an abstract hard goal, or of an abstract softgoal. The refinement of abstract goals must be validated to maintain the abstract contribution links, which can often be fulfilled by weaving the concrete non-functional (operationalized) tasks into the functional tasks. As such, every OR-decomposed subgoal must fulfill the same commitment to the softgoals as their parent goal does. It is often the case, if not always, that non-functional tasks crosscut several functional ones that belong to different OR-decomposed subtrees.

Figure 2 illustrates early requirement aspects in the media shop study [23, 9]. The top level softgoals such as “Security [system]” and “Usability [language]” are captured as goal aspects, which are operationalized into advising tasks. The aspect weaving is achieved by composing the advising tasks with the functional tasks of effected hard goals. As an example, the aspect “Customization [language]” is operationalized into an advising task “Translate [language, NLS]”, meaning that the media shop is advised to translate occurrences of natural language strings (NLS) into the desired language. This advice crosses all hard goals that display Web pages and is intended to enhance system usability for native users of the desired language. Basic functionalities (e.g., “Informing”, “Reporting” and “Shopping”) defined by hard goals via functional tasks shall not be changed, though, by weaving such a usability aspect.

Aspect verification is an active area where research focuses on different ways to carry out tests, select test cases or generate test cases for a given AOP mechanism [24]. For example, unit testing aspects has been proposed by Lesiecki [25], equipped with eight patterns to verify the crosscutting behavior as well as the crosscutting specifications. The specifications correspond to the pointcuts in aspects, which are validated.
Aspects, together with their advising tasks, are represented as first-class modules in the upper-right corner of the goal model. The contribution links, from hard goals and functional tasks to requirement aspects – operationalized softgoals – modularize the crosscuts that would otherwise be tangled and scattered in the goal model.

Through aspectJ visualization and mock targets manually. The behavior is tested with a focus on the crosscutting functionality, i.e., advice.

In this work, we attribute the crosscutting behavior to both functional and non-functional requirements, and propose to reuse existing testing mechanisms to work out a natural way to validate the implementation of the identified early aspects. Since requirement aspects address non-functional requirements, the functionality of the base components must not be modified by weaving the aspects. Thus, existing unit test cases that target at the original base components can be reused for testing the functional behavior of the aspects.

In our approach, it is crucial to validate whether the non-functional behavior of an implemented aspect fulfills its intended NFR defined by the early goal aspect. Certain qualities in a system with weaved aspects must outperform the one without aspects, so that the effort of managing aspects can be justified. Such examinations are guided by the quality metrics derived from requirement aspects, and are supported by modularizing testing aspects [26], or via other means depending on the type of NFRs under investigation.
4 An aspect tracing and validating framework

In this section, we explain how tracing and validating requirement aspects are carried out throughout the software life cycle. Figure 3 overviews the process of our approach. The upper part of the figure highlights the early aspects discovery process discussed earlier. Advising tasks, which operationalize softgoals and relate to hard goals, are modularized as aspects and weaved into goal models to enable aspect-oriented requirements analysis.

Key concepts of AOP implementation are depicted in the middle of Figure 3. Functional modules \((f)\) and code aspects (advice + pointcut) are derived from functional and advising tasks respectively. The weaved system \((f \circ a)\) is obtained by composing advice \((a)\) with bases according to the pointcut description \((p)\). Some aspects identified at the requirements level may not be mapped to code at all. For example, a typical performance requirement might state that the system shall complete a task within 2 seconds. These early identified aspects play a key role in monitoring the system’s behavior, and shall not be lost in software development. We record them as quality issues to establish their traceability throughout the software life cycle, but the discussion of handling these quality issues is beyond the scope of this work. The success criteria for aspects are specified in \(t\), which gathers quality metrics and shares the same pointcut with \(a\). It is important to incorporate the metrics \(t\) so that one can measure system qualities with \((f \circ a \circ t)\) and without \((f \circ t)\) aspects. Note that our framework is viable for applying different programming languages and weaving mechanisms such as AspectC, aspectJ or HyperJ. We use phpAspect to illustrate our approach in the following discussion.

System validation is shown in the lower part of Figure 3. The weaved system \((f \circ a)\) is subject to two kinds of tests on basis of respectively hard or soft goals of stakeholders. The first test ensures that systems with and without aspects have the same functionality defined by hard goals: \(H(f) = H(f \circ a)\). Existing testing mechanisms, such as unit testing, can be reused to validate whether the weaved system satisfies the functional requirements. The second test checks whether the weaved system indeed improves system qualities in terms of the degree of softgoal satisfaction: \(S(f \circ t) < S(f \circ a \circ t)\). Our approach enables both forward and backward tracing of crosscutting concerns throughout the software life cycle.

4.1 Goal aspects in Q7

Q7, or 5W2H – why, who, what, when, where, how, how much – is a pseudo programming language that captures the structure of requirements goal graphs, including the major entities of the NFR framework. The syntax of the language is designed to facilitate the reuse of solutions in the non-functional domains while incorporating aspect orientation. The seven questions are usually asked for eliciting and elaborating goal-oriented requirements following the Tropos methodology.

The answers to the why and how questions respectively indicate the composition and decomposition relations between abstraction and implementation. Adapted from the goal model of Figure 2, the following example shows the AND/OR decomposition relations among the hard goals in the media shop domain. The front page of the shop has the functionality for “informing” the end-users and administrators. This goal
Fig. 3. A process overview of tracing and validating aspects

is decomposed into “finding” and “reporting” relevant information. In order to find information, a user is allowed to “search” or “navigate” the shop. The nesting structure of curly braces helps visualize the decomposition hierarchy of the goals.

The answers to the how much question show the degree of contributions between hard goals and softgoals. Q7 uses the labels “++”, “+”, “−”, and “−−” to indicate the “make”, “help”, “hurt”, and “break” relations between the goals. The answers to the what question connect the goal to its subject matter [28]. In Q7, such information is placed inside square brackets as topics of the goals or softgoals. For example, when the system meets the shop’s “Front [page]” goal, it also makes (++) major top-level softgoals (“⇒”), such as “Security [system]” and “Usability [language]”.

Front [page] { ...
  } ⇒ ++ Security [system],
    ++ Usability [language] ...
The answers to the when question indicate the feasibility of the goals under certain contexts [29], and those to the who question attribute a goal to an encapsulating module. In the i* terminology [22], such a module is called an actor that either processes or delegates a goal or a task to other actors via strategic dependencies. In Q7, we use the idea of “namespaces” to represent the actor names. For example,

```xml
<MediaShop>::Front [page] &
    Managing [page]
...
```

Here, “MediaShop” is the actor that processes the goal “Front [page]”. If the actor is not explicitly specified, a goal inherits the namespace from its parent goal. Thus, “Managing [page]” belongs to the same “MediaShop” actor.

As an extension of the encapsulating actors, we create a new namespace to modularize the aspect that cuts across multiple entities in the goal model. As an example, the security aspect in Figure 2 is represented as follows.

```xml
<aspect>::Security [system] &
    Confidentiality [credentials] <=+ [page] &
    Redirect [login]
|
    Security [information flow] <=+ [account] &
    SSL [connection]
|
```

The goal hierarchy within the aspect module is an advice and the leaf-level tasks in the hierarchy are called advising tasks. These tasks do not exist by themselves, since they have to be woven into the functional goals by indicating where to attach the advice. The answers to the where question are designed to express the pointcut of an aspect, indicating which functional goals are suitable for applying the advice. For example, the following Q7 statements show a pointcut expression after the “<=+” symbol: + * [page], which matches the hard goals of any name (indicated by the wildcard *), of the subject matter Web “page”, and those helping (+) achieve the usability soft-goal. The advising task translates the “natural language string” (NLS) appeared in the Web page into the desired language (e.g., Spanish or German). Note that a pointcut can also be specified by enumerating the effected hard goals.

```xml
<aspect>::Usability [language] &
    Customization [language] <= + * [page] &
    Translate [language, NLS]
|
```

All matched goals are therefore the join points of the aspect. A weaving algorithm [27] has been implemented in the OpenOME modeling tool (http://www.cs.toronto.edu/km/openome) to identify the join points and attach the advising tasks as siblings to the join point tasks. Both join point tasks and advising tasks then share the same parent, which is called the weaved goal. The weaving algorithm implemented in Q7 makes it possible to analyze the weaved goal model through a goal analysis tool, e.g., a goal reasoning algorithm [30].
As we can see from the examples presented above, Q7 provides a quality-based reuse mechanism for representing and modularizing crosscutting concerns in goal models. The Q7 language is not only capable of handling the characteristics of the quality knowledge, but also capable of relating those with functional descriptions. In addition, the textual form of Q7 greatly facilitates the tracing of stakeholder concerns throughout the software life cycle, as we shall demonstrate via a case study in Section 5.

4.2 Implementation in phpAspect

The early candidate aspects discovered in goal models are usually suited to be implemented as code aspects, but developers may choose other means to address these crosscutting concerns, as previously stated. Nevertheless, our approach explores the possibility to equip developers with a full-fledged aspect-oriented framework so that a clear separation of concerns is promoted throughout software development.

As one can see from Figure 3, functional and advising tasks from requirements are mapped into functional and aspectual modules in design and implementation, respectively. Since the subject in our case study – osCommerce – is implemented in PHP, we select a solution for AOP in this language, phpAspect, to facilitate the discussion in implementing early aspects.

PhpAspect is designed as an extension to the PHP language. It adds new constructs, such as: aspects, pointcuts, advices, and inter-types declarations, inspired by aspectJ for expressing aspects relating to objects and classes, while embracing specific features for Web-based applications. It provides pointcut expressions for constructions used in these applications, such as function call and execution, Web-based variable access, XML/HTML enclosing context identification, and the like. Moreover, phpAspect is able to weave aspect components in portions of code that are embedded into XML or HTML elements.

PhpAspect uses a static weaving process that performs source code transformation of a PHP program with aspect extensions into a standard PHP program. Both the source PHP program and the aspect modules are XMLized into abstract syntax trees, which can be weaved through customized XSLT stylesheets. The woven XML syntax trees, as a result, are transformed into the target PHP program via unparsing XSLT stylesheets.

The following code shows an example of the security aspect for a Web application. This aspect first introduces a credential checking around all Web pages that require access authentication (captured with the checkCredentials pointcut on goto method call). This checking prevents users from accessing a Web page if they are not logged in or do not have the right credentials. In these cases, users are redirected to a more appropriate page, either the login or index page. Secondly, the security aspect checks that all cart operations performed by the client are done in an HTTPS (SSL) mode and deny them otherwise.

```php
<?php
aspect Security {
    //Intercept all instantiations of a page
    pointcut checkCredentials:call(Page->goTo($arg2));
}
```

Developed by William Candillon during the Google Summer of Code, see http://code.google.com/soc/php/about.html.
//Intercept all method execution of the cart
pointcut checkSSL:exec(Cart-*(*));

//Advice: Around all page instantiations, check the credentials
around(User $user) checkCredentials {
    if($user->hasCredentials($_GET['page'],
        $_GET['action'])) {
        proceed();
    } elseif (!$user->isLoggedIn()) {
        $thisJoinPoint->getObject()->goTo('login.php');
    } else {
        $thisJoinPoint->getObject()->goTo('index.php');
    }
}

//Advice: Around all method execution of the Cart,
// We check whether the connection is SSL
around checkSSL {
    if(!$_SERVER['https']) {
        header("Location: https://{$_SERVER['HTTP_HOST']}
            {$_SERVER['REQUEST_URI']}");    
    } else {
        proceed();
    }
}

The above example not only demonstrates phpAspect’s competence in working out the implementation of requirement aspects in question, but also shows its capacity to map and trace early identified crosscutting concerns in the code base.

4.3 Aspect validation

It is crucial to validate the implementation against stakeholder requirements. We propose a goal-based testing approach to ensure that system functionalities are preserved and system qualities are enhanced by weaving aspects into base modules. This concept is highlighted by the validation flows in Figure [3].

In goal-oriented requirements engineering, when it is concrete enough to express the function of a task in terms of input and the expected output, a unit test case can be created to check whether the function is violated by comparing the output of the implemented function with the expected output of the required function. Therefore, the leaf-level functional task in the goal model corresponds to a set of unit test cases that tells whether the base program delivers the required functionality. Having enough unit test cases in terms of the coverage of the input domain, the functional task can be labeled “validated”.

Aspects discovered in goal models provide a baseline for code aspects validation. If an advising task cuts across multiple functional tasks, the unit test cases of the functional tasks at the join points can be reused to test the functionality of the weaved system. This is because requirement aspects must not change basic functionalities defined by hard goals and functional tasks. The implementation of aspects, therefore, has to preserve this property.

On the other hand, the degree of certain softgoal satisfaction must be enhanced by the weaved system. Measuring quality attributes typically presents an obstacle to tradi-
Table 2. Tracing the security (S) and usability (U) aspects in an osCommerce media shop

<table>
<thead>
<tr>
<th>Concept</th>
<th>Q7</th>
<th>phpAspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspect (S)</td>
<td>&lt;aspect&gt;::Security [system]</td>
<td>aspect Security</td>
</tr>
<tr>
<td>pointcut (S)</td>
<td>&lt;= + * [page]</td>
<td>call(Page-&gt;goTo($arg2))</td>
</tr>
<tr>
<td></td>
<td>&lt;= + * [cart]</td>
<td>exec(Cart-&gt;**(*))</td>
</tr>
<tr>
<td>advice (S)</td>
<td>{ &amp; Redirect [login] }</td>
<td>checkCredentials(...)</td>
</tr>
<tr>
<td></td>
<td>{ &amp; SSL [connection] }</td>
<td>checkSSL(...)</td>
</tr>
<tr>
<td>aspect (U)</td>
<td>&lt;aspect&gt;::Usability [language]</td>
<td>aspect Usability_Language</td>
</tr>
<tr>
<td>pointcut (U)</td>
<td>&lt;= + * [page]</td>
<td>call(Page-&gt;<em>printf(</em>))</td>
</tr>
<tr>
<td></td>
<td>&lt;= + * [date]</td>
<td>call(Data-&gt;strftime($arg2))</td>
</tr>
<tr>
<td></td>
<td>&lt;= + * [amount]</td>
<td>exec(Amount-&gt;display($arg2))</td>
</tr>
<tr>
<td>advice (U)</td>
<td>{ &amp; Translate [language, NLS] }</td>
<td>translatePage(...)</td>
</tr>
<tr>
<td></td>
<td>{ &amp; Display [format, date] }</td>
<td>dateTimeFormat(...)</td>
</tr>
<tr>
<td></td>
<td>{ &amp; Convert [currency, amount] }</td>
<td>convertCurrency(...)</td>
</tr>
</tbody>
</table>

...tional testing mechanisms, since NFRs are not always easy to metricize. Our effort of modeling aspects early in the requirements pays off here. The results from goal-oriented analysis, including the quality metrics, the advising task and pointcut of requirement aspects, can be reused and extended to test softgoal satisfaction.

For example, the media shop keeps users from accessing a Web page if they are not logged in or do not have the right credentials. We model this requirement as a security aspect, and map it to a code aspect in phpAspect, as explained in Section 4.2. We can define a set of unit test cases that act as unauthorized agents and try to break into the system. The expected output would be redirecting these malicious visits to the login or index page. Since these security-related test cases crosscut the ones devoted to testing system functionalities (e.g., shopping and searching), they can be regarded as unit testing aspects [25], thereby reusing the security aspect’s pointcut description to perform the test case weaving.

It is worth pointing out that validating requirement aspects can be carried out by other means than defining unit testing aspects. For example, typical Web layer components do not lend themselves to unit testing, unless proper frameworks such as HttpUnit or PHPUnit are employed. In order to ensure that shopping is done securely, testing scripts can be developed to automatically verify that all cart operations are performed in an HTTPS (SSL) mode.

5 An Extended Case study

We used an exploratory case study [31] as the basis for our empirical evaluation. The research question focused on how to leverage our approach in a real-world setting. Specifically, we derived the following hypotheses to guide the study design: (1) Modularizing and tracing broadly-scoped non-functional concerns throughout the software life cycle are enabled by our framework, and (2) Goal-based validation justifies why certain code aspects appear in the implementation.

The single case in our study is osCommerce, an open-source platform written in PHP, on which a Web-based media shop development can be fully based. In our...
Building on previous work [9], we used osCommerce to show how to discover aspects from media shop goal models. In particular, 7 requirement aspects were identified in [9], among which we choose security and usability aspects as 2 embedded units of analysis within the current case study. Such a selection is guided by the previous work in a familiar e-commerce domain, and represents a typical case and units of analysis since both security and usability are commonly discussed early aspects in the literature.

The data collection in our study consisted of three parts. First, the goal aspects of media shop were presented in [9] and further represented in Q7. Second, the implementation of osCommerce in PHP was accessible through open-source repositories. Our implementation of osCommerce’s code aspects in phpAspect was available at [32]. Third, the goal-based validation instrumentation was developed and gathered by the authors of this work (also available at [32]).

The analysis results of aspects tracing are presented in Table 2, from which the mappings between requirement aspects in Q7 and code aspects in phpAspect can be readily spotted. Specifically, a one-to-one correspondence exists between the name of a requirement aspect and that of a code aspect. Moreover, we map goal’s topics into parameterized pointcuts, and map softgoal’s operationalizations into advices.

We focus on the usability aspect in this section, as security is discussed in the previous section as an illustration of our approach. The requirement aspect “Usability [language]” is AND-decomposed into 3 parts. One translates natural language strings (NLS) appearing in a Web page to the local language. Another deals with displaying date and time in the desired conventional format. The third converts money amounts from a country’s currency into the local currency. The Q7 representations for each point-cut and advice of the usability aspect (U) are given in the second column of Table 2.

Correspondingly, Table 2’s third column highlights these concepts’ counterparts in the phpAspect implementation.

The implemented aspects were weaved into osCommerce’s base modules by the phpAspect weaver. We tested the weaved system in two respects: hard goal preservation and softgoal enhancement, as indicated by the validation flows in Figure 3.

Unit test cases existed for validating the functional requirements of the osCommerce system. Such test cases should not be affected by introducing the aspects that implemented the NFRs. Therefore, we reused the functional testing units without any change for checking the functionalities of the weaved system. For example, the shopping cart sum computation must be the same regardless of which natural language being used by the media shop customer. A unit test case using PHPUnit [6] was reused.

```php
require_once 'PHPUnit/Framework/TestCase.php';
require_once 'classes/cart.class.php';
class CheckoutTest extends PHPUnit_Framework_TestCase {  
  private function getOrder(){
    $cart = new Cart();
    $cart->addItem('Bread', 2);
    // 2.20 each in USD
    $cart->addItem('Butter', 1);
    // 3.20 each in USD
    return $cart->getAmount();
  }
  public function testCheckoutTotal(){
  }
}
```

We reused 22 functional unit test cases for the weaved system to make sure that introducing requirement aspects does not change the function of osCommerce. If one introduces an aspect that does change the functionality of the original system, we consider either the function is not intended originally, or new test case needs to be designed and weaved into the original set of test cases along with the code aspect. However, it is beyond our scope to discuss how an aspect should implement a functional requirement, and how such an aspect should be traced and validated.

Having checked that the weaved system preserved system functionalities, we wanted to test whether the aspects indeed addressed the quality concerns, and more importantly, whether they helped better achieve the original stakeholder softgoals. Such a validation was guided by the quality metrics derived from goal-oriented analysis. Take “Usability [language]” for example, osCommerce currently supported English, German, and Spanish users. Figure 4 shows a Web page in the default language – English. The usability aspect should render a Web page by using the language chosen by the user as natural as possible. This included showing textual strings, date, and currency in the desired language and format, as described earlier and indicated in Table 2. Figure 5 shows a screen shot of the weaved system after the language customization aspect is applied.

![Screen shot of an osCommerce media shop shown in default language (English)](image-url)
Fig. 5. Screen shot of the weaved system that enhances usability for Spanish users

We validated the usability aspect via two means. A Spanish tester confirmed that the language customization aspect worked very well, in that most Web page contents shown in the desired language, including date and currency, were semantically correct. To evaluate this result in a triangulating fashion [31], we also chose the pspell testing harness[7] to check the syntax of the resulting Web page texts automatically. The fact that all customized pages contained less than 5% syntactic errors increased our confidence that the aspects’ weaved system indeed helped better meet stakeholders’ usability requirement.

The analysis results of the 2 embedded units – security and usability – within the case study presented positive empirical evidence for accepting our two initial hypotheses. However, we did observe that existing goal models might not be complete to justify all code aspects. On the contrary, certain aspects in the implementation could help justify stakeholder goals and uncover the missing parts in the requirements.

When reengineering osCommerce using aspects, developers wanted to achieve a high degree of maintainability to facilitate modification and reuse. Aspects modularized tangled and scattered code, which led to a cleaner code base. For instance, in the original implementation, 603 natural language string variables were defined in each of the English, German, and Spanish language header files to be included in specific Web pages. This caused scattered code duplication. We defined a single usability aspect to

modularize these language customization concerns, and removed 3,990 lines of code, 7.6% from the whole code base. This helped address the maintainability softgoal from the developer’s perspective, which uncovered a missing part of media shop goal models presented in [9]. In this sense, one shall not apply our aspect tracing and validating framework in a strict forward-engineering way, but in an iterative fashion within an integrated software development process.

Several factors can affect the validity of our exploratory case study: construct validity, external validity, and reliability [31]. The key construct is the idea of a requirement aspect. Although softgoals have huge potential to become early aspects [33], others may argue that requirement aspects can be functional as well. We believe that requirement aspects are intended to enhance system qualities while preserving functionalities, and the early aspects community needs to make it more carefully. In regard to external validity, we chose Q7, phpAspect, and various testing mechanisms in tracing and validating requirement aspects for a common e-commerce application. Further empirical studies are needed to examine the applicability and generality of our framework in coping with other modeling notations, programming languages, and application domains. To reduce the threats to reliability, we selected an open-source project and made all our empirical data publicly accessible [32]. Thus, our reported study is replicable, and we believe we would obtain similar results if we repeated the study.

6 Related work

Grundy et al [34, 35] and Rashid et al [8] are among the earliest authors who recognized the advantage of aspects for reasoning about component-based software at the requirements level. Araujo et al [36] compose aspectual and non-aspectual early design models (e.g. Scenarios). They use sequence diagrams and executable state machines to weave aspectual scenarios represented by interaction pattern specifications. When this state machine-based approach scales up, it is possible to establish further traceability between the statecharts-based models with implementations [37, 38].

Haley et al [19] proposed an early aspect approach focusing on deriving security requirements from crosscutting threat descriptions. The security aspect in our case study can be considered as an operationalization of crosscutting anti-threat advices. In our validation framework, security is treated as an NFR that cannot be sacrificed.

In comparing the related work listed in a literature survey of early aspects [16], our key contribution brings early aspects traceability links and validation together. In [18], traceability links between early aspects and designs are set up by comparing naming conventions, term co-occurrences, etc. It is not clear how traceability through such queries can be verified, as the leap from design to implementation may distort the precision. For example, a design element with a similar name as the requirements may be implemented differently from what is specified. In our work, we explicitly rely on the decomposition and contribution links in the goal models to build traceability among goals and aspects. Therefore, our goal-based testing for aspects helps to show how well requirements are carried out by the advices.

The work of Cleland-Huang et al. [17] extracts NFRs based on information retrieval techniques. The strength of traceability is quantified as precision and recall of
the keyword-based search. When naming conventions mismatches the functionality specified in the program, our goal-based validation of traceability may improve their precision through the semantics of executed test cases.

Reflecting different viewpoints of the problem, early aspects can, in the extreme, crosscut each other symmetrically. In that sense, viewpoint merge/matching techniques in requirements engineering [15] can also be considered an early aspect weaving technique. A main technical difference is that model merging requires explicit representation of joinpoints between multiple viewpoints, while in early aspects such joinpoints are typically expressed implicitly by pointcuts expressions.

In [39], proof obligations were introduced to formalize the validation to the requirement aspects. Their approach can be applied to programs of well-defined axiomatic semantics. For the quality attributes that do not have a clear-cut answer to satisfaction, it is necessary to validate whether and how much the system can be improved after weaving the proposed aspects. For example, instead of proving that a word is Spanish, we show how well it is understandable by the Spanish-speaking users. Although we reuse unit testing for functional requirements, we believe a complementary approach based on generating proof obligations can better guide the validation of functional requirements.

In [40], Neil Maiden has called for quantifiable requirements. Frameworks, such as NFR [21], need to have a metricized way to verify that requirements are addressed properly using operationalizations. Most quantifications of NFR help selection of alternatives [41] when the satisfaction of the metricized NFR can be visualized [42, 43]. Our proposed goal-based testing framework can be regressively applied to join points of the softgoal operationalizations, thus more precisely measuring the satisfaction of the NFRs.

7 Conclusions and Further Work

We have proposed an aspect-oriented framework to investigate how aspects discovered in requirements can be validated and traced throughout the software life cycle. We evaluated the approach via an exploratory case study that reengineered a public domain e-commerce platform. The study presented our observations of and insights into the problems of tracing and validating requirement aspects. Even though the study collected positive evidence on the applicability and usefulness of our approach, more in-depth empirical studies are needed to lend strength to the preliminary findings reported here. In addition to confirming our hypotheses, we verified the initial AOP claim that it is natural to implement globally concerned NFRs as aspects that cut across subsystems [1]. From the results obtained, we believe that aspect orientation is a promising solution to tackling crosscutting concerns early on, and that our framework systematically handles traceability and validation issues for requirement aspects.

In future work, we will continue to explore concepts and mechanisms for tracing early aspects throughout the software lifecycle and validating them in a given design. The case study suggests that some candidate early aspects are easier to turn into coding aspects, and some softgoals are easier to measure through testing aspects. Automatic
tracing and validation of early aspects may require in-depth knowledge of the application domain and quality requirements.

We will also investigate how to trace other forms of early aspects into validateable design. The composition of problem and solution structures may share common validation concerns in problem frames [19], it is interesting to know whether problem frame concerns lead to coding aspects and whether the proof-obligations of trust assumptions can be validated using testing aspects.

Along another line of research, we have found aspects useful in instrumenting the monitors into the code such that one can dynamically diagnose problems in requirements satisfaction [44]. Such uses of aspects help to satisfy quality requirements of the software development process, rather than those for the software products. These can be generally regarded as software maintenance aspects, which may indirectly help improve the software product quality. Relating process-oriented early aspects to product-oriented ones may give rise to a better traceability on how software process helps produce high-quality software products.

Acknowledgements: We thank William Candillon and Gilles Vanwormhoudt for their help with phpAspect. We also acknowledge invaluable discussions and feedback from our colleagues: Robin Laney, Bashar Nuseibeh at the Open University, and Eric Yu, Rick Salay at the University of Toronto.

References