Electronic version of an article published as **Requirements Engineering Journal, Vol. 7, Issue 1, 2002, pp. 3-19**
[doi: 10.1007/s007660200001]

Rationale-based Use Case Specification

Allen H. Dutoit* and Barbara Paech°

*Technische Universität München, Institut für Informatik, Munich, Germany
dutoit@in.tum.de
°Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
paech@iese.fhg.de

Abstract. The requirements specification – as outcome of the requirements engineering process – falls short of capturing other useful information generated during this process, such as the justification for selected requirements, trade-offs negotiated by stakeholders, and alternative requirements that were discarded. In the context of evolving systems and distributed development, this information is essential. Rationale methods focus on capturing and structuring this missing information. In this paper, we propose an integrated process with dedicated guidance for capturing requirements and their rationale, discuss its tool support, and describe the experiences we made during several case studies with students. Although the idea of integrating rationale methods with requirements engineering is not new, few research projects so far have focused on smooth integration, dedicated tool support, and detailed guidance for such methods.

Introduction

There is a wide variety of techniques for the elicitation, specification, validation, and management of requirements, but only few of them are used in industry. For example, at a recent seminar given to around 100 developers in the car industries (suppliers and procurers), 90% of the participants used natural language text edited in MS Word for the requirements specification [1]. Also, the experience from several industry projects, in which the authors were involved, shows that even the quality of requirements documents that adhere to some standard is often fundamentally flawed, because:

- they do not contain the information needed by the people who have to rely on them,
- this information is often inconsistent, ill-structured, and imprecise,
- the authors of the specification did not find an adequate level of abstraction that enables them to avoid design decisions while capturing all relevant requirements details.

The reasons for these flaws are manifold and typically depend on the context. However, in general, three issues seem to be essential for a successful requirements engineering process:

- *Smooth integration among the techniques applied*. The lack of integration among techniques is the most critical of these three issues. For example, there is no integrated method established for the simultaneous usage of use cases and class models.

- *Dedicated tool support*. Although there exist modeling and requirements management tools, these tools are general purpose and do not support specific tasks. Again, this holds true, for example, for use cases, where there is no established tool support for the capture and management of use cases.
- *Detailed guidance for participants*. Most techniques suggested from academia are not sufficiently well explained to be usable by persons other than their inventors. Similarly, this holds true for established techniques like use cases, where, for example, there is almost no guidance regarding the right level of abstraction adequate for certain project contexts.

In this paper, we describe the integration into a single process of two techniques, use case specification and rationale capture, along with their associated tool support and guidance. Use case specification enables developers to specify a system in terms of sequences of interactions between users and the system. Rationale methods enable developers to capture the justification of their decisions and the related decision making elements. Hence, integrating both techniques should yield a method that captures all information appropriate for all stakeholders, that supports stakeholders for negotiating and refining the level of detail of this information, and that enables stakeholders to evolve this information as a response to change. We have incrementally developed this process and its associated tool support and guidance, by continuously evaluating and improving them in the context of case studies with students. Student case studies are clearly insufficient for demonstrating the usefulness of this process for industry. However, the use of novice subjects has enabled us to develop guidance and supporting material that, in our view, will make this process more easily transferable to clients and practitioners during field trials and, later, wide-spread use.

The rest of the paper is structured as follows: First, we describe our process for use case specification and rationale capture. Then we provide an overview of the tool support. The fourth section summarizes the lessons learned so far. In the fifth section we discuss related work. We conclude in the sixth section. Throughout the paper we use the well-known meeting scheduler example [2].

Process Overview

Use cases are a popular addition to object-oriented software development. They have first been proposed by Jacobson [3] and are now part of the (Rational) *Unified Software Development Process* [4]. One of the main difficulties with writing use cases is their granularity [5], that is, the partitioning of system functionality into individual use cases and the level of detail for writing each use case. Ideally, the partitioning of the specification into a set of use cases and the level of detail should be such that the resulting specification accurately reflects the customer's and users' goals. This can usually only be attained through an iterative process of negotiation and refinement with the customer.

Rationale methods aim at capturing, representing, and maintaining records about why developers have made the decisions they have [6]. Rationale includes the problems developers encountered, the options they investigated, the criteria they

selected to evaluate options, and, most important, the debate that lead to making decisions. Rationale can be used to support negotiation (increasing the quality of the decisions made) and to capture contextual information (facilitating future changes to system requirements) [7]. Rationale methods are currently not widespread because of their low acceptance among developers and their cost: Under time pressure, it is difficult to justify the capture and documentation of additional information that will only be useful downstream to other, unknown project participants.

Our ultimate research goal is to support the evolution of software by providing an integrated process for use case specification and rationale capture [8]. By providing templates and guidance for use case writing, we hope to address common issues about granularity and facilitate the communication between customers and developers. By providing an explicit rationale process supporting the negotiation among customers, users, and developers, we aim to facilitate decisions about system requirements and use case granularity. By creating a short-term incentive for this rationale process, we also aim to opportunistically capture rationale information that is useful for the longer term (i.e., evolution). Finally, to further decrease the overhead of capturing rationale for the developer, we introduce a new role, the rationale maintainer, whose task is to augment, filter, and structure the rationale for longer-term use.

However, before we can focus on the support for evolution, we first need to understand the details of applying use case specification and rationale capture to a realistic problem. We have done this by incrementally refining and evaluating our process, together with its guidance and tool support, in the context of case studies. In the remainder of this section, we describe in more detail the products and activities of our process aimed at writing use cases and capturing rationale.

Table 1 An example of user task.

User Task Name	Manage Interaction Among Participants
Initiating Actor	Meeting Facilitator
Participating Actors	Meeting Participant
Task Description	The Meeting Facilitator is responsible for getting replies from participants who have not reacted promptly, for notifying participants of changes of date or location, and for keeping participants aware of current unresolved conflicts or delays in the scheduling process.
Realized in Use Cases	Handle Replies, Remind Participant, React to Replan Request

Table 2 An example of a use case.

Name	Handle Replies	
Realized User Task	Manage Interaction Among Participants	
Initiating Actor	Meeting Facilitator	
Participating Actors	Meeting Participant	
Flow of events	**Actors**	**System**
	1. The Meeting Facilitator selects "Handle Replies" for a meeting and a question.	
		2. The system checks if all participants replied [Exception: Slow participant].
		3. The system starts the "Close Question Service" and notifies the Meeting Initiator accordingly.
Exceptions	[Slow participant] The meeting facilitator decides whether to remind the participants or to close the question. In the first case s/he selects the "Remind Participant Service". In the second case s/he selects the "Close Question Service".	
Precondition	The meeting Initiator has initiated the meeting and asked some question.	
Postcondition	The participants have been reminded or the question is closed.	
Includes Use Cases	-	
Used Services	Check Participant Replies, Remind Participant, Close Question	
Non-functional Requirements	Response Time, Minimize Amount of Messages, Flexibility	

Products

We describe the functional aspects of a requirements specification using five types of elements: actors, user tasks, use cases, system services, and glossary entries:

Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a bank customer) or another system (e.g., a central database).

A *User Task* is a unit of work that is meaningful to the user. It includes the environment in which the system operates and is often a step in an encompassing business process. Thus, user tasks are similar to Cockburn's *Summary Goal Use Cases* [9]. We use the term user task because we rely on techniques from task analysis for their identification [10]. Only by knowing the user tasks in detail a

system with maximal support to the user can be designed [11]. Table 1 depicts as an example the user task "Manage Interaction Among Participants".

A *Use Case* describes how a user task can be achieved with a sequence of interactions with the system. This corresponds to Cockburn's *User Goal Use Case* [9]. We use the essential use cases of [5], where each use case step has a number, and actor and system steps alternate. Table 2 shows as an example the "Handle Replies" use case.

A *System Service* describes the input and output of individual system functions. While use cases put system functions into context, system services describe system functions independently of the user task. This corresponds to Cockburn's *Subfunction Goal Use Cases* [9]. While user tasks and use cases are important to communicate with the customer, the service description is the important input for the system designers. Table 33 shows as an example the "Remind Participant" service. The service template looks similar to the use case template. The main difference is that no context information (e.g., actors) is provided. Instead input and output are described explicitly and the flow of event may include user interface details. Non-functional requirements are inherited from the use case and additional non-functional requirements are added which apply only to this service.

A *Glossary Entry* defines an important concept relevant to the user tasks or the system services. There are two reasons for maintaining a glossary in the specification. First, it allows requirements engineers to document accurately the terms of art used by the client. Second, it enables requirements engineers to reduce redundancies and inconsistencies in terms used to describe the system.

Figure 1Figure 11 depicts the relationship between user task, use case, and system service.

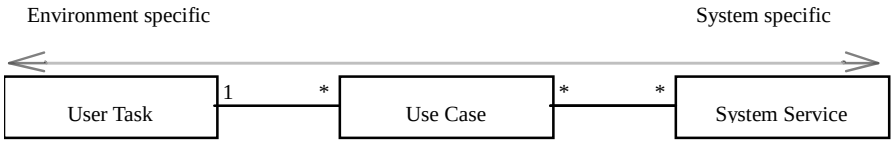

Figure 1: Relationship between User Task, Use Case, and System Service (UML class diagram).

A major feature of our process is that it not only covers functional requirements, but also non-functional requirements (NFR)[1]. NFRs are essential for rationale capture, since they provide criteria for assessing different options for use cases and services. This is similar to the NFR-framework [12], where *softgoals* are refined into several different operationalizations. As discussed in [12], there are different taxonomies of NFR. We distinguish between three types of NFR as explained in Table 44. *Domain properties* describe facts of the domain and therefore have to be satisfied through user tasks and use cases. *Global functional requirements* are high-level functional requirements and therefore have to be satisfied through use cases and system services. *Quality requirements* are additional constraints on the characteristics of the

[1] Often, NFR encompass product and process or project requirements. Here we concentrate on product requirements.

requirements elements. Domain properties play a special role in that they describe facts that are not changeable during the requirements engineering process. Jackson calls these NFRs *indicative* properties [13]. Global functional requirements and quality requirements are subjects of the requirements engineering process. Jackson calls these *optative* properties.

Table 3 An example of a service

Name	Remind Participant	
Used by Use Cases	Handle Replies	
Flow of events	**Actors**	**System**
	1. The Meeting Facilitator selects the "Remind Participant Service" for a meeting and a question and a Meeting Participant.	
		2. The system shows a default text for a message to remind the participant.
	3. The Meeting Facilitator edits the text and triggers the sending.	
		4. The systems sends the message to the Meeting Participant [Exception: Problem with Email address] [Exception: Problem with message system]
Exceptions	[Problem with Email address]: The system displays an error message asking for another address. <continue with 3.> or The Meeting Facilitator aborts the service call. [Problem with message system]: The system displays an error message and stops service execution.	
Precondition	-	
Postcondition	The participant has been reminded to answer the question for the meeting.	
Input	Meeting Identifier, Question Identifier, Participant Identifier	
Output	Email addressed to the Participant	
Non-functional Requirements	Minimize length of message	

Our types are only used as a rough guidance to check for three basic types of NFR. They are much simpler than goal types in goal-oriented approaches to requirements engineering (e.g. GBRAM[35] or KAOS[2]) which drive the requirements elicitation. In contrast to these approaches we use user tasks instead of goals to drive the requirements elicitation and specification process. We only use the NFR as criteria for the evaluation of the adequacy of use case or service design with respect to user tasks and use cases, respectively.

Table 4 Types of NFRs.

Property type	Explanation
Domain property	Facts of the domain to be adhered to by the software system (e.g. “a person may not be at two different places”)
Global functional requirements	High-level functional requirements that cannot be attributed to single use cases, but affect several use cases (e.g. “the meeting scheduler must in general handle several meetings in parallel”).
Quality requirements	Requirements on characteristics of user tasks, use cases or system services, e.g. “the elapsed time between the determination of a meeting date and location and the communication of this information to all participants concerned should be smaller than 5 sec.”.

To represent rationale we use an *issue model* as proposed by argumentation-based rationale approaches [6]. Issue models represent the individual decision making elements that lead to a decision as individual nodes and their relationships with edges. Many different models have been proposed, including IBIS (Issue Based Information System, [14]) and QOC (Questions, Options, Criteria, [15]), to name the principal ones. We use a refinement of QOC that includes the following elements (see the concept model in Figure 233):

Questions represent needs to be solved for the requirements process to proceed. Questions can indicate a design issue, a request for clarification, or a possible defect.
Options are possible solutions that could address the question under consideration. These include options that were explored but discarded because they did not satisfy one or more criteria.
Criteria are desirable qualities that the selected option should satisfy. In our model, criteria are NFRs.
Assessments represent the evaluation of a single option against a criterion. An assessment indicates whether an option satisfies, helps, hurts, or violates a criterion. Assessments are used to establish the fitness of options within a question.
Arguments represent the opinions of individual stakeholders, in particular, about the relevance of a question or the accuracy of an assessment. By arguing about relative merits of options, stakeholders can build consensus and converge towards a solution.
A *Decision* is the resolution of a question representing the selected option. Decisions are already implicitly captured in the use cases during requirements engineering. We only need to capture the relationship between decisions and their corresponding rationale.

Table 5: An example of rationale

Justification	What is the best Option for the system boundary within in the "Handle Replies Use Case" satisfying the non-functional requirements?		
Criteria:	Response Time	Minimize Amount of Messages	Flexibility
Option 1: The system collects replies and reminds slow participants automatically during a given time within a given interval. The system then closes the question and informs the Meeting Facilitator.	+	-	-
Option 2: The system collects replies and informs the Meeting Facilitator about the status automatically after a given interval. The Meeting Facilitator decides whether to close the question or to remind participants.	O	O	+
Decision: The system collects replies. The Meeting Facilitator chooses when to handle replies and accordingly checks the status and decides whether to close the question or to remind participants .	-	+	+

As an example for a rationale consider the justification of the "Handle Replies" use case given in Table 556. The question is the optimal system boundary. Three options are sketched and evaluated against the criteria. The assessments +, O, - indicate good, sufficient, and insufficient satisfaction. An argument for the good satisfaction of the "Response Time" criterion of the first option is that in any case the question is closed within the given time. However, this system behavior impacts negatively on the flexibility of the Meeting Facilitator, because there is no way s/he can extend the time for participants to reply before closing the question. The chosen option is marked by boldface-letters. If the criteria are of different priority, the option with the highest score of "+" need not be the optimal one.

During review, use cases and services are challenged. This way new issues are created, for example: *Can the remind message in the "Remind Participant Service" be created and send without editing through the Meeting Facilitator in order to reduce the "Response Time criterion"?* During the discussion options, assessments and possibly new criteria will be generated and the decision for this question together with its rationale will be consolidated in a table similar to Table 556.

The concept model in Figure 233 shows the relationships among requirements elements and rationale elements that are created and maintained in our process and tool.

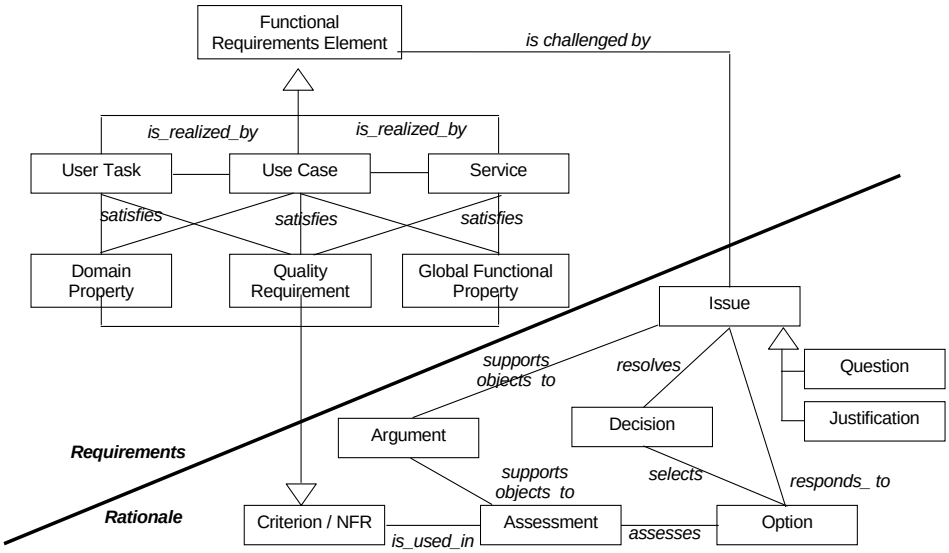

Figure 2: Concept model

As shown in Figure 345, the input of our requirements engineering process is a *Problem Statement*, written by the client and the requirements engineers, describing the user tasks that the system should support. The problem statement serves two purposes: First, it provides an initial description of the environment of the system (e.g., a set of actors and user tasks). Second, it establishes the scope of the work supported by the system (i.e., which user tasks should be supported and which should not). We are well aware that producing an adequate problem statement requires an elicitation process in itself. However, here, we concentrate on the specification of requirements to be used as input to software development.

Based on the problem statement, the requirements engineers write the specification in terms of use cases, services, glossary entries, and NFRs. The specification process is iterative and incremental. The requirements engineers may decide to write and refine only a limited set of use cases, services or NFRs at the time (i.e., a depth first approach), or, conversely, work concurrently on all use cases, services, and NFRs (i.e., a breadth first approach). In parallel, parts of the specification are reviewed which triggers further refinements of the specification.

Process activities

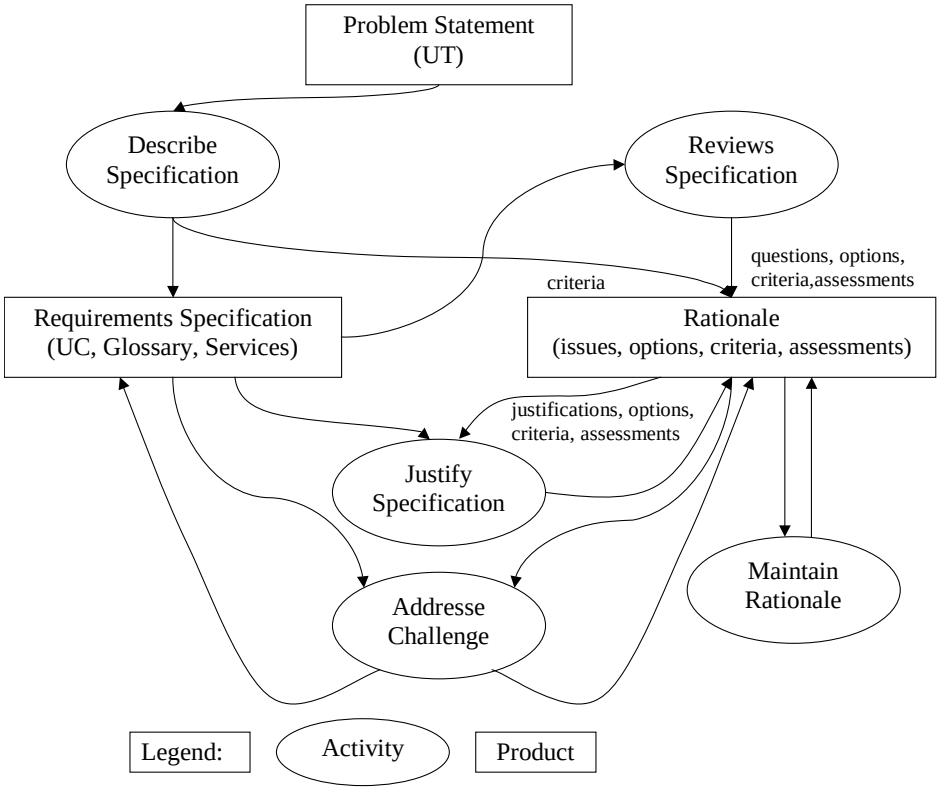

Figure 3: Process Model

As shown in Figure 345, the rationale of requirements is captured during four activities. The *Justify Specification* activity, executed by a requirements engineer or a reviewer, focuses on capturing rationale through explicit justification. The *Review Specification* activity focuses on capturing rationale through requests for clarification and challenges on requirements. The *Review Specification* activity is followed by the *Address Challenge* activity, during which developers and reviewers discuss solutions to address challenges. Finally, the *Maintain Rationale* activity focuses on consolidating and restructuring the rationale for long-term use. Similar to the specification activity, rationale capture is iterative and incremental. Each activity is intertwined with specification.

Next, we describe each activity in more detail.

Describe Specification

The *Describe Specification* activity is executed by the requirements engineer and is composed of the following steps:

Describe Use Cases & Services. This step develops an initial draft of one use case for each user task which determines which parts of the user tasks are realized by the system and which are realized by the user. Each use case is then refined into a number of further use cases and system services. The result of this step is a description of the interactions between the users and the system (in terms of use cases) and a description of the features offered by the system (in terms of system services).

Define NFRs. This step identifies and describes NFRs for each use case and service. These NFRs describe properties that the system must have in order to be useful to the user. This step may also result in NFRs that are applicable to the complete system.

Describe Exceptional Cases. This step describes the response of the system under error conditions, such as wrong user input or component failure. Exceptional cases are also described as flow of events but are separated from common cases for clarity.

Create Glossary. All terminology specific to the use case is captured in a glossary. This includes terminology specific to the user tasks as well as terminology specific to the system services described in the use cases

Justify Specification

The *Justify Specification* activity is executed by the requirements engineer. Requirements engineers explicitly capture rationale by justifying each use case and system services by documenting alternatives that were discarded as options and assessing them against the NFRs to show how the current option is the best (wrt. the NFRs). A justification takes the same form as any other question in the system, except that it is usually created by a single author and that it is closed.

Review Specification

The *Review Specification* activity is executed by a reviewer and is composed of the following steps:

Request Clarification. A reviewer reads some part of the specification and finds it unclear, and requests a clarification regarding a term or the phrasing of a paragraph.

Challenge Specification. A reviewer reads some part of the requirements specification and challenges problem areas with questions. The difference between a challenge and a clarification is that the former points out a definite problem in the specification whereas the latter often results from a misunderstanding from the reviewer. Note that the reviewer can also challenge the specification by reading and reopening the justification associated with a use case or a system.

Address Challenge

The *Address Challenge* activity is composed of the following steps:

Propose and Assess Options. Questions can result in the discussion of possible changes in the requirements specification. A possible option that is always available is the status quo, that is, not to change the requirements. Clarification questions are addressed with options to improve the requirements specification

without necessarily resulting in changes to the system. Once a sufficient number of options have been proposed, requirements engineers need to evaluate them and refine them to satisfy the NFRs. The resulting QOC models are similar to those resulting from justification. The difference is that a justification is systematically written by a single author (the requirements engineer responsible for the use case/service) whereas a challenge and resulting discussion is incrementally written and refined by a number of authors (the reviewer and the stakeholders interested in the use case/service).

Discuss Options. During this step, requirements engineers create arguments supporting and opposing options. While the previous steps focus on the objective evaluation of options against well-defined criteria, this step focuses on the arguments and negotiation among requirements engineers to validate these assessments and to prioritize criteria.

Decide. Once requirements engineers have evaluated and refined (most or) all options, requirements engineers create a decision by selecting an option which can result in minor or substantial change in the requirements specification. Note that a clarification question can be resolved without any changes. Note also that addressing a question may invalidate previous decisions.

During rationale capture, requirements engineers may skip any of the above steps. Options can be generated and evaluated without an explicit question. Decisions can be taken and changes implemented without explicit discussion. It is desirable, however, that at least some of the components of the decision are recorded so that the rationale maintenance process can recover the missing parts.

Maintain Rationale

The *Maintain Rationale* activity is executed by the rationale maintainer whose responsibility is to keep the content and structure of the rationale up to date. The *Maintain Rationale* activity is composed of the following steps:

Identify Missing Questions. Given that requirements engineers and the reviewers may skip steps in capturing rationale, there can be options that were captured without their corresponding question. In most cases, the implicit question can be made explicit using the options.

Identify Missing Decisions. Most decisions occur during meetings or face-to-face conversations. Consequently, they may be implemented in the requirements specification but not captured in the issue model. The rationale maintainer can identify these decisions by ensuring each change is associated with a decision.

Consolidate Options. When discussing a question, the requirements engineers may propose similar options. The rationale maintainer consolidates identical options into single nodes and restructures similar options.

Consolidate Questions. When reviewing requirements elements, reviewers may raise similar questions. The rationale maintainer consolidates identical questions into single nodes and restructures similar options.

The task of the rationale maintainer can be quite cumbersome if requirements engineers and reviewers capture too much rationale that does not have much value for long-term rationale. In particular, questions requesting clarification or challenging the

form of the specification are resolved quickly and are not worth remembering. However, during rationale maintenance, if the rationale maintainer were to read all these questions and filter them out manually, the rationale maintenance activity would be excessively time consuming and error prone. To address this issue, we use a type attribute for the question node, as shown in Table 667. The authors of questions indicate the type of question they are raising, which makes the post-processing task of the rationale maintainer much easier when filtering out questions without long-term value.

Table 6 Types of questions.

Question type	**Relationship to requirements**	**Available actions**	**Value for rationale**
Challenge on form	Linked to one or more elements that do not comply with the structure supported by the tool (e.g., confusion between user tasks and use cases).	Close question by revising related elements	None
Challenge on content	Linked to one or more elements the author of the question disagrees with.	Propose options Select criteria Revise assessments Close question once consensus is reached	High
Clarification	Linked to statement in a requirements element that is not clear.	Close question by clarifying unclear requirement. (No criteria or options are associated with this question)	None
Inconsistency	Linked to two or more elements that are inconsistent.	Propose option Close question by revising related elements.	Low
Justification	Linked to requirements element that is being justified.	Reopen question (in which case this question behaves the same way as a challenge on the content)	High
Omission	Linked to one or more elements and describes statements that have not been written down.	Propose option. Close question by filling gaps.	Low

A side effect of typing questions is that the issue model becomes much more specific. The types in Table 667 effectively correspond to a taxonomy of defects. Consequently, these question types makes it easier to develop tool and process guidance, by providing, for example, different actions and views for each question depending on its type. The second column of Table 667 lists the relationships between the questions and their related requirements elements and the third column of Table 667 lists the restricted set of actions available for each type of question.

Integrating Specification and Rationale

Capturing and maintaining rationale will yield benefits only if both requirements and rationale capture and their corresponding tool support are integrated. Indeed, the integration of rationale methods and tools with various aspects of development is a fundamental issue that has received little attention in rationale research [16].

A novelty of our approach is that NFRs are used as the integrating concept between the specification and its rationale (see the concept model in Figure 233). On the one hand, NFRs represent domain properties, global functional requirements, and quality requirements that must be satisfied. On the other hand, NFRs represent criteria that can be used when assessing options in justifications or in responses to challenges.
The two following examples illustrate the integration and interaction between requirements elements and rationale elements:

Example 1. A reviewer identifies a defect in the "Remind Participant" service because the service does not seem to satisfy the "Response Time" NFR. He indicates this by:

Creating a challenge on content

Describing the current option, including a negative assessment linked to the given criterion explaining the source of the challenge

Describing an improved option, including a positive assessment wrt. the given criterion and to other relevant NFRs.

The original author of the faulty use case can then either select the proposed option or propose a different option.

Example 2. A requirements engineer describes the reasoning behind the "Handle Replies" use case (see Table 556) by:

Creating the justification question,

Describing the current option and the alternatives that were discarded,

Entering the assessments between each of these options and the relevant NFRs, hence, explaining how the current option satisfies these requirements better than the alternatives,

Creating new NFRs and corresponding assessments, as needed, to better justify the current option, and

Closing the question with the current option.

In the first example, we observe how a reviewer can point out inconsistencies between requirements elements and NFRs with negative assessments. In example 2, we observe how a developer can justify the current solution (thus clarifying the specification) and discover NFRs that were left implicit until then (thus improving the completeness of the specification). Such interactions between functional requirements elements, NFRs, challenges, and justifications results from the tight integration between requirements and rationale and enables developers and reviewers to improve the specification.

Tool Support

In the previous section, we described products and activities for developing a use case specification along with its associated rationale, through collaboration, justification, and review. In the following, we give an overview of REQuest, our tool for supporting these processes.

The design goals of the tool were to provide a simple and integrated solution to manipulate use case and rationale models, embedding only minimal process specific knowledge. The tool is a Web application that can be accessed via standard Web browsers. This enables users to access the tool remotely from a variety of environments (e.g., lab, home, office) without the installation of additional software. The main view of the tool presents the user with three frames: a title, a requirements specification view and a rationale view (see Figure 456).

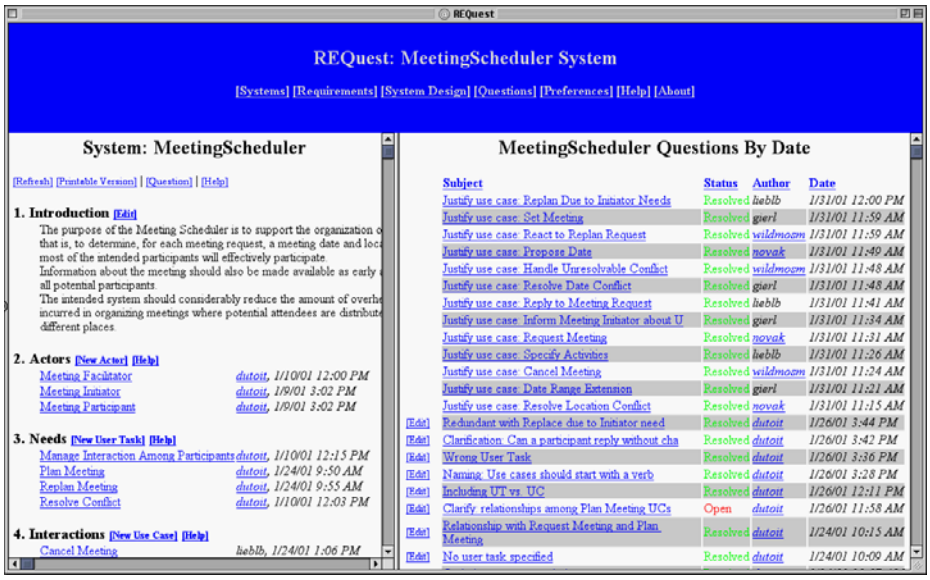

Figure 4 Tool overview: requirements specification (left column) and rationale (right column) are allocated the same amount of screen real estate.

The requirements view displays the requirements specification as a hypertext document, structured into actors, user tasks, use cases, services, glossary entries, and NFRs. The tool provides templates, text boxes, and selection menus for each requirements element. The tool recognizes known terms (e.g., glossary entries, the name of user tasks, use cases, and system services) and highlights them automatically in text fields where the terms appear. For example, if the name of an actor appears in the flow of events of a use case, the name of the actor is highlighted. The user can then click on the highlighted name to examine the attributes of the actor.

In the rationale view, information is structured according to the QOC model presented in the previous section and displayed as tables and hyperlinks, thus maximizing the density of information that the user can read in a single screen. Displaying rationale as text is a different approach than other well-known rationale-based tools (e.g., gIBIS [17], SYBIL [18], QuestMap [19]), which display rationale as a graph. In addition to the QOC structured information, users can annotate questions with informal comments or arguments to provide reference information or negotiate various aspects of the question.

In the following subsections, we focus in more detail on three aspects of the tool that are specific to our process: linking requirements and rationale elements, supporting justification, and supporting rationale maintenance.

Linking requirements and rationale elements

When viewing any requirements element, the user has the opportunity to create questions associated with the viewed element. By clicking on a question button, the user creates a question of a specified type and content (Figure 56). The user may choose to continue the question process and associate more rationale elements with the question, such as options, relevant criteria, and assessments. As the question is created incrementally, the user can choose to enter as little or as much information as necessary. For inconsistency questions, the user is prompted for references to other parts of the specification that are involved in the inconsistency questions.

Since the user must first view a requirements element before asking a question, all questions are automatically associated with at least one requirements element. The relationships between requirements elements and questions is a many-to-many and bidirectional relationship. When viewing an element in the requirements view, the titles of the questions associated with the element appear as a list of hyperlinks. When clicking on the title of a question, the user can examine the content of the question (operations, criteria, assessments, decision) in the rationale view. Similarly, when viewing a question in the rationale view, the list of elements related to the question appear as hyperlinks that the user can use to display a related requirements element in the requirements view. Hence, the user can quickly examine the relationship between two or more seemingly independent requirements elements that participate in related questions.

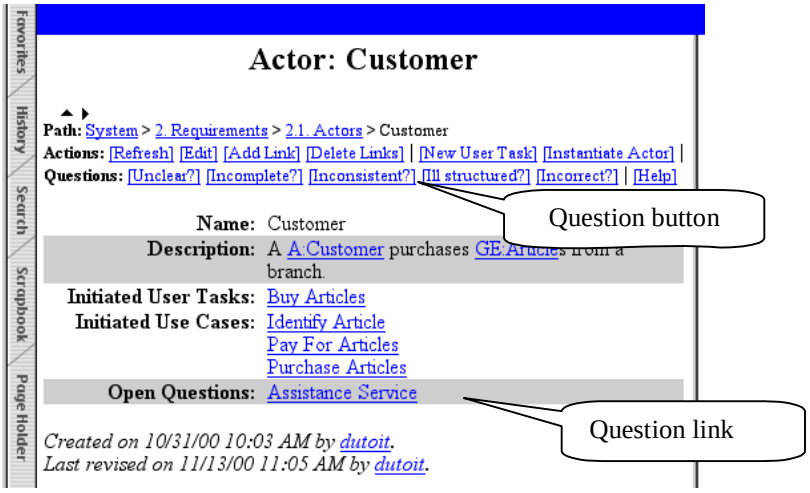

Figure 5: Creating questions and following question links

Supporting justification

The REQuest tool supports the justification of use cases and services. When viewing a use case in the requirements view (i.e., in the left column), the developer uses the [Justify] feature to initiate the justification process, which includes completing several forms in the rationale view (i.e., in the right column). Keeping the justified element and its justification in separate columns enables the developer to examine any part of the requirements specification without disturbing the forms associated with justification. The justification process consists of the following steps:

The tool presents the developer with a summary of the justification process, explaining what forms will appear.

The tool checks if the use case or the service is well formed. A well-formed use case has an initiating actor and is associated with the user task that it realizes. A well-formed service is attached to at least one system step in a use case. The tool also issues warnings if no quality requirements are associated with the use case or service.

The tool computes the set of NFRs that are applicable to the use case or service. This includes the quality requirements attached to the element and any NFR inherited through associations (e.g., domain properties attached to an associated user task). The set of applicable NFRs are used as criteria in the justification question. The developer can extend or reduce the set of criteria if necessary.

The developer summarizes the alternatives that could have been considered.

The developer describes how the selected solution differs from the alternatives.

The developer assesses the alternatives and the current solutions against the selected set of criteria.

In the final step, the tool displays the QOC matrix representing the justification and marks the element as justified.

A reviewer or a developer may reopen the justification at any point to revise it or to challenge it. Once a justification question is reopened, it can be manipulated in the same way as a challenge on content.

Supporting rationale maintenance

The REQuest tool supports the maintenance of rationale by providing several features for viewing rationale elements and their relationships with the requirements elements:

View questions by status enables the rationale maintainer to identify questions that have not yet been resolved. In most cases, such questions indicate issues that have been resolved in the requirements specification, but whose resolution has not been documented. In the case of challenge questions, the rationale maintainer elicits more information from the developers and enters the decisions that have already been taken.

View questions by type enables the rationale maintainer to access questions that are interesting for long term rationale (e.g., justifications and challenges on content) and to review them. If a documented decision is not consistent with the assessments, the rationale maintainer can either attach comments to the question to clarify the decision, add missing criteria in the assessment matrix, or reopen the question and require a developer to enter the missing information.

View unjustified elements enables the rationale maintainer to identify specification elements without justifications or without rationale. For elements with questions but without justification, the rationale maintainer creates a justification and consolidates the information from the other questions into the justification. For elements without questions, the rationale maintainer can request the author of the element to complete the justification process (see Figure 67).

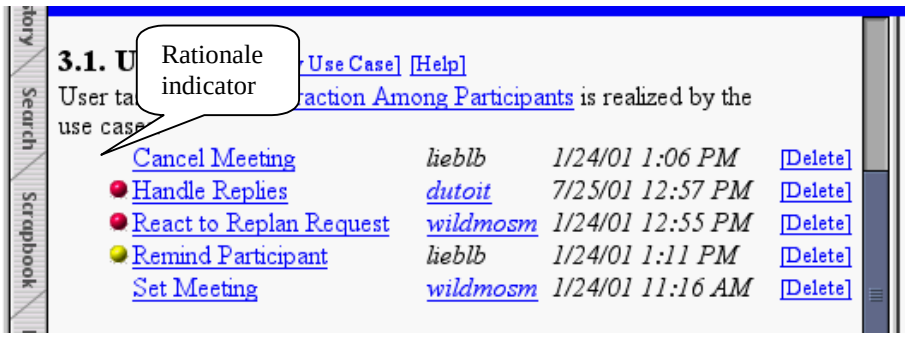

Figure 6:View unjustified elements. Rationale indicators next to elements in the overview indicate the status of each element wrt. captured rationale

While these features are designed to support the rationale maintainer, reviewers and developers may also use these features to access the rationale when accomplishing their own tasks.

Tool architecture

The current REQuest prototype tool is implemented as Java servlets [20] that store their persistent objects (e.g., requirements and rationale products) into an SQL database. Users access the tool with a standard Web browser that supports Javascript and tables. The requirements specification can be exported as an HTML document, which can then be imported into a word processor for final formatting. The tool has scaled up to the situations we face in the project course and the seminars (e.g., 15 concurrent users, specifications of ~30 use cases, rationale of ~60 questions) and could scale up to much larger situations. The current version of the tool, however, is missing several critical features for use in an industrial environment, including supporting interchange formats with other CASE tools (e.g., XMI [21]) and version control.

We also built a prototype of our concept model in the requirements management tool DOORS [22] to evaluate how our process could be supported by a tool developed and applied in industry. We found no major conceptual problem in using DOORS to store our requirements and rationale elements. However, the effort to develop a sufficiently usable adaptation is high. Moreover, we found that the learning curve faced by students when learning to use DOORS is steep as DOORS provides many features that are not always relevant to our process. Thus, we decided not to burden our students with this prototype.

Lessons Learned: Experiences with Process and Tool

We evaluated and incrementally refined the process described in the previous sections in case studies with students. So far, we conducted four case studies with three versions of the process, tool, and guidance. The goal of these case studies was to evaluate qualitatively if the guidance associated with the process was sufficient for novice participants. In particular, we were interested in the following points:

- the distinction (granularity, context, purpose) between user tasks, use cases, and services,
- the representation of rationale as a QOC model displayed as a textual matrix,
- the relationship between NFRs and criteria, and
- the process for asking and resolving questions.

While these case studies were not designed to compare our process against others, we were able to gain qualitative insights into the strengths and weaknesses of our process. We plan future work that will include an in depth evaluation of the process with professional subjects.

In this section, we first describe the experimental context of the case studies. We then summarize the lessons we learned with the first four main activities of our process, namely, specification, justification, review, and addressing challenges. We

have not yet evaluated the maintenance activity with students, as our focus has been initially on the activities capturing rationale.

Experimental context

In each case study, we provided a 45 minute tutorial to the process and the tool, an online help document, and written guidance. We surveyed the students during and after the case study with a structured questionnaire, examined the delivered specification and the issue model, took notes of our observations and of informal discussions with the students. The exploratory nature of the case studies, the number of subjects (4–22 per case study), and variables (background of participants, system under specification, process and tool variations) did not allow for a rigorous quantitative study.

Our primary evaluation context is the software engineering project course offered at Technische Universität München (TUM) [23]. This project provides students with a realistic software engineering experience during which students build and demonstrate a system for a real client. During our first case study in winter 2000/01, 22 students divided into four teams developed a prototype augmented reality application for nuclear powerplant technicians. 15 students were involved in the requirements engineering of the system, which lasted 5 weeks.

Following the project course, we evaluated an improved version of the process and tool in a requirements engineering seminar at TUM. Four students spent four weeks developing a requirements specification for the meeting scheduler problem [2]. This smaller and more focused setting enabled us to investigate in more detail the explicit capture of rationale. One week in this exercise was dedicated only to consolidating existing use cases and entering justifications. Moreover, since the students of the seminar had more experience and were more motivated than the students in the project course, we were able to better distinguish problems with the guidance from problems with the process itself.

During the summer semester of 2001, we evaluated a third version of the process and tool during the summer in a design rationale seminar at TUM (4 students, 6 weeks) and a requirements engineering lecture at the University of Kaiserslautern (8 students, 10 weeks). Both the seminar and the lecture used the same meeting scheduler problem statement as in the winter seminar.

Table 7. Number of requirements and rationale elements by case study.

	Participants	**Use Cases**	**Services**	**Questions (Justifications)**
Project course	15	29	0	62(0)
Seminar 1	4	17	13	40(13)
Seminar 2	4	13	6	43(9)
Lecture	8	7	12	37(12)

Specifying functional requirements

We found that the templates for uses cases and services supported by the tool and the writing guidelines helped avoid several typical problems encountered when training novices [24]:

The use cases were written from the actor's point of view, as the first step of every use case was usually an actor step.

The causality between steps was clear most of the time, as the writing guidelines encouraged students to write flow of events as an alternating sequence of actor actions and system responses.

The naming of actors, user tasks, use cases, and services was consistent (noun phrases for actors, verb phrases for the others).

Most exceptions were identified and handled as alternate flow of events.

While the distinction between user tasks and use cases is now clear to participants, there are still open questions about the granularity of use cases and services. Both templates are still similar (both use cases and services have flow of events), and often, participants model services as short low-level use cases. We will address these remaining issues by improving our use cases writing guidelines and by providing more detailed examples in our tutorial.

Specifying NFRs

We found that the three types of NFRs and guidance in the form of examples of NFRs made it easier to train novices to correctly identify and attach NFRs to the correct element in the specification. Moreover, the tool support for automatically relevant NFRs during justification increased the number of criteria taken into account during the assessment of options.

However, the set of NFRs that the participants identify is still incomplete. The organization of NFRs into a refinement graph as in the NFR Framework [12] would help better address the completeness issue.

Justifying use cases and services

Usually, justifications do not come naturally as a side effect of development. This is consistent with other studies and is a well-known obstacle to the wide spread use of rationale [6]. By explicitly adding the justification activity in the process, differentiating justification questions from other questions, and training developers to enter justifications as part of the deliverables, we were able to capture quite a large rationale (e.g., all use cases justified after the second iteration, all justifications including 2 or more options). While justifications cost additional overhead, we found that there are concrete incentives for including justifications on use cases.

For example, the question associated with use case justifications was phrased as explaining how a use case satisfies better the NFRs than other possible use cases. When assessing the current use case with alternate options, the assessments did not clearly indicate why the current solution was better. One of two things would then

occurred: either the author revised the use case to improve it or identified missing NFRs, adding columns to the assessment matrix in the justification, and thus making clearer the selection of the current solution. In both cases, the specification was improved.

In the last two case studies, we added tool support for selecting the initial set of criteria that are included in a justification. For example, when justifying a use case, the domain properties associated with the realized user task and the quality requirements associated with the use case were automatically included in the specification. The users were offered to expand or restrict the set of NFRs in the matrix. In general, we observed that this helped minimizing the occurrences of missing criteria in justifications.

Reviewing specification

In our case studies, more than half of the questions was generated during review by the instructors, the coaches, and the authors. Of these questions, half were request for clarifications and reports of omissions, which, once the specification is revised to resolve these questions, did not contain much useful rationale. We found that novices were able to correctly classify their questions, which in turn made it easier for us to find the questions that contain the most useful rationale. The type associated with questions also made it easier for reviewers to correctly phrase their questions and subsequently for developers to revise the requirements specification or the justifications accordingly.

However, the elaboration of complex questions by a single reviewer can be laborious. For example, if a reviewer enters an inconsistency question referring to two different use cases, enters several different alternatives for addressing the inconsistency, and assesses the alternatives against all relevant criteria, the reviewer will have to go through a series of five different forms. While a developer familiar with the process can specify the question efficiently, the length of the process may discourage a novice. We believe, however, that the reviewer can see early the benefit of investing the time in documenting complex questions, as it makes it easier for the developer to revise the requirements specification (and hence, minimize the number of review cycles).

Addressing challenges and clarifications

We found that attaching challenges and clarifications provided an effective way to track defects in the specification and their resolution by the responsible authors. The rationale side of the tool effectively acted as a long to do list that could be viewed by status, author, and relevant requirements element. In all four case studies, however, developers collaborated among themselves mostly outside the tool, that is, they did not request clarifications or challenge each others' use cases when defects were identified. Instead, those were addressed in meetings and subsequent changes were made to the use cases.

We believe this lack of collaboration through the tool was due, in part, to the lack of features typically offered by newsgroups or E-mail. Once a question was posted, it was not always obvious who the target of the question was and what actions were expected. Some developers attempted to indicate this with comments, but this was not a common case. While our focus does not directly include supporting distributed collaboration, we plan to improve collaboration or management support to increase the opportunities to capture critical rationale in the form of requests for clarification and challenges on content. Such rationale could then be restructured and formalized by use case authors and rationale maintainers into consolidated justification questions.

Lessons learned summary

We observed that the use case writing guidelines and the incremental teaching of the processs concepts helped participants write better use cases and better rationale. We found that adopting an incremental training enabled participants to master the process more quickly. For example, the process in the last study was composed of the following sequential steps:

Students develop a first version of the use cases.
Instructors review of the form of the use cases.
Students justify the use cases.
Instructors review of the content of the use cases and the justifications,
Students specify and justify services.
Students review and consolidate of the complete specification

By the end of this process, students mastered both the use case specification and the justification tasks. By alternating the focus on each technique, we were able to emphasize and illustrate the benefits of each guideline and process feature. Moreover, once the participants mastered the process, the use of the tool did not incur any problem.

However, we also found lost opportunities for developing NFRs and for capturing rationale. We hope to address the first set of issues by revising our model of NFRs and the second set of issues by providing better collaboration support in the tool.

Related work

The integration of rationale and requirements specification is not new. Several proposals from the requirements literature have included the capture and use of rationale information for addressing a variety of goals, such as improving traceability [25,26,27], driving elicitation [28,29,30,31], supporting negotiation [32], and supporting process improvement [33,34]. While many aspects in these proposals appear similar to REQuest, each differs fundamentally either in the goal they achieve or their approach. In this section, we examine how our work complements and extends these proposals.

REMAP was one of the first rationale approaches focusing on requirements [25]. The goal of REMAP was to support the traceability of requirements to design objects.

Researchers studied how individuals and teams of information systems professionals make requirements decisions. They initially used the IBIS model, including the issue, position, and argument nodes, and extended it with nodes for representing constraints, assumptions, decisions, requirements, and design objects. The prototype REMAP tool enables developers to represent requirements and their rationale as evolving graphs and replay decisions. In addition, the REMAP tool includes a truth maintenance system, which propagates the belief status of each node based on new changes. For example, invalidating an assumption modifies the validity of positions that rely on the assumption, and may prompt developers to reopen closed issues. The REMAP tool has since been extended to better support collaboration among developers and link external material, such as email, video, documents, and so forth [26,27]. While the goals of REMAP and REQuest are similar (capturing rationale for long term use), there are two essential differences between REMAP and REQuest: the representation of rationale and the relationship between rationale and requirements models. REMAP uses IBIS [14] to represent rationale while REQuest uses QOC [15]. IBIS follows the natural flow of argumentation during which participants express arguments for or against individual alternatives. QOC, on the other hand, focuses on the systematic evaluation alternatives against a set of criteria that is relevant to a question. QOC is a consolidated representation for long-term rationale, as it makes explicit the criteria that were considered during assessments. The second difference is the use of NFRs in the requirements model as criteria in the rationale model. The result is that REQuest puts a greater emphasis on NFRs and their relationship with functional requirements.

The *Inquiry Cycle* [28] is a class of methods for incrementally refining and reviewing requirements, using scenarios and rationale during elicitation. The goal of the Inquiry Cycle is to improve the quality of the requirements for evolving systems. The Inquiry Cycle includes the cyclical application of three steps: expression of semantic or episodic ideas (i.e., scenarios), criticism (i.e., raising and resolving issues), and refinement (e.g., addition of detail, decomposition, and corrections). Scenarios are derived from the current requirements as concrete material to provoke discussion and raise issues. The discussion of issues leads to changes in the requirements. *ScenIC* [29] is an instance of the Inquiry Cycle that provides detailed guidelines for each step. The Inquiry Cycle and ScenIC use rationale as a short-term working memory for discussing and keeping track of open issues and decisions to be implemented. Requirements and rationale in terms of objectives, tasks and obstacles are identified and elaborated supported by scenario analysis. Although researchers point out that it is possible to structure and archive the working memory as rationale, the details on how to achieve this restructuring have not been explored (and are not within the goals of the method). Our approach attempts to address the issue of converting the short-term working memory into a longer-term rationale record, and, hence, support the longer-term goal of supporting changes in later phases of development.

SCRAM [30,31] is an elicitation method that presents stakeholders with a combination of scenarios, conceptual demonstrators, and the rationale of specific issues. The goal of SCRAM is to improve stakeholder participation during elicitation sessions by exposing the stakeholders with rationale information. For selected issues, developers present the stakeholder with a complete rationale represented in the form of a QOC model. Different alternatives (in addition to the one illustrated by the

concept demonstrator) are documented together with their evaluation against a set of criteria relevant to the issue. The reason for presenting explicit rationale to stakeholders is to check if the selected set of criteria reflects their position and if the evaluation of different alternatives was done correctly. Sutcliffe [30] observed that, with trained facilitators, the availability of rationale lead stakeholders to ask more questions and more open-ended questions during sessions. Although SCRAM appears similar to REQuest (use of QOC to represent rationale, representation of NFRs as criteria), SCRAM and REQuest address different goals. Hence, SCRAM builds focused QOC graphs for selected decisions to be validated by the user, while REQuest systematically builds justifications for all use cases in the specification. However, by attempting to generalize the results from SCRAM, we propose that the review of use cases and system services can be improved by the availability of justifications. As a side effect, this also results in more extensive documentation for later phases of development.

WinWin [32] is a spiral approach to software development based on Boehm's spiral model. The goal of WinWin is the early identification and resolution of conflicts among stakeholders. Stakeholders post their "win" conditions (i.e., conditions that must be satisfied by the system in their view) using the WinWin groupware tool. A facilitator, with the help of the tool, identifies conflicts, which are then resolved by negotiation among stakeholders. The negotiation and its resolution are captured as an issue model listing issues, alternatives, and decisions. REQuest is similar than WinWin in its inclusion of NFRs in the rationale model (win conditions include NFRs). REQuest, however, differs from WinWin in that WinWin focuses on the higher level task of identifying a set of win conditions that all stakeholders can agree with. REQuest focuses on the detailed development of a requirements specification and its evaluation and justification against this set of win conditions (represented as criteria).

FOOM is a formal object-oriented method for specification that was recently complemented with the use of IBIS and QOC for capturing rationale [33,34]. During a series of case studies, researchers observed that the requirements engineering process can be thought as a series of refinement steps, during which the requirements increase in complexity, punctuated by crisis points, during which the requirements are drastically simplified and restructured as a consequence of new insights. In FOOM, IBIS was used to capture ad hoc rationale during the refinement steps, while QOC was used during crisis points to consolidate this rationale. While this study did not address cost and acceptance issues introduced by the systematic capture and consolidation of rationale, it provides evidence of the potential benefits of making rationale available to developers (e.g., better support during drastic restructuring) and managers (e.g., process monitoring and process improvement). REQuest is similar to FOOM in that - using QOC - it consolidates rationale that has been captured during the requirements engineering process. However, the FOOM effort concentrated on understanding the requirements engineering process and the potential uses for rationale information, while REQuest has also focused on guidance, acceptance, and tool support issues related with capturing and consolidating rationale information. Also, since REQuest is based on use cases, we hope that it is more immune to the drastic restructuring observed in FOOM (which is based on object models), enabling users to incrementally formulate and consolidate rationale on a use case basis as

opposed to a system wide basis. However, we will have to evaluate this hypothesis empirically on longer running studies.

The *NFR Framework* [12] is a method for systematically refining and elaborating NFRs. From a set of high-level NFRs (called *softgoals*) requirements engineers develop more detailed NFRs organized into an AND-OR graph. Requirements engineers then evaluate different options (called *operationalizations*) for their level of satisfaction against the NFRs and examine the interactions between conflicting NFRs. Since most high-level NFRs are rarely qualities that are either met or not, links in the NFR graph represent the degree an NFR contributes to or hinders another NFR. An NFR is *satisficed* (as opposed to satisfied) when the selected option meet the NFR within acceptable limits. REQuest is similar to the NFR Framework in its emphasis of criteria. However, in the NFR Framework, the NFR drive the requirements elicitation. Therefore, it has a much richer representation and set of techniques for operationalizations and for dealing with dependencies among NFRs. In addition, it focuses on the automatic evaluation of the NFR-graphs to determine the impact of decisions. Again, we complement this approach by an emphasis on functional requirement elicitation in terms of user tasks, use cases, and services and by an emphasis on a simpler rationale representation suitable for use in subsequent development tasks.

Conclusion

In this paper, we described guidance and tool support for integrated use case specification and rationale capture as well as four case studies where we have evaluated the tool and the guidance.

We hope to have completed the guidance on use case specification and rationale capture. So we will focus on rationale usage during the winter software engineering project course at TUM. In particular, we are setting up an experiment where two groups of students are required to do some changes on the specification, one group with rationale, the other without. To further study collaboration during requirements engineering, we also plan a distributed case study where students from Kaiserslautern and TUM collaborate for the specification, only by way of the tool. Finally, to evaluate the rationale maintenance part of the process, we plan a case study where we first ask students to produce a first version of the specification, perform rationale maintenance, and then introduce a change in the problem statement. This would give us preliminary results indicating whether or not the rationale maintenance process as currently defined is feasible and produces rationale that can be used during requirements changes.

It is generally recognized that case studies and experiments with students are limited when testing the effectiveness of a process or a tool and for generalizing results to the population of software developers. However, to our experience qualitative case studies using novices as subjects can lead to improvements in both tool support and guidance. To support the claim of the practical usefulness of the

process and tool, we plan to do experiments with practitioners after we have confirmed the usefulness of the rationale.

Acknowledgement

We thank our students for their interest, time, and effort during the case studies. We thank Daniela Ahlisch and Kagan Aksit for their contributions to the REQuest prototype. We thank the anonymous reviewers of REFSQ'2001 and of the Requirements Engineering Journal for their numerous constructive comments and suggestions, which have helped us improve this paper. Finally, we thank the Fraunhofer Institute of Experimental Software Engineering and Prof. Bruegge at the Chair of Applied Software Engineering at TUM who have continuously supported this work over the past two years.

References

1. Seminar "Steuergeräte-Design im Automobilbau und in der Industrieautomation", Haus der Technik, Essen, 24.-25.5.2000.
2. A. van Lamsweerde, R. Darimont & Ph. Massonet. "Goal-directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt", Int. Symp. on Requirements Engineering, pp. 194-203, 1995.
3. I. Jacobson, M. Christerson, P. Jonsson, & G. Overgaard, Object-Oriented Software Engineering—A Use Case Driven Approach. Addison-Wesley, Reading, MA, 1992.
4. I. Jacobson, G. Booch, & J. Rumbaugh, The Unified Software Development Process. Addison-Wesley, Reading, MA, 1999.
5. L.L. Constantine & L.A.D. Lockwood, "Structure and Style in Use Cases for User Interface Design", to appear in M. van Harmelen (ed.), Object-Oriented User Interface Design, 2001
6. S. Buckingham Shum & N. Hammond, "Argumentation-based design rationale: what use at what cost?" International Journal of Human-Computer Studies, vol. 40, pp. 603–652, 1994.
7. A. H. Dutoit & B. Paech, Rationale Management in Software Engineering, in X.Chung (ed.), Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, 2001
8. A. H. Dutoit & B. Paech, "Supporting Evolution: Rationale in Use Case Driven Software Development," in International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ'2000), Stockholm, June, 2000.
9. A. Cockburn, "Writing Effective Use Cases", Addison Wesley, 2001
10. D. Diaper (ed.) , "Task Analysis for Human-Computer Interaction", Ellis Horwood, 1989
11. "Benutzer-orientierte Gestaltung interaktiver Systeme," Normentwurf, DIN EN ISO 13407, 1998.
12. L. Chung, B.A. Nixon, E. Yu, & J. Mylopoulos. „Non-Functional Requirements in Software Engineering". Kluwer Academic, Boston, 1999.
13. M. Jackson, "Software Requirements & Specifications", Addison-Wesley, 1995
14. W. Kunz & H. Rittel, "Issues as elements of information systems," Working Paper No. 131, Institut für Grundlagen der Plannung, Universität Stuttgart, Germany, 1970.
15. A. MacLean, R. M. Young, V. Bellotti, & T. Moran, "Questions, options, and criteria: Elements of design space analysis," Human-Computer Interaction, vol. 6, pp. 201–250, 1991.

16. J. Lee, “Design Rationale Systems: Understanding the Issues,” in IEEE Expert, pp. 78–85, May/June 1997.
17. J. Conklin & K. C. Burgess-Yakemovic, “A process-oriented approach to design rationale,” Human-Computer Interaction, vol. 6, pp. 357–391, 1991.
18. J. Lee, “A qualitative decision management system,” Artificial Intelligence at MIT: Expanding Frontiers. P.H Winston & S. Shellard (eds.) (MIT Press, Cambridge, MA,) Vol. 1, pp. 104–133, 1990.
19. The Softbicycle Company. QuestMap: The Wicked Problem Solver. http://www.softbicycle.com/.
20. Javasoft. Java Servlet Specification. Javasoft. http://www.javasoft.com/
21. Object Management Group (OMG), XML Meta Interchange (XMI). OMG, November 2000. http://www.omg.org/
22. DOORS, Telelogic, http://www.telelogic.com/index.cfm
23. B. Bruegge, A.H. Dutoit, R. Kobylinski, & G. Teubner. “Transatlantic Project Courses in a University Environment,” Asian Pacific Software Engineering Conference, Singapore, December 2000.
24. S. Lilly, “Use Case Pitfalls: Top 10 Problems from real Projects using Use Cases, Technology of object-oriented languages and systems, pp. 174-183, 1999.
25. B. Ramesh & V. Dhar, Representing and Maintaining Process Knowledge for Large-Scale Systems Development, IEEE Expert, pp. 54-59, April 1994.
26. B. Ramesh, & K. Sengupta. “Multimedia in a design rationale decision support system.” Decision Support Systems, 19, 1995.
27. B. Ramesh, & A. Tiwana. “Supporting Collaborative Process Knowledge Management in New Product Development Teams.” Decision Support Systems, 27, pp. 213–235, 1999.
28. C. Potts, K. Takahashi, & A. I. Anton, Inquiry-based requirements analysis, IEEE Software, vol. 11, no. 2, pp. 21–32, 1994.
29. C. Potts, ScenIC: A Strategy for Inquiry-Driven Requirements Determination, International Symposium on Requirements Engineering, RE’99, pp. 58–65,1999.
30. A. Sutcliffe, Requirements Rationales: Integrating Approaches to Requirement Analysis, In Olson G.M., Schuon S, (eds.) Proc. of Designing Interactive Systems, DIS’ *95*, pp. 33–42, ACM Press, New York, 1995.
31. A. Sutcliffe & M. Ryan, Experience with SCRAM, a SCenario Requirements Analysis Method, In Proc. of the 3rd International Conference on Requirements Engineering, pp. 164–171, April 1998.
32. B.Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, & R. Madachy, “Using the WinWin Spiral Model: A Case Study,” IEEE Computer, pp. 33–44, July 1998.
33. L. Nguyen, P.A. Swatman & G. Shanks, Using Design Explanation within the Formal Object-oriented Method, Requirements Engineering, no. 4, pp.152-164, 1999.
34. L. Nguyen, & P.A. Swatman. Managing the Requirements Engineering Process, 7th International Workshop on Requirements Engineering: Foundation for Software Quality. Interlaken, Switzerland. 2001.
35. A. Anton & C. Potts. ”The Use of Goals to Surface Requirements for Evolving Systems,” International Conference on Software Engineering, pp.157-166, Kyoto, 1998.