Developing Guidance and Tool Support for Rationale-based Use Case Specification

Allen H. Dutoit[*] and Barbara Paech[°]

[*]Technische Universität München, Institut für Informatik, 80290 Munich, Germany
dutoit@in.tum.de

[°]Fraunhofer Institute for Experimental Software Engineering, 67661 Kaiserslautern, Germany
paech@iese.fhg.de

Abstract. Few requirements engineering techniques are widely applied in industry. This can–at least to some extent–be attributed to lack of dedicated guidance, tool support, and technique integration. In this paper, we describe dedicated tool support and guidance for rationale-based use case specification, which we incrementally derived from several students experiments. Through this example, we want to promote student experiments as a good basis for developing and improving guidance material and tool support, hence facilitating technology transfer.

1 Introduction

There is a wide variety of techniques for elicitation, specification, validation, and management of requirements, but only few of them are used in industry. For example, at a seminar given last year 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 [18]. Also, the experience from several industry projects the authors were involved in, 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 it,
- this information is often inconsistent, ill-structured, and imprecise,
- the authors of the specification did not find an adequate level of abstraction avoiding design decisions, but 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:

- *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.
- *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 dedicated tool support for the capture and management of use cases.
- *Smooth integration among the techniques applied.* The lack of integration among techniques and with the rest of the process 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.

In this paper, we describe such guidance, tool support, and integration for rationale-based use case specification. Our ultimate research goal is to support the evolution of software by making available to developers rationale captured during requirements and kept up-to-date throughout the software life cycle [10]. However, before we can work towards that goal, we first need to understand the details of applying use case specification and rationale capture to a realistic problem. We have done this by developing a dedicated tool integrating both use case specification and rationale capture. We have evaluated and refined this tool in the context of case studies conducted with students in project courses.

Literature about experiments in software engineering (e.g., [2][20]) warns about the threats to the validity of experiments using student subjects. The main argument is that the experience of the subjects in skills that are relevant to the study must be representative of the population to which the results will be generalized, which is not always the case with student subjects. The case studies described here, however, have given us much insight on how to improve our approach. They also reinforce the position above that guidance and integrated tool support are essential. By evaluating the tool with students, we have the opportunity to develop and improve guidance material for the tool and the process. Student experiments, in this case, are especially valuable for deriving guidance *because* of the general lack of specific experience of students, which would have allowed them to apply requirements engineering techniques without guidance and integration [2]. Therefore, if the guidance and integration is appropriate for the students, it will most likely also be appropriate for practitioners.

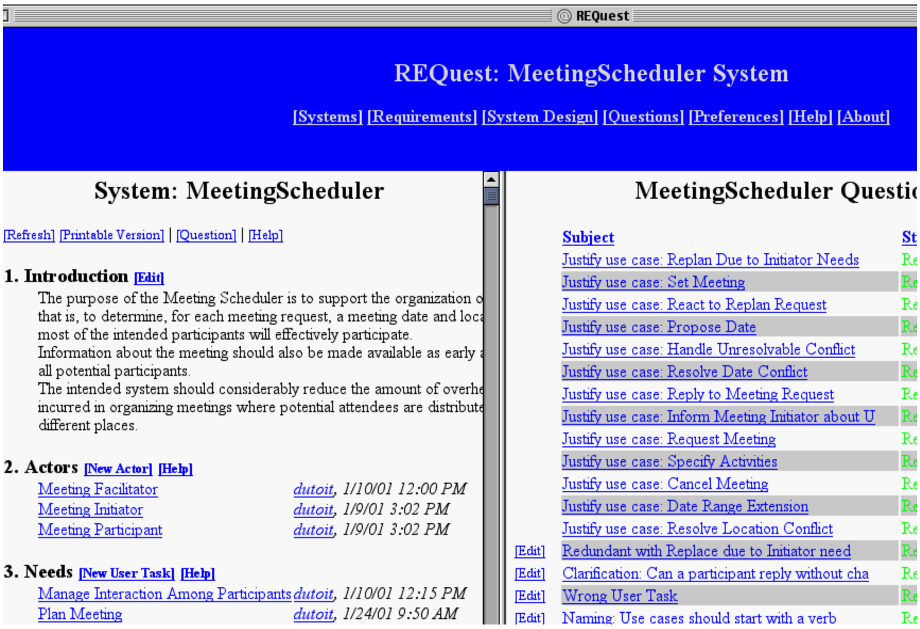
Thus, the purpose of this paper is twofold: First, we describe dedicated tool support for use case specification and rationale capture. Second, with this example, we want to promote the idea of student experiments as a means of refining requirements techniques and tools with specific guidance and integration, hence facilitating technology transfer. In Sect. 2, we describe our assumptions, the tool we developed, and its evaluation setting. We discuss the problems encountered with use case specification in Sect. 3, as well as the guidance we devised to overcome these problems. Similarly, in Sect. 4, we describe the problems encountered with rationale capture and the resulting improvements. In Sect. 5, we describe the problems with the integration and the details involved in improved integration. The conclusion summarizes the guidance and tool support derived from the case studies.

2 Experimental Environment

In [10], we described an integrated process for use case specification and rationale capture, as well as the requirements for a tool supporting this process. Developers use the tool for describing a problem statement (we assume a meaningful problem statement has already been negotiated with the client) and refining the problem statement into a requirements specification. Developers and reviewers have the opportunity for entering rationale information, as a side effect of specification, collaboration, and review. In the following, we give an overview on the tool (Sect. 2.1) and describe the context for the two case studies, a software engineering project course (Sect. 2.2) and a requirements engineering seminar (Sect. 2.3).

2.1 Tool Overview

The design goals of the first version of the tool were to provide a simple and integrated solution to manipulate use case and rationale models without embedding process specific assumptions. The tool is a web application that can be accessed via most popular 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/menubar, a requirements specification view and a rationale view (see Fig. 1.).


REQuest
[Systems] [Requirements] [System Design] [Questions] [Preferences] [Help] [About]
[Refresh] [Printable Version] | [Question] | [Help]

REQuest: MeetingScheduler System

System: MeetingScheduler

1. Introduction [Edit]

The purpose of the Meeting Scheduler is to support the organization o
that is, to determine, for each meeting request, a meeting date and loca
most of the intended participants will effectively participate.
Information about the meeting should also be made available as early a
all potential participants.
The intended system should considerably reduce the amount of overhe
incurred in organizing meetings where potential attendees are distribute
different places.

2. Actors [New Actor] [Help]

Meeting Facilitator	dutoit, 1/10/01 12:00 PM
Meeting Initiator	dutoit, 1/9/01 3:02 PM
Meeting Participant	dutoit, 1/9/01 3:02 PM

3. Needs [New User Task] [Help]

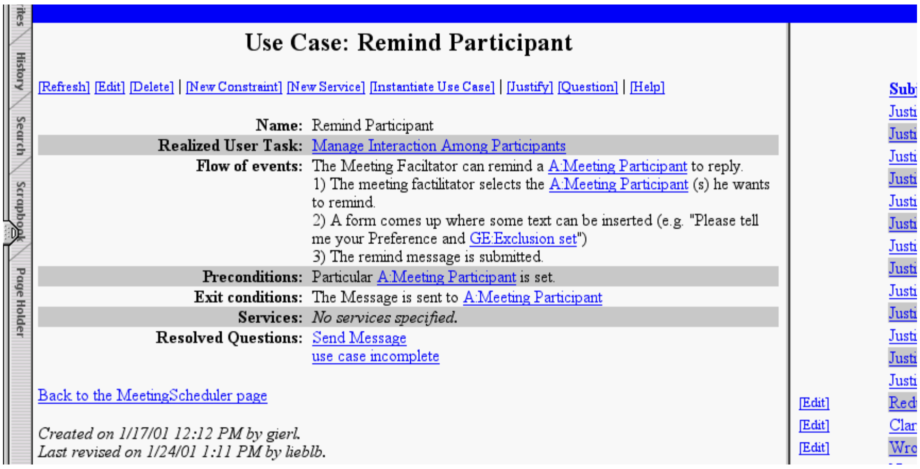
Manage Interaction Among Participants	dutoit, 1/10/01 12:15 PM
Plan Meeting	dutoit, 1/24/01 9:50 AM

MeetingScheduler Questi

	Subject	St
	Justify use case: Replan Due to Initiator Needs	Re
	Justify use case: Set Meeting	Re
	Justify use case: React to Replan Request	Re
	Justify use case: Propose Date	Re
	Justify use case: Handle Unresolvable Conflict	Re
	Justify use case: Resolve Date Conflict	Re
	Justify use case: Reply to Meeting Request	Re
	Justify use case: Inform Meeting Initiator about U	Re
	Justify use case: Request Meeting	Re
	Justify use case: Specify Activities	Re
	Justify use case: Cancel Meeting	Re
	Justify use case: Date Range Extension	Re
	Justify use case: Resolve Location Conflict	Re
[Edit]	Redundant with Replace due to Initiator need	Re
[Edit]	Clarification: Can a participant reply without cha	Re
[Edit]	Wrong User Task	Re
[Edit]	Naming: Use cases should start with a verb	Re

Fig. 1. Overview of tool: 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, and non-functional properties [10]. Actors and user tasks describe the application domain from the user's point of view and independently from the system. The services describe the features of the system, independently from the user. The use cases describe how user tasks are realized in terms of services and provide traceability from the problem statement to the specification. Non-functional properties can be used to describe non-functional requirements on the system or to describe facts of the domain that are more easily expressed as properties than task descriptions. The user can also provide examples in terms of actor instances and scenarios and define important terms using a glossary. The tool provides templates, text boxes, and selection menus for each requirements element. The tool recognizes known terms and highlights them automatically in text fields (e.g., the flow of events of a user task or a use case). For example, Fig. 2. displays the detailed view of a use case.



Use Case: Remind Participant

[Refresh] [Edit] [Delete] | [New Constraint] [New Service] [Instantiate Use Case] | [Justify] [Question] | [Help]

Name: Remind Participant
Realized User Task: Manage Interaction Among Participants
Flow of events: The Meeting Faciltator can remind a A:Meeting Participant to reply.
1) The meeting factilitator selects the A:Meeting Participant (s) he wants to remind.
2) A form comes up where some text can be inserted (e.g. "Please tell me your Preference and GE:Exclusion set")
3) The remind message is submitted.
Preconditions: Particular A:Meeting Participant is set.
Exit conditions: The Message is sent to A:Meeting Participant
Services: *No services specified.*
Resolved Questions: Send Message
use case incomplete

Back to the MeetingScheduler page

Created on 1/17/01 12:12 PM by gierl.
Last revised on 1/24/01 1:11 PM by lieblb.

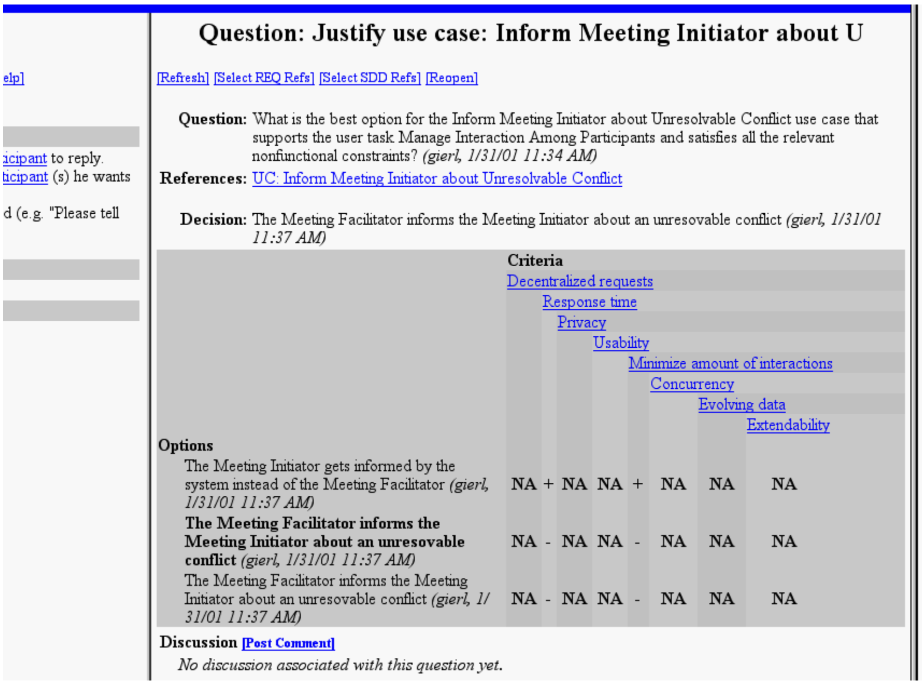
Fig. 2. Detailed use case view. The user is provided a template for each requirements elements. Questions can be associated to any element.

In the rationale view, information is structured according to the Questions, Options, Criteria (QOC) paradigm [14] and displayed as tables and hyperlinks, thus maximizing the density of information that the user can read in a single screen (see Fig. 3.). A single page describes the question under consideration, the decision (if any) that resolves the question, and a table describing the trade-offs that were evaluated. The rows of the table are the different options that were considered. The columns of the table are the criteria relevant to the question that were used to evaluate the options. The table cells contain the assessments of each option against each criteria.

Displaying rationale as text is a different approach than other well-known rationale-based tools (e.g., gIBIS[7], SYBIL[12], QuestMap[19]), which display rationale as a graph. In addition to the QOC structured information, users can annotate questions with informal comments or arguments (with the [Post Comment] feature) to provide reference information or negotiate various aspects of the question.

An important assumption behind our process is that requirements and rationale must be tightly integrated for decreasing the overhead of rationale capture and increasing its utility. This integration is visible in the tool in two ways: the interlinking of requirements elements and questions, and the interlinking of non-functional properties and criteria.

Linking of requirements and rationale elements. the user can create questions in two ways: by challenging a specific requirements element with the [Question] feature or by justifying a requirements element with the [Justify] feature (see buttons at the top of the use case view in Fig. 2.). By clicking on [Question], the user is presented a series of forms to enter a question, its relevant criteria, one or more options, and the assessments of the options against the criteria. Users use [Question] to request a clarification, express a disagreement, or more generally, to initiate a discussion. Once other users have contributed to the question and a consensus is reached, the user can close the question with a decision. By clicking on [Justify], the user is presented with a similar series of forms, except that the resulting question is closed and the current



Question: Justify use case: Inform Meeting Initiator about U

[Refresh] [Select REQ Refs] [Select SDD Refs] [Reopen]

Question: What is the best option for the Inform Meeting Initiator about Unresolvable Conflict use case that supports the user task Manage Interaction Among Participants and satisfies all the relevant nonfunctional constraints? *(gierl, 1/31/01 11:34 AM)*

References: UC: Inform Meeting Initiator about Unresolvable Conflict

Decision: The Meeting Facilitator informs the Meeting Initiator about an unresovable conflict *(gierl, 1/31/01 11:37 AM)*

Options	Decentralized requests	Response time	Privacy	Usability	Minimize amount of interactions	Concurrency	Evolving data	Extendability
The Meeting Initiator gets informed by the system instead of the Meeting Facilitator *(gierl, 1/31/01 11:37 AM)*	**NA**	**+**	**NA**	**NA**	**+**	**NA**	**NA**	**NA**
The Meeting Facilitator informs the Meeting Initiator about an unresovable conflict *(gierl, 1/31/01 11:37 AM)*	**NA**	**-**	**NA**	**NA**	**-**	**NA**	**NA**	**NA**
The Meeting Facilitator informs the Meeting Initiator about an unresovable conflict *(gierl, 1/31/01 11:37 AM)*	**NA**	**-**	**NA**	**NA**	**-**	**NA**	**NA**	**NA**

Discussion [Post Comment]

No discussion associated with this question yet.

Fig. 3. Detailed question view. Users can collaborate by adding options, criteria, assessments, and references to elements.

alternative is described and used as a decision. The resulting question can only be discussed if it is first reopened. [Question] is used to open a discussion with multiple users and hence collaboratively building a rationale behind one or more evolving requirements element; while [Justify] is used (usually by a single user) to enter the justification of the current version of a requirements element.

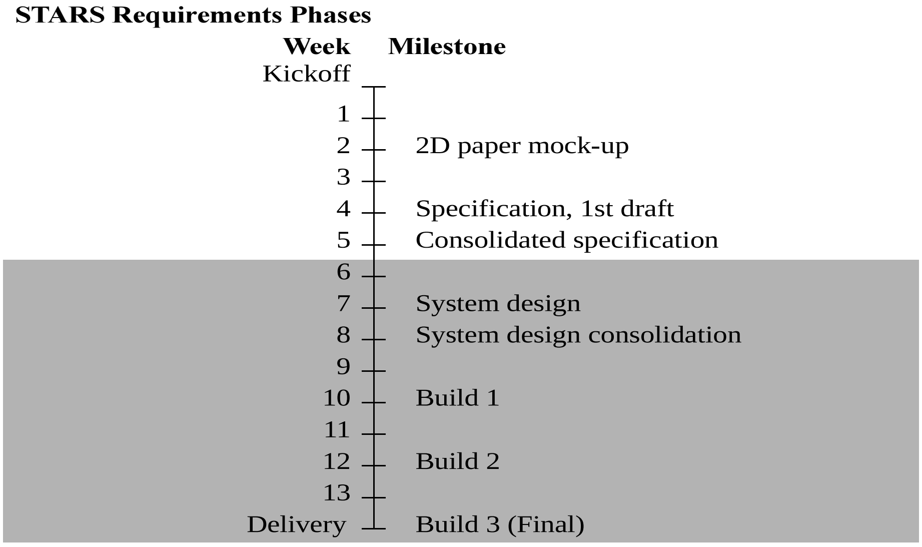
When creating questions, either with [Question] or [Justify], the tool stores a bidirectional link between the questioned requirements element and each of its corresponding questions. This enables the user to navigate and update related questions or related elements quickly.

Linking of non-functional properties and criteria. Non-functional properties are treated as criteria in the rationale view (Fig. 3.). This enables the user to select non-functional properties that are relevant to a given question. These non-functional properties are then included as columns in the assessment table as a regular criteria. Each option can thus be evaluated according to the degree of satisfaction of each non-functional property. Links between the columns in the rationale view and the non-functional properties in the requirements view enables the user to view detailed descriptions of non-functional properties and their context. Note that, while all non-functional properties correspond to a criteria, not all criteria correspond to a non-functional property. Some criteria, for example, can originate from other tools (e.g., design goals during system design).

Finally, the tool has minimal process knowledge, thus to increase the flexibility of the tool and the evaluated processes: Users can manipulate any requirements elements and rationale elements in any order and at any time. Process guidance is provided instead through training and on-line help documentation within the tool.

2.2 STARS Case Study

The tool was first used in the software engineering project course offered at Technische Universität München (TUM) [5]. 22 students divided into four teams developed STARS (Sticky Augmented Reality Technology System), a prototype augmented reality application for nuclear powerplant technicians. Three of the four teams (15 students) were involved in the requirements engineering of the system, which lasted five weeks (Fig. 4.).



STARS Requirements Phases

Week	Milestone
Kickoff	
1	
2	2D paper mock-up
3	
4	Specification, 1st draft
5	Consolidated specification
6	
7	System design
8	System design consolidation
9	
10	Build 1
11	
12	Build 2
13	
Delivery	Build 3 (Final)

Fig. 4. Requirements engineering phases and schedule during STARS. Subsequent development phases (in gray) included for completeness.

The students were provided with a nuclear powerplant maintenance scenario illustrating the use of the STARS system and a general problem statement in terms of actors, user tasks, and non-functional properties. The instructors spent two lectures on requirements engineering in general and a 45 minute tutorial on using the tool. In addition to the on-line help, the students had access to a toy example describing the specification for a supermarket check out system. Coaches and instructors provided feedback to 2D mock-up and on the initial version of the specification during reviews and using questions in the tool. Students spent the final week of the requirements phase consolidating the specification by discussing and answering any remaining open questions.

We used four different source of data for the evaluation of the tool: logged user actions, user questionnaire, informal comments and discussions about the tool, and the final version of the requirements specification and its associated rationale. In using these different data, we focused only on qualitative aspects, as our goal was to uncover limitations and elicit ideas for improving the tool and its associated guidance.

The students produced a 24 page requirements specification using the tool, including 29 use cases. The quality of the requirements specification was average. For example, several use cases documented system design level features of the system and the granularity of the use cases was too small. Several weaknesses of the specification (discussed in Sect. 3) were traced to misunderstandings of the process and lack of distinctions between user tasks, use cases, and services.

The rationale information included 62 questions (with their associated options, decisions, comments, and criteria), spanning 40 pages of text. About half of this information was related to clarifications and omissions, and thus, would not be useful information downstream for system evolution. While students did understand relatively easily the basic concepts of the QOC paradigm, their use of comments and questions sometimes pointed to missing features in the tool (discussed in Sect. 4).

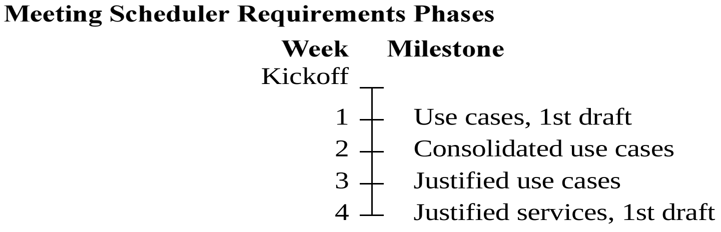
We observed that students used the tool as a mechanism for delivering specifications and for obtaining feedback from instructors. In some instances, the students used the tool to obtain clarifications from other students about their contribution to the specification. However, the tool was not used as a collaboration mechanism within the teams. Instead, they used face-to-face meetings and designated a volunteer to enter their decisions in the tool. In total, only about 6 students used the full range of the functionality provided by the tool.

2.3 Meeting Scheduler Case Study

In the STARS case study, several use cases did not have any questions attached, in other words, there was no rationale associated with them. This lead us to add the [Justify] feature for developers to explicitly justifying use cases and services (discussed in Sect. 2.1). This feature is similar to the feature for raising questions, except that it produces only closed questions. This feature was then evaluated in the next case study.

Four students taking a requirements engineering seminar at TUM spent four weeks developing a requirements specification for the meeting scheduler problem [11]. We structured the original problem statement into three actors, four user tasks, and 10 non-functional properties. The students were already familiar with requirements engineering methods and, hence, were given only a 15 minute tutorial on user tasks, use cases, and QOC.

Except for an explicit phase during which students justified their use cases, the requirements phases followed in this case study were the same as in STARS (Fig. 5.).



Meeting Scheduler Requirements Phases

Week	Milestone
Kickoff	
1	Use cases, 1st draft
2	Consolidated use cases
3	Justified use cases
4	Justified services, 1st draft

Fig. 5. Requirements engineering phases during Meeting Scheduler case study.

Similar to STARS, the students spontaneously decided to meet face-to-face once per week. The students also met once per week with the instructor (as part of their regular seminar duties), which provided an opportunity for clarifying the process and the concepts behind the tool. Given the smaller size of the forum and the better system development experience of the students, much fewer misunderstandings about the process occurred.

The same data collection methods were used as in STARS. This case resulted in a specification of better quality than STARS, including 20 pages and 18 use cases. The rationale information included 40 questions, including 13 justifications which were much likely to be useful during system evolution. Students spent more effort than in STARS on capturing and structuring this rationale.

3 Supporting Use Case Specification

In both case studies we had given the students a tutorial on user tasks and use cases. We did not provide them with a very specific use case template, since it was not our intention to develop and teach a new use case approach. However, it turned out that the students needed more guidance, that the textual representation in the tool was lacking means for use case structuring and that the use of non-functional requirements as criteria for the evaluation of options required more specificity for the non-functional requirements specification. In the following, we describe these problems in more detail together with planned improvements.

3.1 Use Case Guidance

We had emphasized to students that in use cases the *interaction* between system and actors should be described. However, the use cases written by the students suffered from several problems, some of them mentioned also in [13]. In particular,

1. the use cases were written from the system´s point of view,
2. the single steps of the flow of events were often indistinguishable,
3. actor names were inconsistent,
4. exceptions were described as separate use cases or omitted
5. the students wrote too many small use cases with too little coherence.

Another problem was that the difference between user tasks and use cases was not clear. So for example, use cases which described new functionality were added as new user tasks. Therefore, a lot of time planned for review and consolidation of the rationale actually went into review and consolidation of the form of user tasks and use cases.

To overcome these problems we will provide different templates for user tasks and use cases. The *new templates* are shown in Fig. 6 and 7.

The use case template is adapted from the *essential use cases* of [8], where each use case step has a number, and actor and system steps have to alternate. This way, we hope to avoid the first two problems. Explicit naming of actors and exceptions should alleviate the third and fourth problem. The includes relationships as well as preconditions and postconditions can be used to structure the set of use cases, thus supporting coherence between use cases.

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	Cancel Meeting, Handle Replies, Remind Participant, Set Meeting, React to Replan Request

Fig. 6. User Task Template

Use Case Name	Remind Participant	
Initiating Actor	Meeting Facilitator	
Realized User Task	Manage Interaction Among Participants	
Participating Actors	Meeting Participants	
Flow of events	Actors	System
	1. The Meeting Facilitator selects a scheduled meeting and the “Remind Participant”-Feature.	
		2. The System allows a choice between the meeting-participants.
	3. The Meeting Facilitator selects a participant.	
		4. The System allows the formulation of the message.
	5. The Meeting Facilitator writes the message and requests the message sending.	
		6. The System sends the message to the selected participant [Exception: Message Fails].
Exceptions	[Message Fails] If the System cannot the deliver the message, it notifies the Meeting Facilitator.	
Rules	Retry Message Sending for 5 hours	
Precondition	The meeting has been scheduled	
Postcondition	The meeting participant has been reminded through a message by the meeting facilitator	
Includes Use Cases	None.	
Used services	Message Send	

Fig. 7. Use Case Template

We will also provide tool support for filling in the templates. Thus, e.g. the actors can be selected from the set of actors already defined or a new actor can be created. In the latter case, this actor is simultaneously included in the list of available actors.

3.2 Support for Use Case Structure

To emphasize the structure and coherence of use cases we will encourage students to write for each user task one complex use case, possibly including other use cases. Another means to support the structure and the "big picture" of the use cases altogether, is to allow for a use case diagram. While it is beyond the scope of the current tool to automatically link the diagram elements with the specification elements, just providing the picture will be helpful to understand the overall structure. In addition use cases will be grouped according to user tasks, and services will be grouped (possibly with duplicates) according to use cases.

3.3 Guidance for Non-Functional Properties

During the case studies it became clear that there are different kinds of non-functional properties with different usage as criteria. By making this explicit we hope to focus the options, arguments and criteria provided by the students. Thus, we will use the property types described in Table 1. The table also indicates when the different criteria are used. So, for example, domain properties are used as criteria on how well use cases and services realize a user task.

In some sense these criteria can be viewed as goals for the system design. However, our types are much simpler than goal types in goal-oriented approaches to requirements engineering (e.g. GBRAM[1] or KAOS[11]). In contrast to these approaches we use user tasks instead of goals to drive the requirements elicitation and specification process. We only use the non-functional properties as criteria for the evaluation of the adequacy of use case or service design wrt. to user tasks and use cases, respectively.

Table 1. Types of non-functional properties.

Property type	Explanation	Used as criteria in questions for:
Domain Property	Facts of the domain to be adhered to by the software system (e.g. "a person may not be a two different places")	Use Cases, Services
Global functional properties	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")	Use Cases
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 as small as possible".	User Tasks, Use Cases, Services

4 Supporting Rationale Capture

Our ultimate goal is to provide rationale information to developers for supporting evaluation. Currently, we have evaluated how this rationale can be first captured by developers as a side effect of development (entering explicit justifications for specific requirements elements), collaboration (asking questions and negotiating solutions), and review (requesting changes and pointing out defects by raising questions). In the STARS case study, the initial version of the tool provided the same mechanism for all three types of activities, a simple matrix representation based on QOC (see Sect. 2). In the Meeting Scheduler case study, we added the [Justify] feature for supporting the creation of closed questions to better support justification. However, all questions resulting from the three activities were still represented using QOC.

When examining the questions generated during the case studies, we realized that QOC is too general a paradigm. Once a question was raised, users often did not know whether they should discuss the question, realize the first option, or simply comment on the relevance of the question. Moreover, for simple cases (e.g., a clarification question), most of the complexity of QOC is not needed (e.g., clarifications do not have alternative options or criteria associated with them).

For the next series of case studies, we are refining the rationale model to address the issues we encountered while supporting justification, collaboration, and review. In particular:

- Question authors specify the type of question being asked (challenge on form, challenge on content, clarification, inconsistency, justification, omission, see Table 2.).
- The set of available actions that can be invoked on a question is then restricted based on the type of question. For example, clarification questions can only be closed while challenges on content can have new options, criteria, and assessments.
- The tool provides additional views to filter questions by type. This enables, for example, a rationale maintainer to see only the questions that have potential use for long term rationale.
- Arguments can be attached to assessments in addition to questions. This enables users to focus on specific cells in the assessment matrix. This is critical when the QOC matrix alone does not seem to justify the selected decision. We will also focus on heuristics for specifying and reviewing assessment matrixes in the tutorial.

Table 2. summarizes the types of questions, the set of available actions for each type, and the potential value of each type of question for the rationale maintainer.

The taxonomy of questions in Table 2. results from the classification of the defects with which each question is associated.[1] This is a different taxonomy of question than that used in other rationale-based approaches, such as [3] and [16]. [16] uses a taxonomy based on the type of information missing from the specification (e.g., what-is, how-to, who, what-kinds-of, when) which encouraged reviewers to ask questions by identifying missing information, usually the hardest types of defects to identify. [3] uses a domain specific taxonomy that analysts develop as part of the requirements process. This enables stakeholders to identify quickly which questions are relevant to their win criteria. In our case, we decided to select the defect taxonomy given that our focus is on use case developers (as opposed to the reviewers or the clients) and provide them more guidance about what to do with the questions. Moreover, we do not need to create an explicit domain-based taxonomy as questions are already attached to specific requirements elements (which can also be glossary entries).

[1] With the exception of the Justification type which can be viewed as a subcase of the "Challenge on Content" type.

Table 2. Types of questions.

Question type	Relationship to requirements	Available actions	Value for rationale
Challenge on the form	Linked to one or more elements that do not comply to the structure supported by the tool (e.g., confusion between user tasks and use cases).	• Close question by revising related elements	None
Challenge on the content	Linked to one ore 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 the gaps.	Low

5 Integrating Specification and Rationale

When developing the tool and its associated process, we believe that only a tight integration between the requirements and the rationale activities can yield to a cost effective capture of rationale and a significant return on investment. While Sect. 3 and 4 discussed refinements in the requirements and the rationale models supported by the tool, in this section, we describe improvements in their integration. In the following, we describe the problems (and their planned remedies) we encountered for justification, collaboration, and review activities.

5.1 Supporting Justification

In STARS justifications did 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 [17]. In the Meeting Scheduler study, we provided the [Justify] feature for developers to enter rationale about use cases explicitly and made justifications part of the deliverables. While justifications cost additional overhead, we found that there are concrete incentives for including justifications on use cases. For example, when a justification was phrased as explaining how a use case satisfies better the non-functional properties than other versions of the same use case, missing (non-functional) requirements were found.

Currently, however, non-functional properties can only be created in the requirements view. It is then impractical and unintuitive to create new non-functional properties on the fly when writing a justification. We plan to modify [Justify] so that non-functional properties can be created as a side effect of writing a justifications. We believe this will improve the completeness of the requirements elements by the elicitation of more non-functional properties (especially domain facts) and improve the quality of the rationale information by documenting more accurately the trade-offs that were investigated.

5.2 Supporting Collaboration

In both case studies, developers collaborated mostly outside the tool, except for limited cases of collaboration between different teams. We believe this lack of collaboration through the tool was due to two main reasons: First, developers had the opportunity to meet face-to-face. Second, the tool lacked 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 the [Post Comment] feature, but this was not a common case. We plan to improve collaboration support within the tool by adding features for asynchronous and synchronous notification (e.g., by enabling users to send a link to a question via e-mail), and by embedding more information about user responsibilities.

During consolidation, each student (or team) was responsible for resolving the open questions related to their requirements elements. However, since several questions related to different use cases could interact, developers would have to read all the questions related to their elements to be able to comprehensively consolidate their requirements elements. Since questions could only be viewed by single attributes of questions (e.g., requirements element they are related to, time of modification, author, and status), this task was difficult when many questions were still open. We plan to add views into rationale that are more closely related to how responsibilities are assigned. For example, developers should be able to view all questions related to all requirements elements pertaining to a single user task, or to view all the questions by author of related requirements elements.

We also plan to add a facilitator attribute to questions that can be explicitly set by users and that can be used to sort questions. The facilitator of a question (which may be the author of the related requirements elements) is then understood by all users as the person responsible for facilitating the negotiation about the question, and, once consensus is reached, to close the question and revise the related requirements elements.

In general, however, it will not be possible to remove completely the need for face-to-face meeting, in which cases, we still need to capture the important questions that were raised during the meeting. This could be done by a minute taker taking notes as QOC matrixes [7][9] or by supporting the post-processing of meetings using QOC [4].

The definition of views could, of course, be alleviated by the use of a general-purpose requirements management tool like DOORS. A first version of the tool had also been implemented as a DOORS prototype. However, the effort to turn DOORS into a dedicated tool, e.g. with concurrent view of requirements and rationale, specific input and output templates for use cases and rationale, and especially the assessment matrix with options, criteria and arguments, was soon deemed too high for our purpose. Also, the students would have needed much more training on DOORS than they need on the current tool.

5.3 Supporting Review

In both case studies, more than half of the questions were 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 requirements specification is revised to resolve these questions, do not contain much useful rationale information. We also found some rationale that was buried in unrelated clarification questions, and justification questions were often missing assessments and alternative options. Consequently, a hypothetical rationale maintainer would have to spend a substantial effort filtering, completing, and restructuring this information to be useful during evolution.

By adding more information about questions, such as the question types described in Sect. 4 and the question facilitator described in Sect. 5.2., reviewers will be able to better focus their questions and elicit alternative options from specific developers. Moreover, when justifications are written by developers as part of their work, the reviewers will also be able to not only step through the requirements specifications but also the justification questions. The reviewer can comment on the justifications if non-functional properties are missing or if the assessments provided by the developers are not clear (with the argument on assessment feature described in Sect. 4). We anticipate that this will reduce the number of clarification questions while increasing the amount of discussion on the justifications.

6 Conclusion

In this paper, we described tool support for integrated use case specification and rationale capture as well as two case studies where we have evaluated the tool. The problems we encountered and the resulting proposed improvements (including enhanced tool features and improvements to the guidance) are summarized in Table 3.

We will evaluate the tool and its guidance again during the summer in several case studies using again the meeting scheduler as problem domain. With the input of these case studies we hope to have completed the guidance on use case specification and rationale capture, so that we can focus on rationale usage during the winter software engineering project course at TUM. 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.

It is generally recognized that case studies and experiments with students are limited when testing the effectiveness of a method or a tool and for generalizing to the population of software developers. However, we showed, using rationale-based use

Table 3. Summary of improvements for tool and guidance.

Technique	Goal	Tool Feature	Guidance
Use case specification	Distinction of user task and use case	Differentiated templates for use cases and user tasks	
	Improved flow of events	Refined use case template	Essential use cases
	Improved coherence and structuring	Pre- and postconditions, use case diagram	One use case for each user task, inclusion of other use cases
	Improved non-functional properties	Property types & restricted application	Restriction of applicability of types as criteria
Rationale capture	Improved response to questions	Question types, restricted actions, question facilitator	Restrictions of actions by question types
	Improved assessments	Arguments attached to assessments	Heuristics for reviewing assessments.
Use case/ rationale integration	Improved non-functional properties	On the fly creation of criteria during justification	Relationship between non-functional properties and criteria.
	Improved collaboration	E-mail notification, list of active users, responsibility-based views, global glossary	
	Lower maintenance effort	Question types View by question types	Review process includes review of justification

case specification as an example, how qualitative case studies using students as subjects can lead to improvements in both tool support and guidance. To support the claim of practical usefulness of the tool, experiments with practitioners have to be carried out.

References

[1] A. Anton & C. Potts.”The Use of Goals to Surface Requirements for Evolving Systems,” International Conference on Software Engineering, pp.157-166, Kyoto, 1998.

[2] V.R. Basili, F. Shull, & F. Lanubile. “Building Knowledge through Families of Experiments,” IEEE Transactions on Software Engineering, vol. 25, no.4, July/August, 1999.

[3] 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.

[4] A. Braun, B. Bruegge, & A.H. Dutoit. “Supporting Informal Meetings in Requirements Engineering” Submitted to REFSQ’2001, Interlaken, Switzerland, June 2001.

[5] B. Bruegge, A.H. Dutoit, R. Kobylinski, & G. Teubner. “Transatlantic Project Courses in a University Environment,” Asian Pacific Software Engineering Conference, Singapore, December 2000.

[6] S. Buckingham Shum, “Analyzing the Usability of a Design Rationale Notation,” in T.P. Moran & J.M. Carroll (eds.) *Design Rationale: Concepts, Techniques, and Use.* Lawrence Erlbaum, Hillsdale, NJ, 1995.

[7] J. Conklin & K. C. Burgess-Yakemovic, “A process-oriented approach to design rationale,” *Human-Computer Interaction*, vol. 6, pp. 357–391, 1991.

[8] 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

[9] A. H. Dutoit, B. Bruegge, & R. F. Coyne, “The use of an issue-based model in a team-based software engineering course,” Conference proceedings of *Software Engineering: Education and Practice (SEEP’96)*. Dunedin, New Zealand. January 1996.

[10] 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.

[11] 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

[12] 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.

[13] 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

[14] 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.

[15] T. P. Moran & J. M. Carroll (eds.), *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, NJ, 1996.

[16] C. Potts, K. Takahashi, & A. I. Anton, “Inquiry-based requirements analysis,” *IEEE Software*, vol. 11, no. 2, pp. 21–32, 1994.

[17] S. B. Shum & N. Hammond. “Argumentation-based design rationale: what use and what cost? International Journal Human-Computer Studies, 40:603-652.

[18] Seminar “Steuergeräte-Design im Automobilbau und in der Industrieautomation”, Haus der Technik, Essen, 24.-25.5.2000

[19] The Softbicycle Company. QuestMap: The Wicked Problem Solver. http://www.softbicycle.com/.

[20] M.V. Zelkowitz & D.R. Wallace. “Experimental Models for Validating Technology” IEEE Computer, May, 1998.