Using Rationale for Software Engineering Education

Allen H. Dutoit, Timo Wolf
Institut für Informatik
Technische Universität München
{dutoit,wolft}@in.tum.de

Barbara Paech, Lars Borner, Jürgen Rückert
Institute for Computer Science
University of Heidelberg
{paech,lars.borner,juergen.rueckert}@
informatik.uni-heidelberg.de

Abstract

Software engineering courses often use industrial modeling tools for their infrastructure, as it exposes students to the state-of-the practice and increases their awareness about the complexity of their craft. However, this approach has the risk of expending a disproportionate amount of effort on tools as opposed to teaching concepts. Moreover, industrial tools often do not include didactic concepts needed for education.

In this paper, we discuss our experience with an integrated, rationale-based modeling tool in a variety of software engineering courses. By providing an integrated modeling environment, students use a single tool for requirements, system design, test planning, and collaboration. By attaching rationale to the models, students are encouraged to reflect on their work while instructors canmonitor them and provide more insightful feedback.

1. Introduction

For the past decade, teaching project courses has been a favored approach in software engineering education and empirical research [3,12]. By having students collaborate on the development of a realistic piece of software, the instructor can cover issues related with technical complexity (e.g., performance constraints, changing requirements) as well as social aspects (e.g., conflict resolution). The natural tendency is then to focus on industrial tools for development activities. A positive side effect of this approach is that students are exposed to tools they will likely use in their professional careers.

While learning-by-doing is an important aspect of software engineering education, learning-by-reflecting is also critical [7]. In addition to being exposed to the development of a system and its related work products, students must understand the rationale behind the system in order to improve their skills. Reflecting on their decisions, reviewing others' decisions, considering alternatives, and identifying the relevant criteria are critical to the learning process. In a project course, these activities occur in face-to-face meetings, during informal discussions with the instructors, and in the classroom. However, they occur mostly outside of development tools, which only focus on the end product of this process.

These issues led us to use an integrated, rationale-based modeling environment in our software engineering courses. *Sysiphus*, a suite of tools developed at the Technische Universität München (TUM, [6,13]), enables students to develop and share system models and to discuss, justify, and collaborate using rationale models similar to QOC [9]. The rationale models are then linked to the relevant system model elements, providing context for the discussion. The originality of sysiphus is that it puts equal importance on the system models and the rationale models.

To support software engineering education, we use rationale models in several ways:

- By attaching rationale to an example system model, we can convey to students the reasoning behind the example, providing a basis for further discussion with the students.

- By attaching a partial rationale to a model submitted by a student, we can provide structured feedback to the students, encouraging them to further reflect on their design.
- By attaching rationale relating different system models, we can emphasize the relationships among different software engineering activities.
- By monitoring rationale provided by students, we better understand the challenges they face and can provide tailored guidance by adding options or discussion arguments.

In this paper, we report on our experience in using the rationale-based features of sysiphus for teaching software engineering in seminars and project courses, assessing its value for software engineering education using qualitative observations. Section 2 describes the rhetorical model of rationale we have used. Section 3 describes sysiphus. Section 4 discusses the different uses of rationale for teaching and summarizes our experiences. Section **Error! Reference source not found.** concludes with lessons learned and future directions.

2. Rationale model

Rationale is the justification behind decisions, including alternatives explored, evaluation criteria used to assess alternatives, and argumentations. Rationale is usually not captured and only exchanged informally. While requirements and design decisions are captured in system models, their rationale is gradually lost as participants move to other projects or as their memories fade. The assumption behind rationale management is that externalizing rationale, making it available to project participants, and structuring it for long-term use can increase the quality of decisions, increase trust and consensus among stakeholders, and facilitate changes to existing systems [10]. While the usefulness of rationale management has been shown in specific cases, overhead associated with training participants and formalizing knowledge remain significant obstacles during development [2].

To represent rationale we use an *issue model* as proposed by argumentation-based rationale approaches [2]. Issue models represent the individual decision making elements as individual nodes and their relationships with edges. Many different models have been proposed, including IBIS (Issue Based Information System, [5]) and QOC (Questions, Options, Criteria, [9]), to name the principal ones. We use a refinement of QOC that includes the following:

- *Issues* represent needs to be solved for the development process to proceed. Issues can indicate a design issue, a request for clarification, or a possible defect.
- *Options* are possible solutions that could address the issue 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 nonfunctional requirements, system design goals, or test criteria.
- *Assessments* represent the evaluation of a single option against a criterion. An assessment indicates whether an option satisfies, helps, hurts, or violates a criterion.
- *Arguments* represent the opinions of individual stakeholders, in particular, about the relevance of an issue 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 an issue representing the selected option. Decisions are already implicitly captured in the system models during development. We only need to capture the relationship between decisions and their corresponding rationale.

Issues can be linked to any number of model elements (e.g., use cases, nonfunctional requirements, subsystems, test specifications). Similarly, a single model element can have any number of issues linked to it. Model and rationale elements are viewed side by side, enabling the user to go back and forth between the system model and its justification. Unlike in Design

Space Analysis [9], the justification activity does not drive the modeling activity. Conversely, rationale is not buried in the form of annotations contained in the model elements.

When examining the types of issues posted by students during our first tries [6], we realized that the above model was too general for most cases. To address this problem, we added a category attribute to issues, enabling participants to classify issues into *challenges on form, challenges on content, clarification questions, inconsistency issues, omission issues,* and *justifications*. In addition to making the issue model much more concrete and easier to explain to students, the categorization provided hints about the issue's author expectations from others. For example, clarification questions are simply closed with a decision, while justifications are expected to include several options assessed against all relevant criteria.

3. Sysiphus: An integrated rationale-based environment

The design goals of sysiphus are to provide a simple and integrated solution to manipulate system models and rationale, embedding only minimal process specific knowledge. This enables us to adopt different development processes for different courses and to use rationale for a broad range of activities. Sysiphus includes a suite of tool centered on a repository, which stores all models, rationale, and user information. The repository controls access and concurrency, enabling multiple users to work at the same time on the same models.

Model elements are organized into documents that provide activity specific views into the repository. For example, a requirements analysis document can be defined in terms of sections that include use cases and nonfunctional requirements. Moreover, the same model element can also appear in different documents. For example, actors can appear in both a problem statement and a requirement specification. The number, purpose, and goal of documents can be customized to course needs. A project course starts with a real client providing a problem statement document. The instructor then provides a requirements analysis document template to be filled with model elements by students. In a testing course instructors provide a requirement analysis and system design documents, and students develop a test specification by creating test plan model elements. Moreover, to accommodate different terminology, sysiphus let the instructors use different element labels for different projects. For example, nonfunctional requirements would be displayed as "NFRs" in one course and "Constraints" in another.

Users access models either through a web interface (Figure 1) or a graphical user interface (Figure 2). Both interfaces access the repository through a common model layer that defines the type of elements and links. The model layer can be extended to accommodate new element and link types, as needed by the course. The model layer in turn uses a storage layer for persistency and for pushing changes to other clients. With the web interface end users can access the repository from a variety of environments (e.g., lab, home, office) without the installation of additional software. This significantly reduces the administration overhead associated with large project courses. The graphical user interface provides a UML view over the models, adopting the look and feel of typical UML modeling tools. The graphical user interface is used for drawing diagrams and, with drag and drop interaction styles, offers more intuitive ways to specify relationships among model elements than the web interface.

An administration tool for instructors is integrated into sysiphus, enabling instructors to manage student accounts, projects and access rights within the scope of a course. Based on a reference project, the instructor creates identical copies for each group. Each student group can only access and modify its project, while instructors and tutors can monitor all projects. At the end, the student contributions can be compared to the reference project.

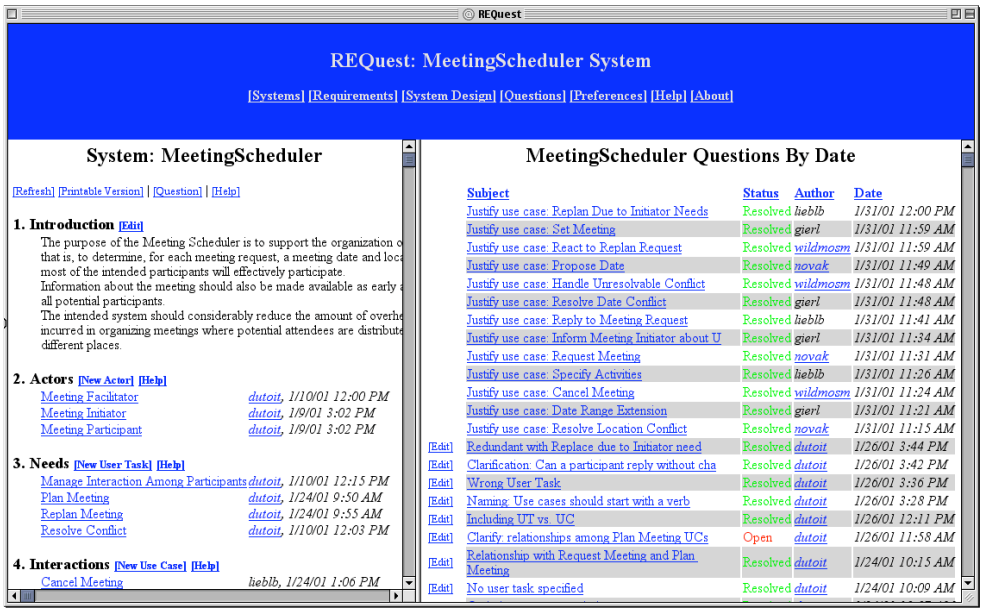

Figure 1 Web interface: system models (left column) and rationale (right column) are allocated the same amount of screen real estate.

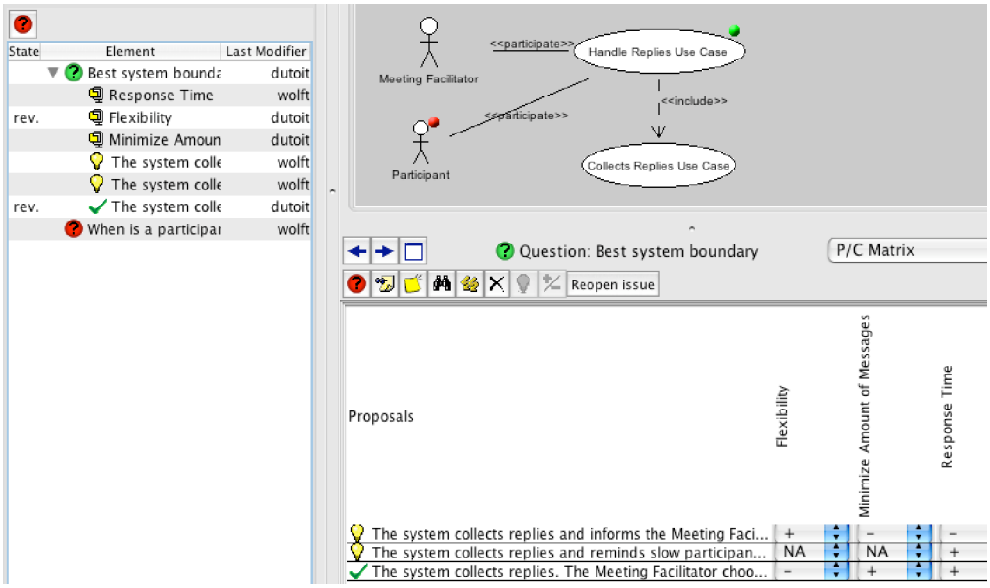

Figure 2 Graphical user interface: Diagrams (top right), issues (left) and a content pane (bottom right) provide alternate ways to access and create links between system elements and rationale.

Sysiphus differs significantly from typical modeling and rationale tools in two ways:

- Rationale elements are first class objects (as opposed to buried notes or comments) and are accessed the same way as system model elements. The tool puts equal focus on system and rationale. The end user can browse back and forth between rationale and system models.
- Changes made by the end user are propagated synchronously to other end users working on the same model, enabling users to collaborate synchronously. When overlaps are discovered, the end user is prompted by the system to merge to conflicting changes.

4. Supporting teaching activities with rationale

In software engineering education, we are interested in using rationale to enhance the communication between instructor and students. Our assumption is that making rationale explicit encourages students to reflect on their work while enabling the instructor to better monitor the progress of students. Instructors can then provide guidance, thereby improving the students' skills. In this section, we describe how rationale can be used for improving system model understanding, for supporting ad hoc collaboration, and for guiding students based on their reflections. We then summarize our experiences at the end of this section.

4.1 Improving system model understanding

In ill-structured domains where learning-by-doing and learning-by-reflecting is critical (e.g., human computer interaction), issue models have been used to describe cases to students and provide a formal framework for reasoning about them [4]. Typically, the rationale for an existing design is reconstructed for the course and structured as an issue model. The criteria and alternatives the instructor wants to emphasize are included in the model.

For example, Table 1 depicts an example justification of a use case for a meeting scheduler system. The issue is about the optimal system boundary. Three options are sketched and evaluated against the criteria, which are in this case nonfunctional requirements that are also part of the specification. 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 Meeting Facilitators, because there is no way they can extend the time for participants to reply before closing the question. The chosen option is marked by bold. If the criteria are of different priority, the option with the highest score of "+" need not be the optimal one.

Table 1 Example justification provided to students. In this case, the issue is associated with a use case and the criteria are nonfunctional requirements relevant to the use case.

Justification	What is the best Option for the system boundary within in the "Handle Replies Use Case" satisfying the non-functional requirements?		
Reference	Handle Replies:Use Case		
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.	–	+	+

4.2 Supporting ad hoc collaboration

Initial efforts in using rationale models in software engineering focused on supporting collaboration, including communication, negotiation, and conflict identification and resolution (e.g., [1,5]). In these approaches, participants collaborate within an issue model. During negotiation, stakeholders declare their win conditions as criteria and assess options proposed by other stakeholders to explain how succeed or fail to satisfy their win conditions. A simpler case of collaboration is the request for a clarification, possibly resulting in raising a new design issue after the misunderstanding is resolved (see Table 2).

In education, the benefit of using rationale for ad hoc collaboration is that students are challenged to separate questions and criteria from options, arguments, and assessments, making it easier for instructors or other students to respond accurately. The drawback of this approach is that students must put more effort during collaboration, often resulting in fewer questions being asked.

Table 2 Example of a simple clarification question (contrast with Table 1).

Request for clarification	Because Players will be the initiators of games, should not Mission Designers - the creators of the game - be derived from the Player class?
Reference	Mission Designer:Actor
Decision	A Mission Designer doesn't actually play a game. He just draws up storylines, items and rules. The task of the Player is to actually walk around, hit other Players on the head, collect stuff from the world, and so on. These two activities have virtually nothing in common. So neither of them should inherit from the other.

4.3 Guiding students based on their reflections

By combining the above two approaches, instructors can use a rationale model to elicit information from students and provide guidance in a structured way. For example, the instructor can present an incomplete rationale for an architectural decision, only listing alternatives and asking students to identify the design goals and the corresponding assessments that led to the selected architectural option (Table 3). Alternatively, the instructor can ask students to update an existing rationale as a result of a change of criteria (e.g., revised design goal, new nonfunctional requirement). By shifting the focus from the system model to the rationale model, the instructor challenges students to reflect on their work. This use of rationale is specific to education and, to our knowledge, has not been systematically researched.

Table 3 Example partial rationale to be filled by students (simplified). Students were asked to fill their assessments and select one out of six option.

Justification	What is the best architecture for Sysiphus?			
Criteria:	Coexistence of different clients	Coexistence of different client versions	Fined grained access control	Portability across operating systems
Option 1: Three layer architecture with storage layer on the server node, user interface and model layer on the client nodes.				
Option 2: Three layer architecture with user interface layer on client, model and storage layer on server.				

4.4 Case studies

We have been using different versions of sysiphus in software engineering courses since 2000. For this discussion, we have selected three courses illustrating different aspects of rationale in software engineering education.

Requirements seminar. We conducted in the context of a requirements engineering seminar a six-hour exercise, replicated in TUM and University of Kaiserslautern, to expose students to the difficulties of changing a requirements specification. During a previous semester, another group of students had developed a use case specification of the benchmark meeting scheduler problem [8]. The instructors defined a change task involving the addition of a nonfunctional requirement, requiring students to adjust existing requirements. The instructors also added justification issues for the use cases involved in the change task. The students worked during three two hour sessions: the goal of the first session was to understand the existing specification, the second session to assess the change impact, and the third session to carry out the change in the specification, including updates to the rationale.

SE project course at TUM. The goal of this course is to provide undergraduate senior level students a realistic experience in software engineering, including complexity related to both technical and social issues [3]. Students are organized into a single project addressing a problem defined by an actual client. During the winter semester 2003/04, students developed a prototype forecasting system for the Wacker chemical company. Sysiphus was used for developing both the requirements analysis and the system design documents. The requirements analysis document turned out to be one of the primary client deliverable for this project for the Wacker IT department for realizing a production version of the system. The rationale part of sysiphus was used for ad hoc collaboration (clarification questions, discussion of design issues) and for tracking open changes. Change requests were indicated as challenges on content questions that were then closed by the participant who implemented the change. This made it possible by instructors to monitor and address potential problem areas.

SE project course at University of Heidelberg. In this course students first reversed engineered an existing system and then extended it with new functionality (forward engineering) [11]. All documents (requirements, system design, and test) were managed in sysiphus. During the reverse engineering students learned basic software engineering techniques such as modeling or testing techniques. The students received each week a partial version of the documents that they had to complete. During this phase the administration feature of sysiphus simplified the management of document versions. With respect to rationale, both improved system understanding and guidance was essential. The rationale was used to pinpoint the students to the most essential decisions behind the existing system. This was supported by the tracing links between model and rationale elements as the students needed to understand, for example, how different classes were related. Most important was the rationale for guidance. On the one hand students were provided partial rationale and asked to consolidate it by uncovering pros and cons of the existing solution (see Section 4.3). On the other hand the tutor could provide detailed questions on the task in the rationale. This helped to guide the students during task execution. During forward engineering the students designed and documented an extension to the system. The rationale supported communication between tutor and student. Students were requested to justify their most important design decisions allowing the tutor to comment asynchronously early. This early and structured feedback helped to avoid rework by students. The latter is important in this course as the time constraints made managing student workload critical.

5. Lessons learned and future directions

We gathered qualitative observations by surveying students and reviewing the rationale contributed by students. Students were told and knew from previous semesters that their feedback would not affect their grade. However, feedback was not anonymous as we wanted to relate answers to tool usage. The table below summarizes the essential points.

	Uses	**Feedback**
RE seminar	Increasing understanding. Eliciting justifications.	Rationale useful for understanding, however, introduced more work during change task (rationale often not revised correctly).
TUM SE course	Ad hoc collaboration Informal guidance Change tracking	Ad hoc communication increased awareness of others within the project. No spontaneous consolidation of rationale.
Heidelberg SE course	Increasing understanding Structured guidance	Structured feedback critical for managing student workload.

We observed that rationale is an effective method for structuring feedback and encouraging students to reflect on their solutions. A rationale model is a compact and readable representation that can serve as a basis for detailed design discussions. Creating rationale, however, is not intuitive. Consistent with other work in rationale, we found that students need an in-depth tutorial and several exercises before they are able to write their own rationale. We also observed that rationale models tend to be overloaded with foreign items such as reminders, to do lists, or status items. We have extended sysiphus to provide explicit support for such information and assess its usefulness during the next semester.

References

1. 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.
2. 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.
3. B. Bruegge, R.F. Coyne, A.H. Dutoit, D. Rotenberger. "Teaching More Comprehensive Model-Based Software Engineering: Experience With Objectory's Use Case Approach" CSEE'95, New Orleans, 1995.
4. T. Carey, D McKerlie, J. Wilson. "HCI Design Rationales as a Learning Resource." In [10]
5. J. Conklin & K. C. Burgess-Yakemovic, "A process-oriented approach to design rationale," Human-Computer Interaction, vol. 6, pp. 357–391, 1991.
6. A.H. Dutoit, B. Paech. "Rationale-based Use Case Specification" Requirements Engineering Journal, vol. 7, pp. 3–19, Springer Verlag, Mar 2002.
7. O. Hazzan, J. E. Tomayko. Reflection Processes in the Teaching and Learning of Human Aspects of Software Engineering. CSEET'04, Norfolk, VA, Apr 2004.
8. 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.
9. 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.
10. T.P. Moran, J.M. Carroll (Eds.) Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum, Mahwah, NJ, 1996.
11. B. Paech, L. Borner, J. Rückert, A.H. Dutoit, T. Wolf. "Vom Kode zu den Anforderungen und zurück: Software Engineering in 6 Semesterwochenstunden" Submitted to Software Engineering im Unterricht der Hochschulen. Aachen, 2005.
12. D. Port, D. Klappholz. "Empirical Research in the Software Engineering Classroom" CSEE&T'04.
13. T. Wolf, A.H. Dutoit "A rationale-based analysis tool." 13th International Conference on Intelligent & Adaptive Systems and Software Engineering, July 1-3, 2004, Nice, France