Studies into interactive didactic approaches for learning software design using UML

Stikkolorum, D.R.

Chapter 7

Teaching of Agile UML Modelling: Recommendations from Students' Reflections

Our research aims to develop a teaching method that solves or addresses the main difficulties for students to integrate modelling in their common practices in software development. In particular, the use of modelling has been challenged by adoption of agile processes. In this chapter we propose an educational approach that reveals several practices for teaching modelling based on evidence distilled from students' peer-reflections. Our approach, that was inspired by pair programming, is used as a means to: i) reveal what typical difficulties students face during software modelling, ii) help students create better (UML) models, iii) enable students to better understand the difference between domain- and design models, and iv) integrate modelling in an agile development process. In this study we asked student pairs to reflect on each other frequently during an UML analysis and design assignment. We qualitatively analysed these reflections. We observed that this approach triggers reflective thinking in our student modellers. Based on the distilled practices, we discuss pitfalls and recommendations for lecturers that are teaching UML analysis and design.

This chapter is based on the following publication: Dave R. Stikkolorum and Michel R.V. Chaudron. Teaching of agile uml modelling: Recommendations from students' reflections. In *Proceedings of the 20th Ibero-American Conference on Software Engineering, Buenos Aires, Argentina*, 2017

7.1 Introduction

Students in software engineering programs are confronted with the challenges of software modelling. During analysis and design learning activities students translate problems into abstractions and abstractions into suitable software solutions. In general analysis- (problem domain) and design models (solution) can be expressed with various diagrams of the Unified Modelling Language (UML) to describe structure and behaviour. Although object-orientation, with UML, is widely adopted at the universities, modelling is often stated to be very hard for novices [79, 69, 122].

Lecturers are challenged to find educational approaches to motivate students. They offer different perspectives on subjects in order to enable or improve students understanding of complex matters, such as identifying of concepts (abstraction), assigning responsibilities or filter out unimportant information. In order to achieve this, lecturers try to understand the challenges students face and base their educational approach on common learning patterns derived from students' behaviour.

One of the challenges is to have a clear understanding of what students find difficult and what is typically helping the students through their learning process, such as feedback and examples from the lecturer. Besides learning a new language syntax, the biggest challenge is learning how to apply abstract reasoning: forming an abstract model of a given problem or suggest a structured solution that should lead to a decent software implementation. Abstract reasoning abilities have often been discussed in the software modelling community [69, 141, 90, 50].

It is difficult to exactly reveal what goes on in the mind of the novice modellers. Lecturers are able to reveal some of the students' struggles in practical lab settings. In these situations they can observe and discuss their challenges. We also are able to see global strategies (depth first - breadth first) as discussed in previous research [137]. Recognising detailed issues that lead to misunderstanding of the learning subject is not clear yet.

We believe that collaborative learning forms have a positive influence on the learning processes of students. These active learning forms also enable us to scale up classrooms, in order to serve a larger group of students. Lecturers report the use of pair programming in their courses [72, 89]. This collaborative form is often integrated in agile development methods. Software modelling does not have to be an individual activity. We have positive experiences with letting our students exercise modelling in pairs. Pair modelling enables discussions around the students' modelling activities. Reflection on experiences is an important part of today's dominant constructivism learning theory [13]. In our experience students feel comfortable with agile methods,

and we believe modelling should be integrated in agile development processes. In a previous study [131] we explored the integration of UML modelling in an iterative development process.

In this chapter we explore an educational approach in which we focus on the interaction between student pairs in order to investigate their learning process. This chapter addresses the following research questions:

> **RQ1**: Does peer-reflection help students to express their difficulties in modelling software designs?
>
> **RQ2**: Which difficulties do students express through peer-reflection during software modelling?

This chapter describes a modelling exercise in which students are encouraged to perform domain analysis and software design in an agile way. Students were asked to perform short time constrained iterations on an analysis class diagram and a design class diagram. During the execution of the exercise students asked each other about their difficulties. With the use of an on-line form we recorded these peer-reflections in order to find typical decision making problems during software modelling assignments. Our active work form should enable lecturers to reveal students' modelling problems.

By answering the research questions we contribute to the improvement of software modelling education. In this chapter we discuss several points of concerns lecturers should take into account when developing their courses. Each point of concern is discussed and concluded with a recommendation.

The remainder of this chapter is organised as follows: Section 7.2 discusses related work. We show the educational method, research method and tools we used in Sections 7.3 and 7.4. Our results are presented and discussed in Sections 7.5 and 7.6. Validity threats are discussed in Section 7.7. We conclude and propose future work in Section 7.8.

7.2 Related Work

In our research we analyse students' design reasoning and strategies in order to uncover common mistakes. In a previous experiment [137] we asked student pairs (N=98) to record their questions during a modelling assignment. We were able to identify some common difficulties. By using peer-reflection we aim to gain a higher response than the previous approach (24%).

Leung [79] and Bolloju [19] discuss common errors in relation to the quality of models from novice analysts. They address that students' models often have similar shortcomings and proposed a categorisation of common errors that could be used for educational check-lists. They do not address software design activities or students' reasoning.

Several authors discuss the benefits of pair programming in education. Lai et al. conclude that pair programming is beneficial for improving students' confidence and the quality of the tasks they have to perform [72]. Mcchesney [89] found pair programming improves students' performance and that students have a positive experience with pair programming. From a professional perspective pair programming is expected to support creativity and the problem solving capability of the software engineer [8].

In comparison to pair programming we believe modelling in pairs could yield similar benefits. Ambler [5] and Rumpe [116] suggest to integrate modelling into agile development. We integrated UML modelling into an agile software development process in a workshop for software engineering students [131]. Zhang et al. [162] mention the use of paired modelling in Agile practices in industry. In this chapter we focus on iterations of analysis and design models with use of modelling in pairs.

In order to make conceptual modelling more attractive to students, Pastor et al. let students experience traditional development versus a model-driven approach (MDD) [100]. Their results show that MDD benefits students for complex problems. Based on classroom discussions they conclude that, for students, the quick code generation was the major pro, whilst the learning curve of MDD was the major con.

Razavian et al. [111] conducted multiple case studies in which a design thinking approach was used to show that active reflection improves design reasoning. The authors argue that, next to problem solving skills, software designs have a need for a reflective mind to challenge the design decisions that were made.

Zhuoyi et al. emphasise that learning requires collaboration and communication. They report their exploration of a constructivism based teaching model [163] in which students actively gain self-experience by the means of a real project.

Tang [148] discusses the bias software designers have that can lead to bad design decisions. He discusses reasoning and reflecting on design decisions. He argues systematic reasoning about designs could yield a better design quality. He concludes that 'we need to conduct empirical studies and experiments to understand how to make better software design decisions'.

With our approach, peer-reflection of students during a pair modelling task, we aim to reveal practices of reasoning, rather than software modelling errors.

7.3 Teaching Method

In this section we describe the teaching method that was used for our study. Figure 7.1 illustrates the assignment activities the students performed.

7.3.1 Modelling in Pairs

Our approach is inspired by pair programming. We believe students learn better by discussing their analysis and/or design models. The discussion enables the possibility to reflect on each other's reasoning, which supports the students' learning. We also believe that similar benefits from pair programming can be obtained with pair modelling, such as enjoyment of modelling and confidence in modelling.

7.3.2 Assignment Task, Assignment Procedure and Tooling

On paper the students received a case description about an exam system for universities. The assignment text was open for interpretation. Students were encouraged to ask questions to the lecturers. There were two tasks: i) analysis, focused on creating a model of the problem domain and ii) design, focused on creating an initial software design, taking into account non-functional requirements that were introduced in the text. The models were expressed with UML class diagrams.

After an introduction of the tool the students first performed 3 iterations on the analysis model and subsequently iterated 3 times on the design model, as seen in Figure 7.1 (rows 2 and 3). Between the iterations the student pairs discussed with their peers about the challenges of the assignment and status of the diagram. The procedure, as seen in Figure 7.1, was repeated for the design task. The assignment took 2 lecture hours of 45 minutes, including a short break of 5 minutes.

For UML modelling we used the on-line editor WebUML[1] that is developed by the authors. The web-based editor does not require students to install software and enables students to send the result digitally in a standard format (XMI). The students brought their own laptop to work on.

[1] https://webuml.drstikko.nl

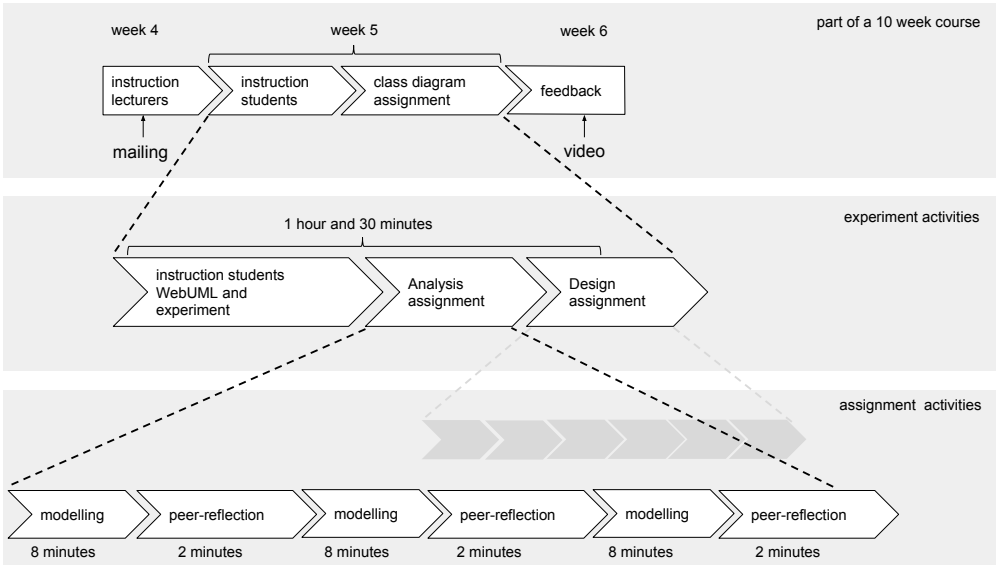

Figure 7.1: *overview of the teaching method*

7.4 Research Method

This section describes the research method we used. We discuss the participants, the data collection method and how we analysed the data.

7.4.1 Participants

Students 78 students participated in this study. They followed the 2nd semester of their first year bachelor's program in ICT at The Hague University of Applied Sciences (The Netherlands). The students were divided over 5 groups and were supervised by a lecturer. The students were paired with a partner of their choice. The study took place during an object-oriented modelling course, with the main focus on UML diagrams. Prior to the object-oriented modelling course the students learned about the basics of Java and databases.

Lecturers 5 experienced lecturers participated in the experiment. The assignment text was prepared in collaboration with the coordinator of the object-oriented modelling course and was discussed with the 5 involved lecturers. Each student group was supported by 2 lecturers. They were available during the assignment for answering the students' questions. The lecturers were selected from the same course as the students

were registered for.

7.4.2 Data Collection and Data Analysis

The peer-reflection records were collected with Google Forms[2]. Per iteration each student filled in 2 free form text boxes with guiding questions. One addressing what the questions were that raised during the activity and what they noticed about their own approach (*Which questions arise? What was notable in your approach?*) and the other one about what steps they found difficult, and, if there was a typical suggestion that had helped them to solve a particular problem (*What were difficult steps? What gave you insight?*).

All data[3] from the Google forms were combined in a text file. We grouped and labelled the different remarks, questions and observations the students wrote down. We identified frequent occurrences of similar cases. Finally we distinguished between analysis, design and general modelling activities.

7.5 Results and Discussion - Points of Concern for Education

In this section we present and discuss the results of the recorded student peer-reflections. Based on the records of the peer-reflections we identified points of concern for education of software analysis and design (shown in Table 7.1). We present three types: i) general concerns, identified in both analysis and design modelling, ii) concerns distilled from the domain modelling task and iii) concerns identified during design modelling. First we explain the format we use to present our points of concerns, subsequently we discuss them per type.

In this chapter we recommend practices based on the discussion and evidence we found in the peer-reflections. In the following subsections, a point of concern is divided in the following sections: i) **problem** - an explanation of the problem we identified, ii) **evidence** - phrases of the recorded text that show the problem is present in the observed student group, iii) **discussion** - discusses the problem by interpretation of the collected text, and iv) **recommendation** - recommends practices for lecturers in software analysis and design modelling.

[2] https://forms.google.com

[3] The data can be provided on request.

Points of Concern for Education of Software Analysis and Design	
General Modelling Points of Concern	
1	Prior knowledge about implementation can be disadvantageous.
2	Students have difficulties with open-ended modelling assignments.
3	Students solve the wrong problems at the wrong time.
4	Students need confirmation during their assignment.
5	Lecturers should be aware of the influence of modelling tools.
Domain Modelling Points of Concern	
6	Students have difficulties with the analysis of text based assignments.
7	Students have difficulties in differentiating between analysis and design.
Design Modelling Points of Concern	
8	Students have difficulties with determining a design's level of detail.
9	Students have difficulties determining the model is finished.
10	Transitioning from analysis to design can be hard.

Table 7.1: *Overview of points of concern.*

7.5.1 General Modelling Points of Concern

Point of concern 1: Prior knowledge about implementation can be disadvantageous.

Problem - Students often followed programming and database courses prior to their (UML) modelling course. In analysis modelling, the software implementation details do not play a role. Still students find it difficult to postpone implementation thoughts. Details such as id's and types are found in models of these students.

Evidence - In the peer-reflections we found remarks that relate to prior database and programming knowledge:

- *"Class names should be singular, we did plural because of database experience"* (Some database standards follow the plural form.)
- *"The lecturer commented that we maybe were too much focused on programming. That made me reconsider the relationships our analysis model"*
- *"It was difficult to convert operations that we found to getters and setters"*

Discussion - Often text books use database-type problems as examples. There are numerous examples of cooking applications, flight scheduling systems etc. These

examples are information-oriented instead of object-oriented. Also in the case of this study the assignment was more information oriented. Students used the prior database knowledge to solve the problem.

The fact that students use implementation detail in this stage could be explained by the fact students often directly focus on a solution instead of analysing the problem first. Already having some programming knowledge and experience maybe makes them think they already 'see' the solution.

Recommendation - Lecturers should focus more on object-orientation in their assignments by choosing other examples and assignment texts. They should target system concepts that relate to behaviour instead of information. Lecturers should explicitly focus on the difference between analysis and design in order to avoid implementation details in domain models.

Point of concern 2: Students have difficulties with the open-endedness of a modelling assignment.

Problem - Often the freedom of the assignments leads to confused students. Not only do they have to interpret textual information, but also the amount of possible solutions is large.

Evidence - The following records show related confusions:

- *"To what extent made up attributes may be added?"*
- *"I need to add information to classes that otherwise seem empty. Normally I can speak to a client for this"*
- *"It is difficult to make the right choices when missing information"*

Discussion - Although students were informed that they could ask questions, a large group did not ask questions to check their assumptions on the case. This could be explained to shyness. Students do not want to ask 'silly' questions or are afraid of asking about a topic that they should know about. It also occurs that lecturers are busy explaining a matter to another student while a question arises.

Students should be trained to handle cases where information is missing. One of the key skills in software development is to reveal users' needs and software requirements. Courses should have assignments that simulate these situations, but students are not equipped to handle them.

Recommendation - Lecturers should have an active approach and ask students for their understanding. To overcome support issues in terms of time, lecturers could

consider hiring teaching assistants. If instructed properly, teaching assistants motivate students to ask questions. Approaching a teaching assistant could be less a hurdle for a student than approaching a lecturer.

Point of concern 3: Students solve the wrong problems at the wrong time.

Problem - The use of the same diagrams in UML for both analysis and design tasks has its disadvantages. Students think of software design problems when they are performing analysis steps and vice versa. Students tend to directly go for a design without doing a proper analysis first.

Evidence - In this experiment we found several occurrences of this:

- *"The first question we had, was if we did not have to much coupling"* (Recorded during analysis. Coupling is a software design problem.)
- *"Analysis model is already based on design"*
- *"We chose to directly go to design. In that way we skip an unnecessary step"*

Discussion - Students have no experience in software development. Therefore they do not have a clear view on the software development process. We think it is hard for them to link theoretical subjects to a particular phase in the development cycle. As seen above they cannot see the value of distinguishing between analysis and design.

Recommendation - Lecturers should find ways to show the relationship between modelling and the different development phases. One way is to use our proposed workshop from previous research [131]. In that workshop we integrate modelling in an agile development approach.

Point of concern 4: Students need confirmation during their assignment.

Problem - Design is an iterative process, hence students want to make sure they are down the right path. Students feel insecure about their designs. From their remarks we learned they are insecure about what the end result should be like.

Evidence - in the peer-reflections we found that students ask for confirmation quite often:

- *"Are we heading in the right direction?"*
- *"Was our analysis correct, did we miss something?"*

- *"We changed some parts of our diagram after comments from our fellow students"* (confirmation by students in stead of the lecturer).

Discussion - For the learning of new topics it is of course a natural phenomenon to be insecure. Receiving confirmation is crucial for learning. Students need to build up their confidence. Leaving out confirmation feedback can lead to wrong decisions of the student.

Recommendation - It is not always possible to give feedback at the right time. For example, in a practical lecture setting lecturers have to deal with a large group of students and a limited amount of time. We believe exercising in pairs, comparable with pair-programming [89, 72], can help to build up students' confidence.

Point of concern 5: Lecturers should be aware of the influence of (problems with) modelling tools.

Problem - The use of tools has been discussed actively in the modelling community [2]. Every lecturer has different reasons for the use of their tool of choice. Some lecturers use more informal drawing tools, while others choose to use formal tools. We chose to use our on-line tool WebUML because it's capability of logging, the small subset of UML class diagrams it uses and it does not require installation. The downside is it is still in development and has bugs.

Evidence - Although explained in advance, a large part of the students that participated struggled with bugs. Other students had a more positive experience, and some came with ideas. Below a couple of examples from the records:

- *"The working-canvas is too small for the case, therefore visually it is too much and this makes it more difficult too reason."*
- *"Hitting delete during typing deleted the full class."*
- *"This tool is better than Visio."*
- *"It would be nice if the tool could have a UML cheatsheet."*

Discussion - Modelling should be useful. Non motivated students see it as an unnecessary step. A bad tool experience leads to the avoidance of modelling. Most industrial tools are quite complex. Students do not need all the 'expert' features of a tool. We believe a small subset of the UML in combination with an easy to use tool should be enough.

Recommendation - Lecturers should invest time to let students get used to their tool of choice. This can be achieved with small instruction assignments and for example screen cast guides as suggested by Liebel et al. [83]. The tool itself should be simple. Ideally a tool should support the learning process.

7.5.2 Domain Modelling Points of Concern

Point of concern 6: Students have difficulties with the analysis of text based assignments.

Problem - Study material such as books, readers and exams in most cases provide the student case texts that have to be analysed. Lecturers often introduce exercises that provide incomplete information to train analysis skills. Students seem to struggle with the lack of information in case texts and consider this to be difficult.

Evidence - Some student remarks about facing the text:

- *"The case description is somewhat small. It does not provide some specific information"*
- *"The case description is somewhat small. Because it is not always possible to ask about the text, it is difficult"*
- *"A lot of matters are not provided by the text, so we took the initiative to add them ourselves"*

Discussion - Students get confused and frustrated about the fact they cannot see 'the big picture' because of the lack of information. Own initiative is of course what we want from students, but how early in learning modelling this should be done? In the end we maybe should argue if providing case texts is really a reflection of the real life. How often does a client have a piece of text that explains the system it wants to have?

Recommendation - Lecturers could overcome these problems by preparing several scenarios based on previous student remarks or questions. Lecturers should actively motivate students to ask questions to clear up their understanding of the texts. Their should be enough time and/or enough lecturers to support the group. When a lecturer does not supply specific information in the assignment(s) he/she creates a responsibility to verify the students assumptions. When using text assignments it is of course inevitable that not all assumption are explicitly described.

Point of concern 7: Students have difficulties in understanding the difference between analysis and design.

Problem - Although emphasised more than once by lecturers during software analysis courses students cannot resist the temptation to think about implementation of software instead of the analysis of the problem. Therefore they have difficulties to distinguish analysis models from design models and have the idea that design is the only purpose of modelling.

Evidence - Some of students' records during the analysis task:

- *"We have to stay focused on analysis"*
- *"We made the design step by step"* (while making a domain model)
- *"What exactly belongs to an analysis diagram and what to a design diagram?"*

Discussion - Students seem not to be aware of the different concepts. They are not aware of the fact that one model reflects the problem domain and therefore real-life concepts, whilst the other reflects computer concepts, the software solution.

Recommendation - A suggestion for lecturers is to explicitly address the difference and purpose of analysis and design modelling. They should address what to expect of the model students produce during an exercise.

7.5.3 Design Modelling Points of Concern

Point of concern 8: Students have difficulties with determining how detailed a design model should be.

Problem - A larger group of students has difficulty in deciding whether a design is detailed enough or not. Adding details could involve adding elements, such as types, visibility, multiplicity, or relationships names.

Evidence - Consider the following student remarks from the records:

- *"What is excessive in design diagram?"*
- *"How much notation is too much?"*

Discussion - There is a link with the domain modelling phase. Because the assignment text often does not contain all the detail the student should add details based questions to the client (in this case the lecturer). In the design phase also the details of technical

decisions are added such as used types, private or public visibility or for example getter and setter operations. How detailed should a design stay? Or, how close should the design be to implementation?

Recommendation - Lecturers could provide guidelines for the details that they expect. They should differentiate between the phases a design is in and explicitly focus on refinement. The further in time a design develops the more detail is added. For example: an initial design should maybe have data types and relationship names, but multiplicity is not included in the diagram at this stage.

Point of concern 9: Students have difficulties determining if the model is finished.

Problem - In addition to the previous point of concern: for students it is difficult to decide whether their completed their design task or not. The open character of most design assignments plays a big role in this case. Ideally software designs should be clear enough to be implemented by a programmer. Students lack of concrete experience to deliver these clear designs.

Evidence - Students struggle with questions like:

- *"Did we complete the relationships at this point?"*
- *"What else needs to be added?"*

Discussion - A model is finished when it reflects its purpose. The purpose of a (class) design is to have an implementable structure of the future system. Still in practice developers have different ideas about what to accept as a finished model. Some see benefits in a more raw design, others in a more complete one. Educators overestimate students by assuming they can decide a (design) model is finished. They are inexperienced.

Recommendation - Students should discuss each others' models. Lecturers can discuss students solutions in the classroom (for example with the use of a beamer). During a discussion students could be guided by a lecturer or teaching assistant, supporting the students with their experience.

Lecturers could also provide follow-up assignments in which students have to implement their models. In this way students discover what they missed during the design and build up some experience.

Point of concern 10: Transitioning from analysis to design can be hard.

Problem - As mentioned before analysis and design have different point of views. Students have to switch from a problem perspective to a solution/software perspective.

Evidence - transitioning to design can be surprisingly hard for students:

- *"It is quite hard to go from analysis to design"*
- *"It is harder to make a design class diagram in comparison with an analysis diagram"*

Discussion - The difficulty of making the transition from analysis to design can be explained by the semantic distance between both phases [122]. Saying design is harder than the analysis is probably is an underestimation of the students. Although, in comparison to design models, domain models are less rich of UML features and therefore maybe less complex, the real complexity lies in the discovering of the key concepts from the problem domain.

Recommendation - We recommend to use development phase related assignments. The assignments should follow up on each other in order for the students to get 'the big picture'. In this way students get used to the different purposes each model has.

7.6 Additional Discussion - Implications for Education

In addition to the previous section we discuss what implications our findings should have for educating software analysis and design. We categorised the points of concerns and aim to present a more general view. Subsequently we suggest an interactive learning approach to enable students reasoning abilities.

7.6.1 Concern Categories

After analysing and discussing the collected points of concern in Section 7.5, we categorise the concerns into the following four categories:

- **Prior knowledge concerns** - these concerns relate to the prior knowledge students have. In this research we identified the influence of prior programming experience and modelling experience in the form of database modelling. We noticed students tend to skip certain steps or make wrong decisions based on their prior knowledge. Examples of these points of concern are 1 and 3.

- **Transition concerns**- points of concern 3, 4, 7 and 10 all reveal students difficulties with transitioning from analysis to design. Students perform activities at wrong moments. They apply a solution view when a problem view is required and vice versa. Students often don't map the purpose of a specific phase to the total of the development process.

- **Teaching method concerns** - these concerns relate to the methods or tools chosen by the lecturers. They often choose to use open ended assignments. Next to that, students indicate to have difficulties with the analysis of text based assignments. Points of concern 2, 4, 5 and 6 are examples of those concerns.

- **Feedback concerns** - students mention the need of feedback. Feedback is a prerequisite for learning. Out of the student records' analysis, confirmation is the feedback most students need. We observed students also appreciate peer feedback. Points of concern 3, 8 and 9 discuss feedback concerns.

7.6.2 The Value of an (Inter)active Learning Approach

We see software development as an iterative process that has modelling embedded. Iterative development is a process of rethinking and re-constructing. Previous decisions are reconsidered and intermediate products are re-shaped. This requires the training of - and using students' reasoning skills.

In a modelling course modelling should be presented as natural part of the development process. This should not be limited to the lecturer briefly mentioning the topic. Students should be actively engaged in practical modelling exercises. Thinking and especially **re-thinking** is activated by letting peers have discussions about their problems and solutions.

7.7 Threats to Validity

In this section we discuss the threats to validity of our research.
Construct Validity In our study we are aware of the following construct validity threats: i) recorded questions. This study analyses records of students that they have written down themselves. We are aware of the fact that the students can skip important information. The same records could also cause interpretation validity when students write down sentences that are incomplete and thus not clear to understand. In this research we avoided misleading and unclear sentences as much as possible. ii) Realistic assignment. The degree of reality of the assignment should be the one of the student

group that was observed. The assignment was prepared and discussed with the lecturers beforehand. The assignment included reflection moments, but the actual task (modelling class diagrams) did not differ from the other task that were given in the course.

External Validity Many matters could influence the generalisability of our study, such as cultural background, prior knowledge of students or educational methods used by universities. Although we expect to see similar results in other European countries, further studies should be conducted to confirm this.

Internal Validity We chose to use a very basic UML class diagram tool. We explained the tool briefly in advance of the assignment. Although the students commented on the usability of the tool, most identified modelling problems do not seem related to the tool. We believe that larger, more complex tools are more likely to introduce extra difficulties into the task.

Conclusion Validity We base our conclusions about the student difficulties on the recorded peer-reflections. There is a risk that remarks of students are incomplete. We base our conclusions about the educational approach with a limited amount of educators and the peer-reflections.

7.8 Conclusion and Future Work

In this chapter we proposed an educational approach in which we used peer-reflection during an analysis and design assignment. We used this approach to: i) explore if this helps students to express their difficulties in software modelling (RQ1) and ii) distil which common difficulties students have (RQ2).

We analysed the peer-reflections of 78 ICT students during an analysis and design assignment. We identified common problems and difficulties and presented 10 concrete recommendations for teaching software modelling.

From the recorded peer-reflections we conclude that peer-reflection helps students to express their difficulties in modelling software designs. The students actively discussed the difficulties they had as well as the progress and quality aspects of their models.

The peer-reflections revealed a collection of challenges the students identified. We categorised and presented our findings as point of concerns lecturers should take into account when teaching software analysis and design.

With the presented points of concern we expand the knowledge about common diffi-

culties in students' learning of software modelling. With the expanded knowledge we aim to improve our future educational approaches. We recommend other lecturers to use our approach in order to extend our insights.