Studies into Interactive Didactic Approaches for Learning Software Design Using UML

Dave R. Stikkolorum

December 2022

Studies into Interactive Didactic Approaches for Learning Software Design Using UML

Proefschrift

ter verkrijging van

de graad van doctor aan de Universiteit Leiden,

op gezag van rector magnificus prof.dr.ir. H. Bijl,

volgens besluit van het college voor promoties

te verdedigen op woensdag 14 december 2022

klokke 15:00 uur

door

Dave Roy Stikkolorum

geboren te Den Haag, Nederland
in 1976

Contents

Part I

Introduction, Context and Problem Statement

Chapter 1

Introduction – Challenges in Software Design Education

In this chapter we introduce the research context, our main objective and research questions. Further we present our research methods. In addition, we provide an overview of the structure of this dissertation. Furthermore this chapter concludes with an overview of the published work that this dissertation is based on as well as work that was published during the research period of this dissertation.

1.1 Problem Context – Education in Software Design

Software Engineering (SE) is a rapidly changing discipline. Software developers have to respond and adapt to constantly changing hardware and implementation platforms in a society with increasing challenges [94]. Therefore, software developers are challenged to create increasingly complex systems. The effective development and maintainability of such complex systems requires well thought out designs.

Software designs are the 'blueprints' for the system that is going to be delivered. The designs lay the foundation for later implementation steps. Software designs are expressed with software models. They are the means for communicating designs. In software engineering it is common to use visual representations for models. A collection of diagrams that focus on different views of a system (e.g. structure and behaviour) represent the system's model.

After its introduction in the 1990's the Unified Modelling Language[1] (UML) became the de facto standard for the modelling of software designs, and most universities started introducing UML in the classroom at the end of the 1990's. UML integrated the notations of the Booch method [20], object-oriented modelling technique (Rumbaugh et al.) [115] and object-oriented software engineering (OOSE, Jacobson) [58] into one language. UML consists of a collection of diagrams that visually express (parts of) a system's model. From this set, class diagrams are the most used diagrams for modelling software designs.

Even when students have an understanding of the syntax of a modelling language, the application of the language for creating a software design remains challenging. In particular software design is considered to be a difficult problem solving task, because it heavily depends on abstraction skills [69]. Students are constantly challenged to identify concepts from the problem domain and translate them to software artefacts. This, combined with the different views and abstraction levels that models can have, makes the understanding and application of modelling techniques challenging for students.

Educators face challenges at different levels:

- motivate and engage students in their courses,
- facilitate and use appropriate tools, and
- give feedback to students at the right level at the right time.

For helping educators to overcome the above mentioned challenges, we focus in our research on the involvement of education in the development of students' reasoning skills, students' software design strategies and the use of appropriate modelling and learning tools during software design tasks.

1.2 Background - UML, Design Principles, Tools, Agile Development and Theories of Education and Learning

This section discusses UML, the use of modelling tools, the application of common software design principles during software design, the increasing popularity of agile development and theories of education and learning.

[1] https://www.uml.org

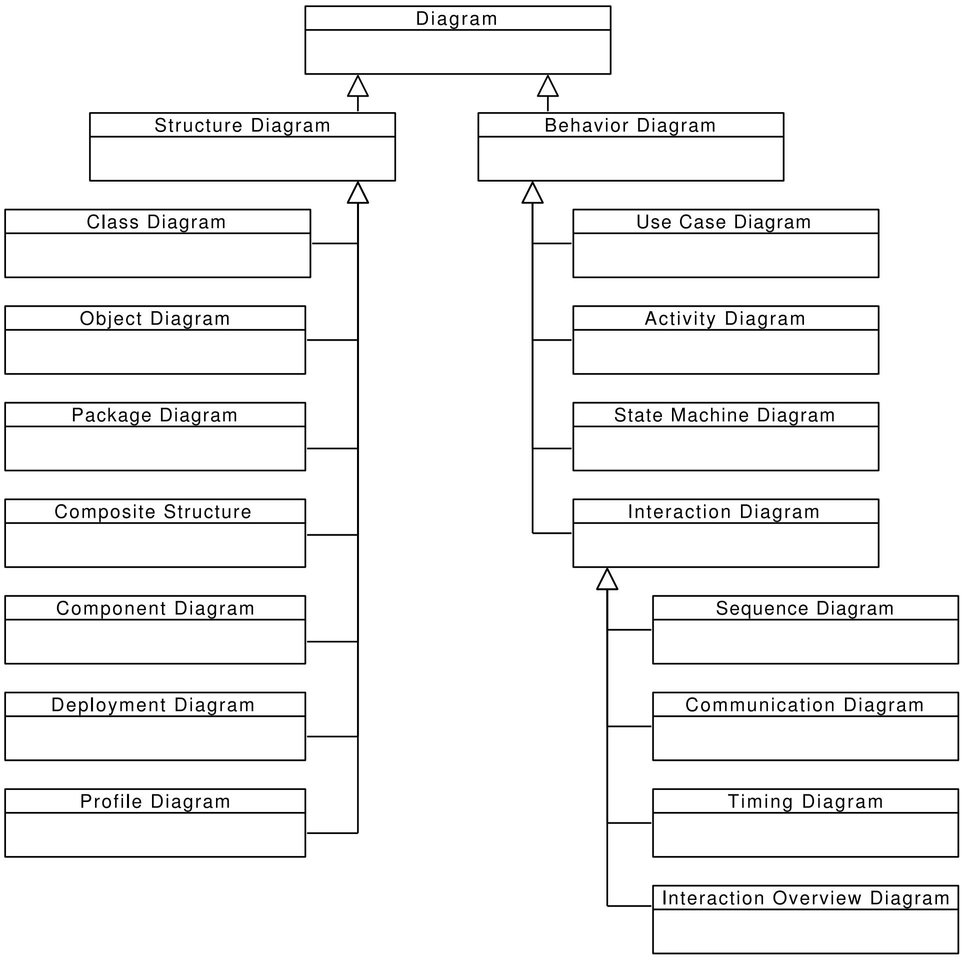


Figure 1.1: *Simplified diagram overview of UML 2.5.*

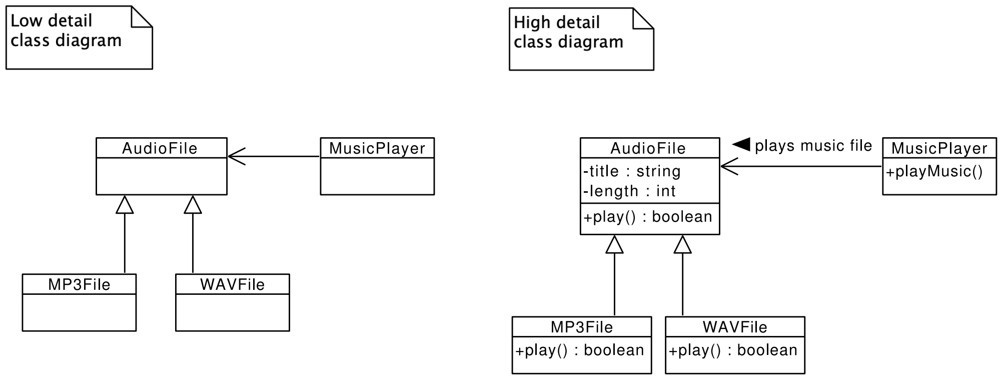


Figure 1.2: *A UML class diagram in low detail and in high detail*

1.2.1 Unified Modelling Language

UML is widely taught at universities and can be considered as the de facto standard for object-oriented modelling in SE courses. The universities that were involved in the studies that are discussed in this dissertation also use UML as their standard language for modelling during software development (analysis and design).

UML consists of several diagrams that can be used during analysis and design in software development methodologies. The diagrams describe mainly two types of views: structural views and behavioural views (see Figure 1.1).

The most commonly used diagrams from the *Structure Diagram* set are: the Class Diagram and the Component Diagram. From the *Behaviour Diagram* set the most commonly used are: the Use Case Diagram, the Activity Diagram, the State Machine Diagram and the Sequence Diagram [35]. In this research we mainly focus on the structural perspective of the design which is typically presented through class diagrams.

Class diagrams describe the (internal) structure of problem domains during analysis or software systems during design. In this dissertation we focus on the design phase(s). The diagram structure consists of related classes that can have attributes and operations. Eventually classes can be programmed with object-oriented software languages such as C++, Java or for example C#. Class diagrams are considered to be the blueprint of (parts of) the to-be system. Depending on the development stage or purpose of the diagram, a class diagram can have a low amount of detail or a high amount of detail. Figure 1.2 shows an example of a low detailed and high detailed class diagram of the same system.

1.2.2 Software Design Principles

In software development certain design principles are applied. A software design principle is a guideline for decision making during software development. Eventually it supports several benefits, such as:

- organised software (e.g. readable code)
- maintainable software
- changeable and extensible software
- less software bugs

In general such principles steer away from bad design constructions and thus aim to contribute to a better quality of the software.

An overview of key works in the field of software engineering that explicitly discuss design principles [20, 88, 73] are presented in Figure 1.3. The overview also shows the design principles that we consider as a generic design principle. These design principles do not only apply to object-oriented design but also to other design paradigms and software design in general.

In our research we use design principles as a basis for measuring students' design skills (see Chapter 2) and to evaluate the quality of their software design models. In this dissertation the following design principles play an important role:

- abstraction – define the conceptual boundaries (outside view) of an object by focusing on the characteristics of that object related to the context of a problem. This separates the object's expected behaviour from its implementation.
- modularity – split your software into separate units that are responsible for a specific task.
- low coupling – try to create less dependencies as possible to other software units. This lowers the amount of modifications in other (related) units when a unit has to be modified.
- high cohesion – software units should fulfil tasks and contain data that belong together.

We believe that this collection of principles constitutes a generic basis of the large collection of design principles. Moreover these principles are not specific to any programming language or paradigm.

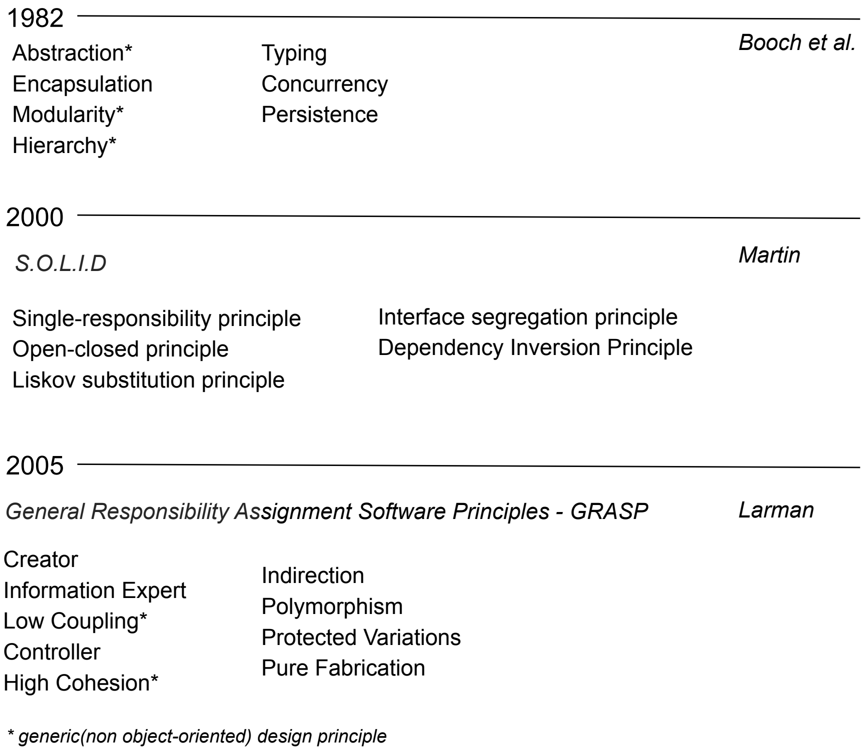


Figure 1.3: *An overview of the widely known software design principles.*

Unfortunately an optimal design is not achieved by optimising all principles individually. For example: lowering the coupling could result in lower cohesion. The challenge for software designers is to make design decisions that create an optimal balance of the applied design principles. Mastering this skill requires training and experience.

1.2.3 Tooling

There is an ongoing discussion going on about the use of (modelling) tools in education. Examples of topics of these discussions are: the difficulty of using tools, tools support and standard industry tools versus tailored tools for education [2, 83]. In our research we have looked at industrial modelling tools and educational tools (including serious games) or a combination of both (see Chapters 3 and 4)

1.2.4 Agile Software Development

In the past decades, there is an increasing popularity of evolutionary development methodologies such as agile. Instead of assuming that the set of requirements is clear from the start of a project, agile processes plan for very frequent feedback from stakeholders on the direction of the project. Openness to change is central to the way that agile development processes are organised. The consequence of such a flexible approach is that also the design process should be organised in such a flexible and incremental way. This requires for example the ability to evolve designs instead of 'freezing' a design from the start of a project. We explored whether designing software in an agile way contributes to the understanding of design concepts and to the understanding of design concepts in relation to the different software development phases (see Chapter 11).

1.2.5 Theories of Education and Learning

In this section we discuss related theoretical background on education and learning in general and related to software engineering. Based on the educators' challenges that were mentioned in Section 1.1 we discuss the following topics: course creation and the levels of learning, student motivation and didactic forms.

Creation of Courses and Learning levels Educators have to consider different matters when creating programs and courses within these programs. Matters such as:

- The main learning objectives of a program.
- The main learning objectives of a course.
- Smaller learning goals for lectures, workshops or other didactic moments.
- The order of courses and topics within courses.
- The increasing competence (e.g. knowledge and skills) levels (e.g. complexity).
- The prior knowledge of the students.

In addition, the field of software engineering develops constantly. Therefore programs are constantly subject to change.

Different taxonomies are supporting educators all over the world for designing learning programs. The following three are well know and most applied:

- Bloom's Taxonomy of Educational Objectives (revised by Anderson in 2001) [70] – a taxonomy that consists of three hierarchical models related to a domain: cognitive (knowledge-based), affective (emotion-based), psycho motor (action-based). The levels per domain build upon each other. The cognitive domain model is most used by educators for creating their courses. Later Anderson et al. revised the taxonomy. Amongst other things they added a dimension to knowledge, the kind of knowledge: factual, conceptual, procedural and meta-cognitive. This is the version that most people refer to as *Bloom' taxonomy*

- Fink's Taxonomy of Significant Learning [38] – a taxonomy that is not hierarchically organised. Each element in the taxonomy interacts with one and other. Like Bloom, Fink also has an affective part, but integrates this in the model. In addition Fink also recognises the importance of the meta-cognitive knowledge level.

- Wiggings and McTighe's Six Facets of Understanding [161] – like Fink a non hierarchical framework that is student centred. By examining six facets of understanding this should give a holistic picture of a student's understanding about a certain topic. The facets are examined with the help of questions to the student (e.g. 'What are examples of ... ?' or 'How might ... feel about ... ?'). You could say that this framework also integrates the affective area that Bloom handles as a separate domain.

Because of its popularity and the fact that the universities that collaborated with us in our studies also use Bloom's Taxonomy we chose to stay with Bloom's cognitive domain taxonomy.

Student Motivation As mentioned in Section 1.1 there is a challenge in motivating students for learning software design. Irvine maps several theories that are related to motivation in education onto two major theories: expectancy-value and intrinsic-extrinsic motivation [57]. Expectancy-value theory states that peoples' motivation depends on the belief that they have for being successful in a certain task and the value the task has for them. Intrinsic-extrinsic motivation theory states that people are motivated because: they are inherently interested in something or enjoy it; or, it leads to a distinct outcome (e.g. is/feels rewarded on success).

Literature shows [57] that when students are engaged in learning because of intrinsic motivation they develop a positive attitude toward the topic that was learned. While software design is an ever evolving discipline it is important for professionals to keep their knowledge and skills up to date. Intrinsically motivated students would become the life long learners that the industry needs.

It seems that intrinsic motivation can be influenced by educators. Positive feedback could enhance a student's motivation, while negative feedback would reduce it.

When people experience flow (an intense state, when a student is fully immersed in a task and enjoys it highly) it has an enormous positive impact on learning. Flow is the result of the balance between skills and challenge. An optimal situation is when skills and challenge are similar. When aware educators can use this for developing material. Flow is also a well-know term in games. Educational games could therefore be a good means of educating students.

In this research we explored the use of games and gamification and tailored tooling in relationship with students' intrinsic motivation and live long learning.

Didactic Forms In 2007 researchers state that constructivism (Piaget, later Papert: constructionism) is the most popular approach in computer science education [86]. Currently this is still the case. Although many didactic approaches are used, they mostly have their origins in constructivism.

In general constructivism states that learning occurs when students are actively involved in the process of gaining knowledge. The practical approach that computer science programs offer fits the idea of active learning.

Researchers observed a shift from traditional teaching towards a more active and practical approach. In general the following approaches are used in the field of software engineering [87]:

- traditional – classroom teaching and lab sessions (constrained assignment under supervision of a TA and/or lecturer).
- learning by doing – experience of clash of theory and practice by taking on a project.
- case study – investigation of a real-life problem that is recorded in one or several sources (e.d. documents, video's, etc.)
- problem- / outcome based learning(PBL) – a practical problem as driver for the learning process.

Based on a literature review in the field of teaching computational thinking (k-12 and university level) the researchers show that in the reviewed studies the following approaches stand out: game-based learning, collaborative learning, problem-based learning and project based learning as an answer on the need for another approach of teaching in this area [56]. The authors describe computational thinking as 'a process of problem-solving'. We see problem solving as an elementary skill for software design.

The didactic forms that were applied in the cases that this dissertation discusses were mainly practical and were aimed for training problem solving skills and for revealing students' reasoning to themselves and to the researchers.

1.3 Problem Statement

Several researchers point out that a large number of students have difficulties with (object-oriented) modelling during analysis and design of software systems [69, 50, 122, 62]. Students have to elicit concepts from the problem domain, transfer the concept of this domain to the solution space and propose software structures that should lead to maintainable software. These tasks require abstract reasoning skills and these skills seem to be difficult to teach. It is not well understood what the hurdles are in students reasoning while creating software designs. In this dissertation we explore the characteristics of students' learning in order to improve our educational approaches.

Some initial thoughts about the sources of the difficulties students face are:

- Modelling can be done in different phases of software development such as during analysis and design. During analysis students explore and model the problem. They are required not to 'think' about implementation details and see the 'big picture'. In contrast, when they are designing a solution they have to introduce implementation details into their designs. For students (and novice software developers) it is difficult to stay away from implementation. Especially when the solution domain is familiar for them. In addition it is also difficult for them to be aware of the development phase they are in. The fact that in software development the same UML notation (although with detail difference) is used for modelling in different phases confuses students.
- Modelling languages introduce the possibility of having multiple views of software systems (e.g. structural view and behavioural views, but also object view versus class/concept view). This seems a great feature, but is challenging for students. Identifying important concepts in the problem domain and creating extensible designs already requires a lot of abstraction power of the brain. Switching between views requires a high memory load for related diagrams (e.g. state machine diagrams that relate to instances (objects) from class diagrams).
- Modelling languages introduce different levels of abstraction. UML for example offers compressed versions of notations (e.g. a class without any detail, only the

name) and applies diagrams on different 'zoom levels' of a system (e.g. sequence diagrams on system levels and sequence diagrams on object level). We observe that students have difficulties to distinguish between abstraction levels and their purpose.

- A difference between modelling and programming is that with programming students have the possibility of continuous feedback by means of testing and compilation. This may positively affect the motivation for programming. With modelling this is not the case. Although it is possible to model in high detail, because of the general nature of (in our case) UML there is a gap between the model and the implementation. An executable model is therefore mostly not possible. We are aware of model driven environments such as low/no code platforms where the model is the implementation. But they are often developed for a specific problem (context).

Now and even more in the future, we depend on and trust software systems. At the same time these systems are not only growing in size, but also in complexity. Students should be well prepared and capable of handling these complexities. To keep future systems maintainable, modelling should play an important role [94].

1.4 Research Objective and Research Questions

This section presents the main research objective followed by the research questions of this dissertation.

The main objective of this dissertation focuses on how to improve our approaches in software design education:

Main Objective *How can we improve the ways of teaching software design?*

Not a lot research has been done on the particular topic of software design education, therefore we are motivated to contribute to this.

Education can be explored in many different ways. In our research we focus on the understanding of students' reasoning and decision making. This dissertation discusses three aspects of the educational process of students:

1. students' comprehension of software design,
2. students' motivation for learning software design, and

3. the learning process of students during software design tasks.

In support of the *Main Objective* the following research questions are defined:

RQ1 Can we assess how good students are at designing software?

RQ2 What guidance do students need to improve their understanding of designing software?

RQ3 How can we increase the motivation and engagement of students for learning software design?

Section 1.6 presents an overview that maps the chapters of this dissertation onto the research questions.

1.5 Research Approach

The research questions of this dissertation are answered with the use of empirical research. Empirical research gains knowledge by means of observation and experimentation. Empirical comes from the ancient Greek word for experience: ἐμπειρία (empeiría). Empirical research methods typically study real-world phenomena, such as human activities. Empirical research is being applied in several kinds of scientific fields, such as physics, psychology and medicine.

In the field of software engineering (SE) this research type is relatively young. Perry et al. discussed the state of empirical research in the SE domain in the year 2000 [102] and stated it to be relatively immature. Two years later Kitchenham et al. suggested preliminary guidelines for empirical research in the field of software engineering [65]. They suggested guidelines for six basic topic areas: experimental context, experimental design, conduct of the experiment and data collection, analysis of the results, presentation of the results, and interpretation of the results. Two examples of guidelines from Kitchenham et al. are:

> *If the research is exploratory, state clearly and, prior to data analysis, what questions the investigation is intended to address and how it will address them* – **experimental context**

> *Define the process by which the subjects and objects were selected* – **experimental design**

Currently empirical research is embraced more widely. The *International Conference on Model Driven Engineering Languages and Systems (MODELS)* publishes empirical work and has hosted the *International Workshop on Human Factors in Modelling* that was initiated to be a venue for empirical research involving human factors in modelling. Empirical work is also published by *The International Conference on Software Engineering (ICSE)*. This conference also organises the *International Workshop on Conducting Empirical Studies in Industry*. Other parties that embrace empirical studies in software design are: The *Computer Science Education Research Conference (CSERC)* and *Special Interest Group Computer Science Education (SIGCSE)*

In an empirical research, the collected data is analysed qualitatively or quantitatively, depending on the chosen method. The methods that were used for our research were experiments and case studies.

In this dissertation our main subjects are students. We investigate students' learning activities. An empirical approach is therefore suitable.

1.5.1 Experiment

An experiment is set up to measure a specific effect that was caused by the change of an (independent) variable. In most experiments the researcher wants to test his/her prediction (hypothesis). Data collection can be done with the use of a survey, or with measurements. Quantitative analysis is dominant in the experiment research method. In our research the experiments did not take place in a laboratory, they were *field experiments*. The experiment method is used in Chapters 2, 5, and 9.

1.5.2 Case Study

In a case study a group or person is studied in a particular situation. The researcher has an observing role. The data collection is often done by interviewing the subjects or let them 'think aloud' [81]. Qualitative analysis is dominant in case study researches. Often the analysis is used to create a theory or provoke a discussion. The case study method is used in Chapters 3, 7, and 8.

1.5.3 Mixed Methods

Some studies require a combination of qualitative and quantitative analysis: mixed methods. The mixed methods approach is used in Chapters 6, and 11.

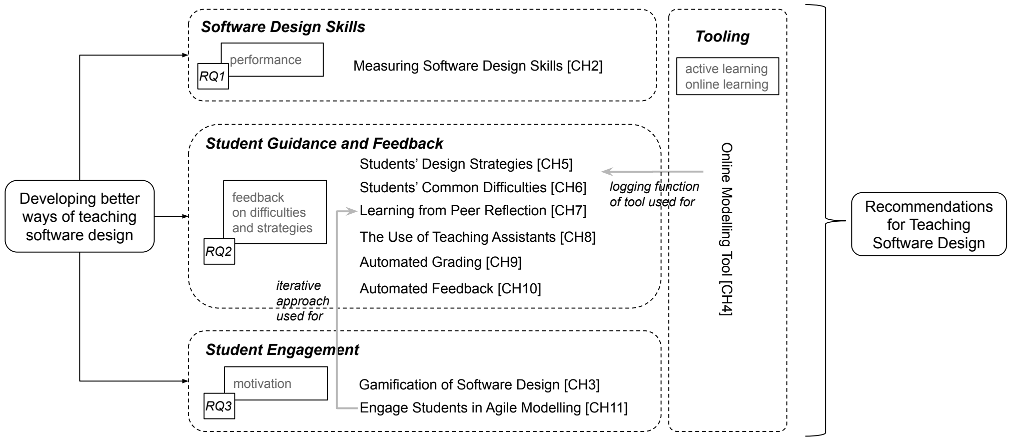

Software Design Skills
performance
RQ1
Measuring Software Design Skills [CH2]
Tooling
active learning
online learning
Student Guidance and Feedback
Students' Design Strategies [CH5]
Students' Common Difficulties [CH6]
Learning from Peer Reflection [CH7]
The Use of Teaching Assistants [CH8]
Automated Grading [CH9]
Automated Feedback [CH10]
feedback on difficulties and strategies
RQ2
logging function of tool used for
iterative approach used for
Online Modelling Tool [CH4]
Developing better ways of teaching software design
Recommendations for Teaching Software Design
Student Engagement
motivation
RQ3
Gamification of Software Design [CH3]
Engage Students in Agile Modelling [CH11]

Figure 1.4: *Overview of the research themes and their corresponding chapters.*

1.6 Contribution and Dissertation Structure

This research contributes to the improvement of software design education. The contribution is five-fold. First this dissertation provides a method for assessing students' software design skills. Evidence is found for: i) measuring the learning yield of students during a software design course and ii) the relation between abstract reasoning performance and the performance of students on software design tasks.

Second we identified engaging educational approaches and tools for learning software design. We showed this by using gamification and the integration of modelling in agile software development processes. We developed a modelling tool that enables on-line learning and equipped the research with the possibility to investigate larger groups of study subjects.

Thirdly our research contributes to the collection of common design strategies and difficulties in students' reasoning processes. With the use of the investigated educational approaches we gained insights in the type of guidance or feedback students need and we are able to make recommendations to lecturers in the field of software engineering.

Fourthly we explored the automation of grading and feedback for providing direct and/or on demand feedback for students. Based on the results of our approach that uses machine learning for grading we conclude that the models of the experiment were not accurate for 10 point grading, but may be used as a rough quality indicator of the quality of a software design. We developed a feedback agent module for WebUML to explore automated guiding when students perform software design tasks. Based on the early findings, we see a future for integrating feedback agents into future learning environments.

Lastly, with our findings we aim to contribute to research approaches and tools for studying students doing software design.

Figure 1.4 shows the set of ideas and approaches we have developed. We started off with a tool for assessing student's knowledge of software design (using classical multiple choice testing). As a complement, we moved into the area of monitoring and assessing students while *doing* design. In that area, we found that feedback during the doing of design is a good aid in learning; i.e. instead of assessing for the purpose of grading, we looked into assessing for the purpose of providing feedback and thereby enhancing the learning. In line with this theme, we also looked into enhancing the engagement of students in the learning of software design. In addition Figure 1.4 illustrates the relationships between the contribution themes and the chapters of this dissertation. Some chapters relate to previous chapters. This is also illustrated in the figure.

The structure of the remainder of this dissertation is as follows:

Part II - Software Design Comprehension

Chapter 2 – An Instrument for Measuring Design Skills The main objective of this chapter is i) to create an instrument that assess students' software design skills and ii) investigate the relation between abstract thinking abilities of students and performance of software design tasks. (supports RQ1)

Part III - Tooling

Chapter 3 – Exploring the Application of Game-Based Learning in Software Design Education The main objective of this chapter is to explore the influence gamification has on the engagement of students in software design activities. (supports RQ3)

Chapter 4 – WebUML - a UML Class Diagram Editor for Research and Online Education The main objective of this chapter is to present our on-line UML editor. The editor is a research instrument that was needed to conduct the several experiments that are discussed in the different chapters of this dissertation.

Part IV - Student Guidance and Feedback

Chapter 5 – Strategies of Students Performing Design Tasks The main objective of this chapter is to reveal common strategies students use to solve their software design tasks. (supports RQ2)

Chapter 6 – Uncovering Common Difficulties of Students Learning Software Design The main objective of this chapter is to uncover difficulties students have during a software design task. (supports RQ2)

Chapter 7 – Teaching of Agile UML Modelling: Recommendations from Students' Reflections The main objective of this chapter is to distil students difficulties in the process of analysis and design modelling by analysing their peer-reflections. From the distilled difficulties recommendations for the guidance of software engineering students are presented (supports RQ2)

Chapter 8 – Evaluating Didactic Approaches used by Teaching Assistants for Software Analysis and Design using UML The main objective of this chapter is to create a profile for teaching assistants that are involved in the education of software design courses. (supports RQ2)

Chapter 9 – Towards Automated Grading of UML Class Diagrams with Machine Learning The main objective of this chapter is to explore the application of machine learning for automated grading of class diagrams. (supports RQ2)

Chapter 10 – An Online Educational Agent: Automated Feedback for Designing UML Class Diagrams The main objective of this chapter is to explore the application of automated feedback during software design tasks with a didactic software agent.(supports RQ2)

Part V - Student Engagement

Chapter 11 – A Workshop for Agile Modelling The main objective of this chapter is to explore the integration of UML modelling in an agile development process in order to engage students in software modelling. (supports RQ3)

Part VI - Reflection

Chapter 12 – Conclusion and Future Work The closing chapter of this dissertation discusses recommendations based on pitfalls and practices of software design education, conclude this research and propose future work.

1.7 Publications

Below we list a chronological list of publications that were (co-)authored during this doctoral research:

1. Dave R. Stikkolorum, Michel R.V. Chaudron, and Oswald de Bruin. The art of software design, a video game for learning software design principles. In *Gamification Contest MODELS'12 Innsbruck*, 2012

2. Hafeez Osman, Arjan van Zadelhoff, Dave R. Stikkolorum, and Michel R.V. Chaudron. UML class diagram simplification: what is in the developer's mind? In *Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling*, page 5. ACM, 2012

3. Dave R. Stikkolorum, Claire Stevenson, and Michel R.V. Chaudron. Assessing software design skills and their relation with reasoning skills. In *EduSymp 2013. CEUR, vol. 1134, paper 5*, 2013

4. Seiko Akayama, Marion Brandsteidl, Birgit Demuth, Kenji Hisazumi, Timothy C Lethbridge, Perdita Stevens, and Dave R. Stikkolorum. Tool use in software modelling education: state of the art and research directions. In *the Educators' Symposium co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, 2013

5. Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty HC Cheng, Philippe Collet, Benoit Combemale, Robert B France, Rogardt Heldal, James Hill, et al. The relevance of model-driven engineering thirty years from now. In *Model-Driven Engineering Languages and Systems*, pages 183–200. Springer, 2014

6. Dave R. Stikkolorum, Birgit Demuth, Vadim Zaytsev, Frédéric Boulanger, and Jeff Gray. The MOOC hype: Can we ignore it? In *Reflections on the current use of massive open online courses in software modeling education. MODELS Educators Symposium, EduSymp*, 2014

7. Dave R. Stikkolorum, Truong Ho-Quang, and Michel R.V. Chaudron. Revealing students' UML class diagram modelling strategies with WebUML and LogViz. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*, pages 275–279. IEEE, 2015

8. Bilal Karasneh, Dave R. Stikkolorum, Enrique Larios, and Michel R.V. Chaudron. Quality assessment of UML class diagrams. In *Proc. Educators' Symp at MoDELS*, 2015

9. Dave R. Stikkolorum, Truong Ho-Quang, Bilal Karashneh, and Michel R.V. Chaudron. Uncovering students' common difficulties and strategies during a class diagram design process: an online experiment. In *Educators Symposium 2015, co-located with the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, 2015

10. Dave R. Stikkolorum and Michel R.V. Chaudron. A workshop for integrating UML modelling and agile development in the classroom. In *Proceedings of the Computer Science Education Research Conference 2016*, pages 4–11. ACM, 2016

11. 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
12. Dave R. Stikkolorum, F. Gomes de Oliveira Neto, and Michel R.V. Chaudron. Evaluating didactic approaches used by teaching assistants for software analysis and design using uml. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 122–131, New York, NY, USA, 2018. ACM
13. Boban Vesin, Aleksandra Klašnja-Milićević, Katerina Mangaroska, Mirjana Ivanović, Rodi Jolak, Dave Stikkolorum, and Michel Chaudron. Web-based educational ecosystem for automatization of teaching process and assessment of students. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, pages 1–9, 2018
14. Dave R. Stikkolorum, Peter van der Putten, Caroline Sperandio, and Michel R.V. Chaudron. Towards automated grading of uml class diagrams with machine learning. In *BNAIC/BENELEARN*, 2019
15. Goda Jusaite, Pim Sanders, Damani Lawson, Koen van Polanen, Hani Al-Ers, and Dave R. Stikkolorum. Improving the quality of online collaborative learning for software engineering students. In *International Academic Conference on Teaching, Learning and E-learning*, pages 8–15, 2020
16. Marcella Veldthuis, Matthijs Koning, and Dave R. Stikkolorum. A quest to engage computer science students:using dungeons & dragons for developing soft skills. In *Proceedings of the Computer Science Education Research Conference 2021*. ACM, 2021

In addition, the author has chaired symposia and conferences in the area of Modelling and Education :

1. Birgit Demuth and Dave R. Stikkolorum, editors. *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 29, 2014*, volume 1346 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014
2. Ebrahim Rahimi and Dave R. Stikkolorum, editors. *CSERC '19: Proceedings of the 8th Computer Science Education Research Conference*, New York, NY, USA, 2019. Association for Computing Machinery
3. Dave R. Stikkolorum and Ebrahim Rahimi, editors. *CSERC '20: Proceedings of the 9th Computer Science Education Research Conference*, New York, NY, USA, 2020. Association for Computing Machinery

4. Dave R. Stikkolorum and Ebrahim Rahimi, editors. *CSERC '21: Proceedings of the 10th Computer Science Education Research Conference*, New York, NY, USA, 2021. Association for Computing Machinery

5. Dave R. Stikkolorum and Ebrahim Rahimi, editors. *CSERC '22: Proceedings of the 11th Computer Science Education Research Conference*, New York, NY, USA, 2022. Association for Computing Machinery

Part II

Software Design Comprehension

Chapter 2

An Instrument for Measuring Design Skills

Description of contributions of the authors
The research in this chapter was done in collaboration with Claire Stevenson. At the time of writing, Claire was Ph.D. student in Psychology at the faculty of social science at Leiden University, where she specialised in the topic of statistics for testing. At the same time she studied for the bachelors program in computer science at Leiden University. As part of this bachelor program, she did a project-study on this topic in collaboration with Dave Stikkolorum. Claire introduced us to the tests that measure abstract reasoning and helped us to perform statistical analyses and interpret the results. The writing of the publication on which this chapter was based was a joint effort.

The contributions of Dave Stikkolorum were: creating the design skill test, running the experiment, and collaborating on the analyses and interpretation.

Michel Chaudron acted as supervisor to both Dave and Claire on this project.

In this chapter we present an instrument for assessing software design skills. In order to create educational interventions for teaching software design we need to know which reasoning skills are related to students' software design performance.

This chapter is based on the following publication: Dave R. Stikkolorum, Claire Stevenson, and Michel R.V. Chaudron. Assessing software design skills and their relation with reasoning skills. In *EduSymp 2013. CEUR, vol. 1134, paper 5*, 2013

We introduce an online test for measuring students' software design skills and relate those to abstract reasoning. Two student groups of two different European universities participated in an experiment in which we were able to relate students' visual and verbal reasoning skills to students' software design skills and measured learning improvement. In the future proper interventions can be chosen while using the test as a diagnostic tool.

2.1 Introduction

Lecturers from all over the world see students struggle with the subject of software design. Not only syntactic errors are made when using modelling languages like UML, but also semantic or organisation (design) errors. Kramer argues that the key lies in students' abstract reasoning skills [69]. One objective of our research is to discover which reasoning skills are related to the design skills of software engineering students. We focus on two types of abstract reasoning: visual and verbal reasoning. In this study the main question is:

RQ$_{main}$ Which type of knowledge and/or reasoning skills are related to students' software design skills?

This leads to the following underlying questions:

RQ$_1$ Can verbal or visual reasoning ability predict a student's software design skills?

RQ$_2$ Do language skills influence software design skills?

RQ$_3$ Does prior domain knowledge (UML) influence software design skills and learning?

Answering these questions can help lecturers to create educational interventions. In order to measure students' software design skills we developed a test. As far as we know there is no measurement instrument of software design skills. In this chapter we analyse two groups of students at two different universities. They participated in a series of tests addressing software design, modelling, reasoning and language skills. The remainder of this chapter is organised as follows: in Section 2.2 we describe related work. In Section 2.3 we describe our method. The results are presented in Section 2.4 and discussed in Section 2.5. We conclude and propose future work in Section 2.6.

2.2 Related Work

Several researchers have discussed the importance of subjects that should be included in the curricula of university software engineering programs [52] [53]. Especially inclusion of mathematics is a subject of discussion. Lethbridge found that software professionals remembered little mathematics from their study programs[77]. Some use this research to state that curricula emphasise mathematics too much while others, like Henderson use this as an argument to claim not to trust professionals' opinions[51], because there is too little research on the effect of mathematics on software engineering skills.

In our study we aim to identify what general reasoning skills (not only mathematical) are related to performance on software design. Bennedsen and Caspersen studied abstraction ability as indicator for students' learning performance on software engineering [14]. They were not able to find evidence for this relationship. Roberts [113] found positive correlation between abstraction ability and course grades, but observed a small number of students (N=15). We target a larger group of students (N=243), included language knowledge and used our test as main indicator of students' design ability.

2.3 Method

In this section we explain the research method employed to develop our instrument for measuring software design skills. We want the measure to show an increased knowledge of software design after following a course on software design. Therefore, we asked students to perform the test at the start (pretest) of a course and at the end (posttest) of a course. We found subjects for our test through two different courses on software design taught at two different universities in Northern Europe. We presented our design skills test as additional learning material.

In this section we describe our hypotheses. We address the participants and discuss the different types of test instruments that we used.

2.3.1 Hypotheses

In all hypotheses we focus on the effect of the independent variables on the level of design skills (dependent variable), shown in Table 2.1. The level of design skills is

Hypothesis	Construct	Description	Type of variable
1	UML Knowledge	UML syntax knowledge	Independent
2	Visual Reasoning	Raven figure series	Independent
3	Verbal Reasoning	Verbal analogies	Independent
4	Knowledge of English	C-Test for languages	Independent
all	Design Skills Pretest	Software Design Skills	Dependent
all	Design Skills Posttest	Software Design Skills	Dependent

Table 2.1: *Measured Constructs*

measured at two points in time: with a pretest and with a posttest. The hypotheses we want to examine are:

- H_1 - UML domain knowledge will not influence students' design skills.
- H_2 - Visual reasoning is related to design skills test performance.
- H_3 - Verbal reasoning is related to design skills test performance.
- H_4 - Knowledge of the English Language (language of our design skills test) is related to design skills test performance.

2.3.2 Participants and Data Collection

The students that participated in the test were 2nd year BSc. students from two universities in Europe. A group from Chalmers University in Gothenburg - Sweden and a group from Utrecht University in Utrecht - The Netherlands. Both groups had no or very little experience with software design at the start of the course. The initial number of students(N) was 243, however not all students participated in all tests during their course. For some parts of the analysis we had to use a smaller number of students (N=155 was the minimum). This was due students that left the course prematurely or due technical issues. Section 2.4.2 shows Table 2.2 with the descriptive statistics of the test instruments.

All data was collected with on-line multiple choice tests[1]. This was convenient for assessing a larger group of participants. We used an open-source questionnaire tool called LimeSurvey[2].

[1] A demo is available at: `https://designskillstest.drstikko.nl`

[2] `https://www.limesurvey.org`

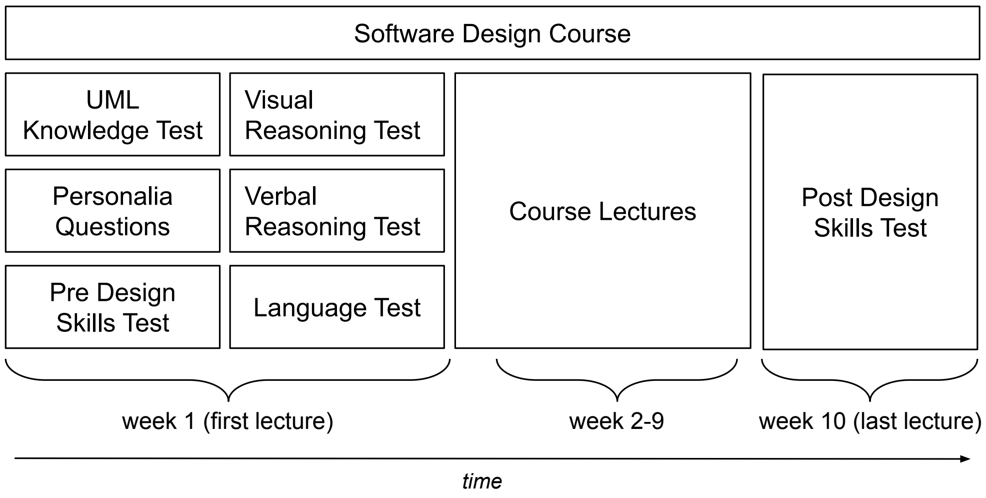


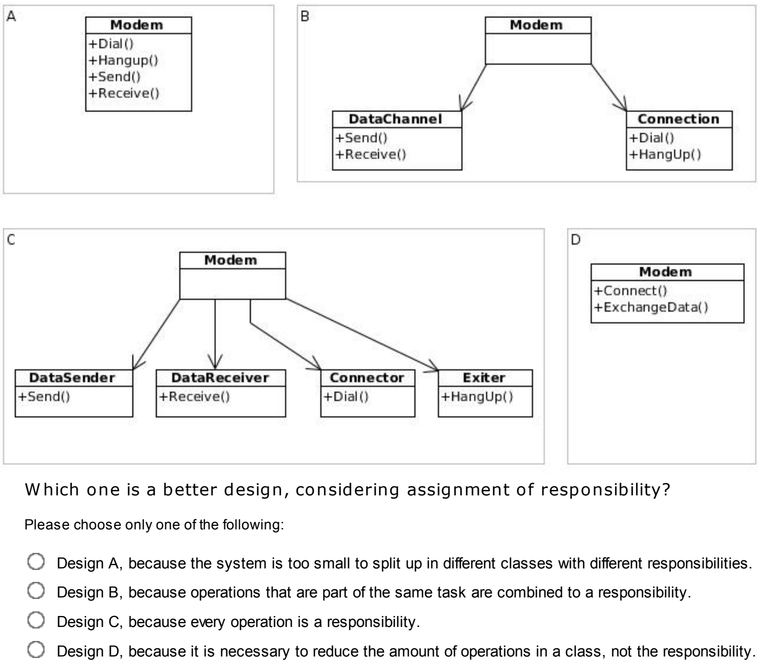
Figure 2.1: *Test construction in time dimension*

2.3.3 Experiment Design

In Figure 2.1 the organisation of the test is shown in the dimension of time. The whole experiment consists of 6 test parts: design skills pre- and posttest, UML Knowledge, Abstract Reasoning, Language Skills and one part that is about personal information. The experimental procedure was as follows: 1) In the first week students were administered the software design pretest, the UML prior knowledge test and answered general questions about age, background and experience. 2) In the next weeks they followed the software design course at their university and were asked to complete the verbal and visual reasoning tests. Also their level of English was tested in these weeks. 3) At the end of the course the students made the software design skills posttest.

Pre and Post software design skills tests The pre- and posttest both consisted of 20 similar multiple choice questions about software design principles such as mentioned in [112] and [88] with a total time limit of 40 minutes. In some questions the student is asked to compare different designs for the same system. An example of such a question is shown in figure 2.2. In other questions only one design was presented and students had to answer questions about this design. The designs were presented to the students in the Unified Modelling Language (UML[3]). UML is the most popular software modelling language at the moment of writing. We choose a very small subset of UML for the reason that we only see UML as a vehicle for designing software

[3] https://www.uml.org



Which one is a better design, considering assignment of responsibility?

Please choose only one of the following:

- ○ Design A, because the system is too small to split up in different classes with different responsibilities.
- ○ Design B, because operations that are part of the same task are combined to a responsibility.
- ○ Design C, because every operation is a responsibility.
- ○ Design D, because it is necessary to reduce the amount of operations in a class, not the responsibility.

Figure 2.2: *Example question from the design skills test*

systems. Lecturers and PhD students discussed about the possible answers. Only those questions were elected, where they agreed on the answer. The cognitive difficulty levels we used are up to level two of Bloom's taxonomy of educational objectives [159]: 'understanding'.

UML prior Knowledge A set of 22 items about UML syntax knowledge was administered after the pretest to be able to study the relationship between prior UML knowledge and design skills afterwards. There was a 20 minutes time limit for this UML test.

Language and Reasoning tests We focused on three possible types of knowledge and/or skills that could be related to software design skills: English language knowledge, verbal reasoning and visual reasoning. In order to study the relationship between the performance on the design skills test we asked the subjects to make a test that measures these skills. For the language knowledge we used the automated C-test for languages from Leuven University[4][12]. For verbal reasoning we used a verbal analo-

[4]http://www.arts.kuleuven.be/ctest/english - not available anymore

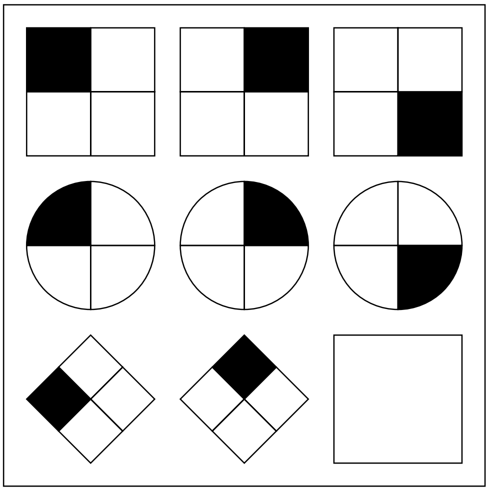


Figure 2.3: *Raven test example question*

.... stands to book as painter stands to ....

1. Chapter	2. Writer	3. Reading	4. Word	5. Literature
A. Painting	B. Picasso	C. Brush	D. Workshop	E. Paint

Figure 2.4: *Verbal analogies example question*

gies test[5], for visual reasoning we used a test based on Raven's progressive matrices [110]. Example questions can be seen in Figures 2.3 and 2.4. The time limit was 60 minutes.

Personalia After the first test students were asked five questions about their personal situation. Topics were: gender, age, prior software design experience, education and work experience.

2.4 Results

In this section we describe the results of the individual test instruments. The analysis[142] of this data will be discussed in section 2.5. We show psychometric properties, descrip-

[5] https://www.fibonicci.com/verbal-reasoning/analogies-test

tive statistics, investigate correlations and compare the universities' performances. The student groups from the universities are anonymized and shown as 'A' and 'B' or we consider the groups as a total.

2.4.1 Psychometric Properties

We used classical test theory to determine reliability of our instruments. Cronbach's α coefficient of internal consistency was 0.44 for the pretest, 0.58 for both the posttest and UML knowledge test. The α is somewhat low because of measuring different knowledge constructs. The item difficulty (i.e., proportion correct) was lower for the pretest (M=0.59, SD=0.17, range=0.21-0.82) than the posttest (M=0.68, SD=0.17, range=0.25-0.89). For the UML knowledge test the students solved on average 41% of the items correctly (M=0.41, SD=0.25, range=0.09-0.90).

2.4.2 Descriptive Statistics

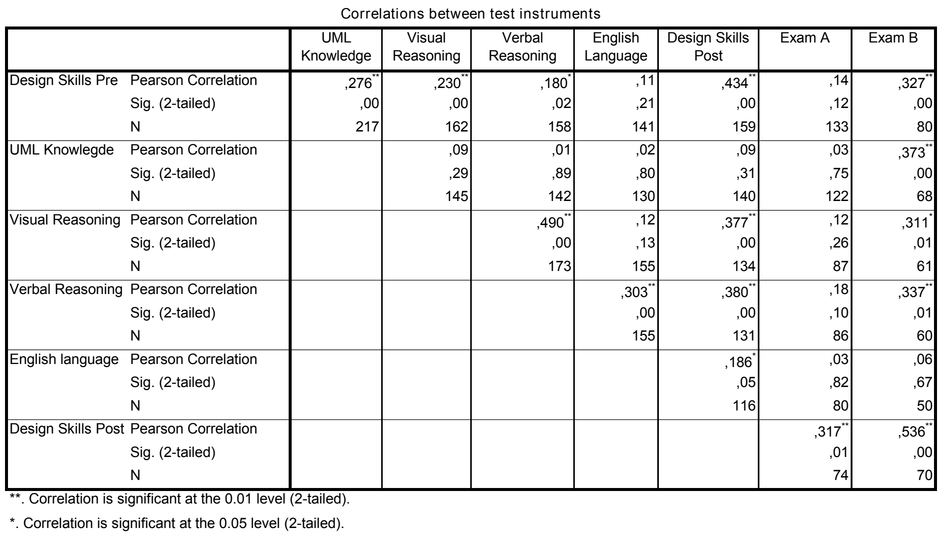
Table 2.2 shows the number (N) of students that participated per test, Minimum (Min) and Maximum (Max) score, Mean (M) score, standard deviation (SD), the Skewness (Skew) and Kurtosis (Kurt). We excluded students' responses if they responded to less than 50 percent of the questions on a test.

Construct	N	Min	Max	M	SD	Skew	Kurt
Design Skills Pre	243	3	19	11.73	2.75	-.31	-.03
UML Knowledge	217	2	19	9.11	3.12	-.09	-.21
Visual Reasoning	177	0	18	13.27	2.80	-1.41	4.24
Verbal Reasoning	173	0	15	9.05	3.06	-.55	-.12
English language	155	0	38	25.31	8.08	-1.31	1.86
Design Skills Post	171	5	19	13.41	3.00	-.44	-.15

Table 2.2: *Descriptive Statistics Test Instruments*

2.4.3 Correlations Between Instruments and Linear Regression

Figure 2.5 shows the Pearson correlations that were found between the individual tests. A correlation coefficient of 0.10 is considered as a weak relationship, 0.30 as moderate, and 0.5 as a strong relationship [28]. Figure 2.5 show a significant ($p < 0.01$


Correlations between test instruments

		UML Knowledge	Visual Reasoning	Verbal Reasoning	English Language	Design Skills Post	Exam A	Exam B
Design Skills Pre	Pearson Correlation	,276**	,230**	,180*	,11	,434**	,14	,327**
	Sig. (2-tailed)	,00	,00	,02	,21	,00	,12	,00
	N	217	162	158	141	159	133	80
UML Knowlegde	Pearson Correlation		,09	,01	,02	,09	,03	,373**
	Sig. (2-tailed)		,29	,89	,80	,31	,75	,00
	N		145	142	130	140	122	68
Visual Reasoning	Pearson Correlation			,490**	,12	,377**	,12	,311*
	Sig. (2-tailed)			,00	,13	,00	,26	,01
	N			173	155	134	87	61
Verbal Reasoning	Pearson Correlation				,303**	,380**	,18	,337**
	Sig. (2-tailed)				,00	,00	,10	,01
	N				155	131	86	60
English language	Pearson Correlation					,186*	,03	,06
	Sig. (2-tailed)					,05	,82	,67
	N					116	80	50
Design Skills Post	Pearson Correlation						,317**	,536**
	Sig. (2-tailed)						,01	,00
	N						74	70

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Figure 2.5: *Correlations between the individual test instruments*

) moderate relationship (r = 0.377) between visual reasoning and the design skills posttest. This also counts for verbal reasoning and the posttest (r = 0.380, p < 0.01). The visual and verbal reasoning tests do not have this relationship with the design skills pretest. The English language test does not seem to correlate with other tests. There is a moderate to strong relationship between the verbal and visual reasoning tests (r = 0.490, p < 0.01). Also the design skills pre- and posttest have a moderate to strong (r = 0.434, p < 0.01) correlation. We found a moderate correlation between posttest and the exam of university A (r = 0.317) and a strong correlation between posttest and the exam of university B (r = 0.536) both at significant level of 0.01. A series of linear regression models were used to investigate which factors (pretest, verbal reasoning, visual reasoning, UML knowledge or English language proficiency) best predicted the student's posttest performance. The best fitting parsimonious model explained 34% of variance (F(3, 121)=122.36, p<.001) and is represented by posttest = β_{pre} • pretest + β_{vis} • visual reasoning + β_{verb} • verbal reasoning. With β_{pre}=.40, t_{pre} = 5.27, p_{pre} < .001 ; β_{vis}=.14, t_{vis} = 1.63, p_{vis} = .11 and β_{verb}=.25, t_{verb} = 2.99, p_{verb} < .01

2.4.4 Comparison Between Universities

We compared the performance of all instruments between the universities. We found significant differences between the scores on the UML Knowledge test and the C-test. University A performed better on the C-test (M_A=27.06, SD_A=8.2, M_B=24.11, SD_B=7.9, t(153)=2.27, p=.03). University B performed better on the UML test (M_A=8.3, SD_A=3.03,

M_B=9.8, SD_B=3.04, t(215)=3.57, p=0.00).

2.5 Discussion

The correlation coefficients show that both verbal and visual reasoning explain almost 40 percent of the performance on the students' design skills posttests. This is in contrast with the correlation of these skills with the design skills pretest. This indicates abstract reasoning contributes to improvement of software design skills ($H_{2,3}$). We did not use a control group. One could argue improvement of skills is due to retesting and not due to learning. The correlation between the posttest and the exam scores provides evidence the we measure learning improvement. We used tests that are considered not trainable. They measure students' abstract intelligence. This means we have to investigate the specific sub tasks related to abstract intelligence or how problems are presented during lectures for those that do not have this 'natural talent' for abstract reasoning. The fact that both the UML knowledge and language test had no correlation with the design skills pretest and posttest ($H_{1,4}$) indicates that we indeed succeeded in questioning design concepts and not UML notation problems. Also the fact that university B performed better on the UML knowledge test while both universities did not perform significantly different on the design skills pretest provides further support for trusting our test is mostly independent from knowledge of UML notation. The students achieved higher scores on the design skills posttest than on the design skills pretest. This indicates that they have learned design skills during the course.

2.6 Conclusions and Future Work

In this chapter we presented our findings of an on-line test for measuring software design skills and abstract reasoning skills of students. We showed the relationship between abstract reasoning and the ability of solving software design problems. Although abstract intelligence cannot be trained, we see challenges and opportunities in exploring educational interventions for specific reasoning tasks and/or alternative teaching methods. We believe game based learning could be used. This is motivated in our other research in game based learning (Chapter 3). We already gained positive feedback on a pilot of our motivational game 'The Art of Software Design'[6][31][133]. We plan to extend the game with the findings of this experiment.

In the future, indicated by our regression model, lecturers can use our test to diag-

[6]https://gamejolt.com/games/the-art-of-software-design/21373

nose students and choose appropriate interventions when educating software design students.

Part III

Tooling

Chapter 3

Exploring the Application of Game-Based Learning in Software Design Education

Description of contributions of the authors
The research in this chapter was done in collaboration with Oswald de Bruin. At the time of writing Oswald followed the master program on computer science at Leiden University. He took part in the discussions about the game design. He designed the 'look and feel', implemented and tested the game.

The contributions of Dave Stikkolorum were: taking part in the discussions about the game design, research design and writing the publication. Michel Chaudron acted as supervisor to both Dave and Oswald on this project.

This chapter describes an exploratory study on the application of game based learning for learning software design. Based on software design principles we developed a game for training students in software design principles. The game aims to engage and motivate the player for learning, using guided feedback in a challenging level based puzzle game with increasing difficulty. We used Bloom's taxonomy to determine the learning objectives of the different game levels. The

This chapter is based on the following publication: Dave R. Stikkolorum, Michel R.V. Chaudron, and Oswald de Bruin. The art of software design, a video game for learning software design principles. In *Gamification Contest MODELS'12 Innsbruck*, 2012

learning objectives let the students learn to cope with the complex balancing problem of interfering design principles during software design. Players are supported while evaluating their design actions by means of the visualisation of control- and data flows. The game based approach supports the educational approach of 'learning by doing'. Initial user tests indicate player's engagement and a possible learning effect.

3.1 Introduction

Designing software systems is one of the difficult tasks in the field of software engineering. It is no surprise that learning software design is a difficult task for students too [80][69][113][141]. They have to face the challenge of abstracting structure and behaviour for possible software solutions. Next to these abstraction skills, students have to learn to integrate them in a software development process. For learning software design skills, students have to practice much in order to get familiar with the tools and the application of their design skills in project settings.

In their literature study Marques et al. conclude there is a need for teaching software engineering in a practical way [87]. They report a shift in studies of teaching approaches from traditional approaches towards learning by doing. For software engineering they report that most studies (until 2014) focus on:

- *learning by doing* – experience of clash of theory and practice by taking on a project.
- *case study* – investigation of a real-life problem that is recorded in one or several sources (e.d. documents, video's, etc.).
- *problem/outcome based learning (PBL)* – a practical problem as driver for the learning process.
- *games* – game development as fun element for learning.
- *traditional* – classroom teaching and lab sessions (constrained assignment under supervision of a TA and/or lecturer).

In our research and educational practice we learnt that universities offer a variety of approaches that enable students to practice as much as possible. Dominant approaches for learning software design are:

- *lab sessions* with practical assignments under the supervision of the lecturer and possibly teaching assistants. – *traditional*
- *group work sessions* with discussions (similar to lab sessions). – *PBL / traditional*
- *homework assignments* that are discussed during the lectures. – *traditional / flipped classroom*
- *projects* in which students have to deliver a solution for a more complex problem than home work assignments or lab session assignments. Projects can solve a real-life problem (even with a 'real' client) or students can be provided with an artificial problem. – *mixed approach*

As introduced in Chapter 1, there is a challenge for lecturers to motivate students for learning software design. A difficulty of learning software design, is that it heavily depends on abstract reasoning. This abstract nature makes it difficult to create assignments that appeal to the students and keep them engaged. On one hand lecturers have to create assignments that focus on a particular issue while in reality one will face the integration of multiple issues. On the other hand more complex problems can be addressed in project-based approaches but can be overwhelming for novice software designers and are often too complex for the classroom setting [23] (engaging students in complete software development processes is discussed in Chapter 11). With both of the approaches there is a risk that the students lack overview of the matter and therefore this could have a negative influence on the engagement.

In this chapter we explore an additional didactic approach for learning software design. We introduce the gamification of a software design environment as part of our bachelors' software engineering course. We aim to motivate students to learn software design by providing them with learning material in the form of an interactive computer game that relates to software design concepts and uses a graphical interface that relates to a modelling tool for making software designs. By slowly introducing concepts we aim to keep the students overview clear. A better overview should support the engagement of the student in a positive way.

In this chapter we discuss our gamification approach, give insight in the game itself and discuss early findings based on think aloud sessions and user testing.

In this chapter we use the term 'design' rather than 'modelling', because we see modelling as a vehicle for designing. The game uses modelling as a vehicle for

- explaining software design principles,
- discovering the usefulness of models for abstract reasoning about a design, and

- finding the proper abstractions for representing a system in a domain.

In this chapter we use the term 'learning by doing' in the sense of experience learning by practising by trial and error. We do not see this as the only right approach. We think that teachers should mix didactic approaches dependent on the context and needs of their students.

We mix the terms 'gamification' and 'game' in this chapter. In our opinion we gamified (applied gamification) the software design tasks by approaching the software design challenge students face as a puzzle. On one hand we applied gamification: we build and environment according to the analogy of an UML design tool and introduced scoring and progress elements. On the other hand we went a bit further and built a game: it contains levels, with clear level goals, feedback guides and gameplay. We build and educational puzzle game by strongly gamifying the software design task. The eventual product could be seen as a proof of concept.

The remainder of this chapter is structured as followed. In Section 3.2 we discuss related work. In Section 3.3 we describe our method, in Section 3.4 we describe the game design. After a 'walk through' of the game in Section 3.5 we evaluate and discuss in Sections 3.6 and 3.7. Finally we conclude and propose future work in Section 3.8.

3.2 Related Work

Prensky introduced Digital Game Based Learning [104]. He states: "Video games are not the enemy, but the best opportunity we have to engage our kids in real learning" [105]. He suggests to make use of the interest in (video) games of young people for the engagement in learning.

Currently the use of video games in education is widely applied by teachers from different levels of education. Souza et al. show in their literature study of 106 studies a variety of the application of game related approaches in the field of software engineering [126]. The categorised the didactic approaches as follows: Gamification, Game Development Based Learning, Game Based Learning and Hybrid. The authors found a predominance of the application of GDBL for the knowledge area of software design: students learn by developing a game.

Sheth et al. introduced a hybrid form: a code tournament and a gamification of the testing phase [121]. The authors aimed to engage students into software design and testing by the means of competition and collaboration. First the students were involved in a code tournament for learning to make good software design decisions.

The assignment was to create a battle ship game that uses a particular interface that was also used by an AI. The main objective was ‘programming to an interface’. They had to compete by playing against the AI in a tournament. Second the students learned about testing by using a gamification of testing techniques such as white - and black box testing. Students are given a number of quests that have to be complete. The quests disguise the testing techniques. The authors report that the qualitative feedback shows that most of the students were very positive about the experience.

Rusu et al. developed a puzzle-based game for the introduction of design patterns to middle school to college junior students [118]. The game is similar to *Lemmings*. In the game robots have to find their way like the lemmings in the original game. The player performs specific actions that resemble a software design pattern, for example: a robot with binoculars is watching a robot that signals that is safe to cross a bridge, the observer pattern. A user study indicated, besides a new appreciation of software engineers, that the students learned about the concepts that were thought. A questionnaire was used where specific scenarios were presented and the students had to choose between metaphorical roles of the robot, such as observer, mediator, etc. The students were not familiar with the topics before and did not specifically learned about the terminology.

In the field of software engineering research can be found about games for learning to program (GBL approach). Most of these games have children as their target audience. Less research is found about games for software design for students.

Lee et al. present their game *Gidget* [74] [75]. Gidget is a game for learning to program. In this game learners learn to program by debugging code that has failures. The failures are made by a robot character that has a damaged chip. By solving puzzles the learner is confronted step-by-step with programming concepts. In addition Lee et al. found that a personification of the feedback (Gidget the robot) had a positive influence on the players’ motivation.

Although environments like Alice, Greenfoot or Scratch are not game environments, they do break with more conventional programming environments. In stead of sorting a list of numbers or generating prime numbers they try to connect with the learners interests, such as games, stories and simulation [152]. While Greenfoot and Scratch provide the user a scene that can be filled with game objects from a repository, Alice even provides a 3D game engine.

In the field of architecture modelling Groenewegen et al. [41] introduced a game for validating architecture models. Groenewegen et al. used a board game where the actual models play a role. We also aimed to create a video game where the models itself are game elements.

BlueJ is not a game, but a Java IDE especially designed for an introduction to programming [68]. Later the authors developed Greenfoot. What makes BlueJ worthwhile to mention here is that it is an IDE that gives the UML design a central role. A programmer starts by making a visual design, a class diagram.

By providing the students an environment in which they practice modelling while playing the game, we support a 'learning by doing' approach [33]. This approach is already widely applied for learning students to program [156, 114, 15, 128].

Tenzer et al. present their tool GUIDE that supports the exploration of a software design by playing a game. [149]. This tool supports the process of a multiplayer game in which so called Verifiers and Refuters compete to proof the correctness of a design. By the means of the game a software design can be improved and flaws can be eliminated. The game should help to cope with the difficulty of not being able to apply formal verification techniques on incomplete and/or informal UML models. The tool, and thus the game, aims at designers that are already (basically) skilled in software design. It does not aim at learning in particular. This in comparison to our game, which is meant for novice learners.

Fernandez-Rayes et al. implemented the gamification of an achievement-driven advanced software design course [37]. During the course students compete with each other by attacking and stealing points with the help of virtual cards, represented as text and values in a spreadsheet. They found (partial) evidence that students have higher grades for the course when they are motivated for playing the game. Although game based learning is applied in the field of software engineering, we were at the moment of writing our publication not aware that such a video game as *AoSD* in the field of software design exists. Also, a more recent literature study does not report a similar game [126].

3.3 Method

The aim of this study is to explore if a game supports students in learning software design. The game we have built is created using an iterative approach. The authors came together periodically and discussed and improved their different ideas until there was a first version.

The learning objectives of the game are chosen using Bloom's taxonomy [70] combined with a set of general design principles [88] : coupling, cohesion, information hiding and modularity (discussed in 1.2.2). Bloom's taxonomy is widely used by lecturers. 'Design principles' is a typical topic in a software engineering curriculum. The learning objectives are explained in Section 3.4.1

During development we tested the game by conducting a user test in the form of a simple version of the 'think aloud' [81] method. We asked the test users to articulate their thoughts while making steps in the game. We improved elements of the game and fixed bugs based on these sessions. Subsequently we addressed a larger group for testing the game and answering a questionnaire before (N=67) and after playing the game (N=13). We are aware that this is not a complete validation. As mentioned before there is no game to compare with.

3.4 Game Design

In this section we discuss the learning objectives, the type of the game, game levels, the role of UML, the game mechanics and the software that was used to implement the game.

3.4.1 Learning Objectives, The Aim of The Game

The main learning objective of the game is to introduce the software design principles coupling, cohesion, information hiding and modularity.

We categorised the learning objectives according to Bloom's taxonomy for the cognitive domain. Because the game is intended for giving an introduction we mostly focused on levels up to the *apply* level.

RECALL

- Students will know what the following concepts mean: classes - associations - methods - attributes - packages - coupling - cohesion - interface - agents - data flow - control flow.
- Students will know the following software design principles: low coupling - high cohesion - information hiding.

UNDERSTAND

- Students understand the relation between the different elements that form a class diagram.

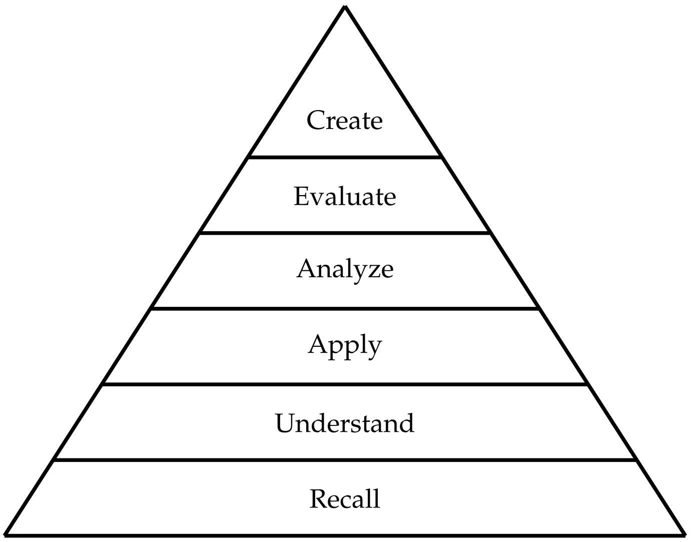


Figure 3.1: *Bloom's Taxonomy – overview of the cognitive domain levels*

- Students understand that there is a trade-off when applying the design principles coupling and cohesion.
- Students understand that there are multiple solutions for creating a software design.

APPLY

- Students can assign methods and attributes to classes in a simple design (4-6 classes) chosen from a list of suggestions.
- Students can associate classes in a simple design (4-6 classes).
- Students can apply the high cohesion principle to a simple class design (4-6 classes).
- Students can apply the high cohesion principle to a simple class design (4-6 classes) with a simple packaging (2-4 packages).
- Students can apply the low coupling principle to a simple class design (4-6 classes).
- Students can apply the low coupling principle to a simple class design (4-6 classes) with a simple packaging with the use of an interface.

3.4.2 Type of Game

We chose to make the game a puzzle game. Software design is a complex balancing problem. The designer has to solve the problem by finding the best trade-off considering different design principles that interfere. This complex task involves abstract reasoning and has multiple solutions.

We see looking for different solutions for balancing design principles as a similar task to solving a puzzle. The type of puzzle is a logic puzzle in which the students synthesise a solution that satisfies multiple logical constraints. An example of this is balancing between coupling and cohesion. In both situations the person uses his/her abstract reasoning skills.

We are aware that in reality the solution space for a software design problem may be wider than a puzzle. We think that is wise to restrict the solution space for novice software designers. Therefore a puzzle game is suitable. It provides a 'closed world' problem that is in control of the lecturer: he/she can choose the width of the solution space.

3.4.3 Game Levels

For each of the design principles, there is a set of levels that offers puzzles of increasing complexity. A puzzle offers a design fragment - typically a set of classes, attributes and methods - and asks the player to complete the design.

Each level starts with an explanation pop-up of the concept that is handled. A somewhat loose definition for the concept is given. The definition is loose because we do not want to emphasise on definitions. In the explanation pop-up also the challenge of the level is given.

The moves that a player can make differ per level. For simple levels, there are predefined classes and methods that the player can move around and connect to existing classes in the design. At more advanced levels, the player can also create new classes.

The initial levels of the game test for understanding the main concepts in isolation. Subsequent levels offer puzzles that require combined understanding of multiple design principles. Subsequent levels unlock when all linked preceding levels are finished (this is explained in Section 3.4.6).

The player receives hints when hovering over an item from the toolbox as well as when hovering over elements that are placed in the drawing area. Hints in the drawing area

Table 3.1: *The implemented levels of 'The Art of Software Design'*

Level	Description
Classes	Introduction to classes and the mechanics of the game.
Associations	Introduction to association.
Methods	Introduction to methods and their functionality.
Attributes	Introduction to attributes and their role.
Packages	Dragging classes to packages.
Coupling	Lower the coupling in a design.
Cohesion	Drag methods and attributes to classes and get an as high as possible cohesion score.
Control flow	A tree-like structure in which the player has to change the control flow in a certain way.
Data flow	Drop an attribute in the right class to let its data flow to the right class.
Package cohesion	Similar to the cohesion puzzle, but now cohesion within packages is taken into account.
Coupling v.s. Cohesion	Put 4 methods in as many or few classes to optimise the coupling and cohesion.
Interfaces	Maintain the data flow between 2 packages while only using 1 association between them.
Agents	Construct an agent between 3 classes to make a data flow.
Information hiding	Extract relevant information from a shielded data flow, without a certain class having access to the shielded data flow.

help players forward to their solutions.

Table 3.1 show the 14 puzzle levels that are created for the proof of concept of the game that was used for this research.

3.4.4 UML Notation

The game uses a notation that is close to UML. Although this notation is used for representing software designs, the game is not intended for learning the syntax of UML. We prefer to address software design principles in a more general manner. Therefore we use a notation that is a very simple subset of UML and tried to stay away from object-orientated dependent principles.

The subset we chose only consists of class diagrams because our research mainly focuses on the structural part of software design. Within the rich collection of class diagram elements we again chose for a restricted set that covers the basics of software design.

For our research we use the following UML elements:

- Package
- Class
- Attribute and Operation
- Visibility (only private and public)
- Data type
- Relationships – Dependency, (directed) Association, Aggregation, Composition and Inheritance
- Multiplicity

Examples of the UML elements that we don't use are: Interface, Realization, Abstract Class, Stereotype, Collection.

For the more high level assignments we use the condensed form of the class notation: without attributes or operations. Other assignments demand more detail in the diagram. The two different notation variations, accompanied with an example of the graphical variant that is used in the game, are illustrated in Figure 3.2.

Differences with UML For learning purposes we chose to use notation elements in additional to the UML subset. For emphasising the input- and/or output roles for method arguments we equipped these with the following prefixes:

- 'io' – a method argument that has both an input and an output role.
- 'i' – a method argument that has an input role.
- 'o' – a method argument that has a output role.

The output role can be compared with a method return result or a method argument that can be manipulated (e.g. a reference)

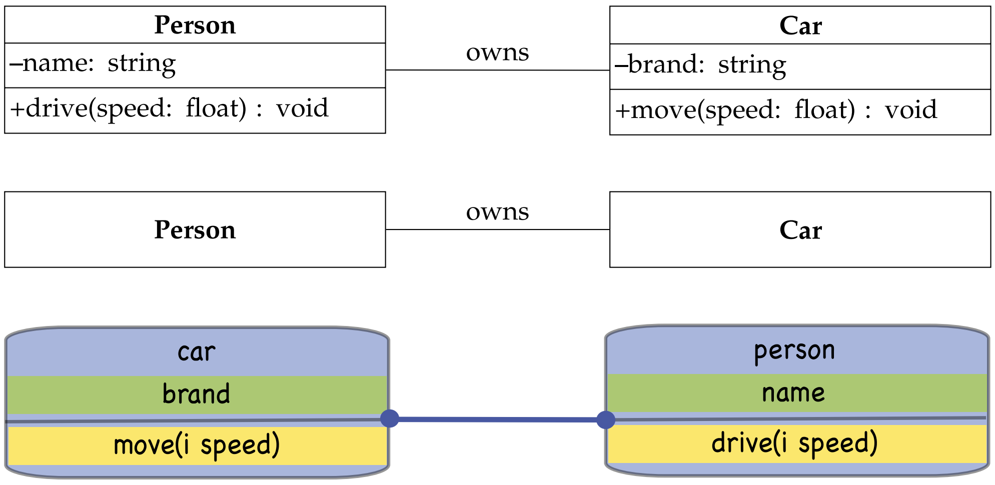


Figure 3.2: *Detailed Class Notation versus Simplified Class Notation and the AoSD version*

3.4.5 Graphical User Interface

Figure 3.3 shows the graphical user interface (GUI) of the game while playing a level. The GUI aims for the analogy of class diagram editors. The GUI consists of the following components:

- *the toolbox* – a collection of draggable items (class, attribute, and method) for making a simple class design. They are represented as crayons. Further, you can put the puzzle in a 'connect mode' with the blue line segment (4th item) to associate classes. In addition, there is an eraser to enable erase mode. By clicking items they can be erased in this mode. For resetting the level there is a curved arrow button. The last item is the exit button represented by a door icon.

  Toolbox items can be hidden by the level designer (e.g. lecturer). It is useful to hide items that are not related to a certain level's goal. In this way levels can be designed in a way that the player is offered a restricted tool set for a better focus on the learning goal.

- *the puzzle area* – a canvas that can contain the draggable items from the toolbox. Also, the game can offer partial designs (unsolved puzzles) in this area to start a level with.

- *the control and data flow area* – two circle shaped buttons for enabling the animation of control - and data flows. The animation of the data flow explains how data is transferred from one object to another object. The control flow animation explains what function is used on an object (e.g. a method call).

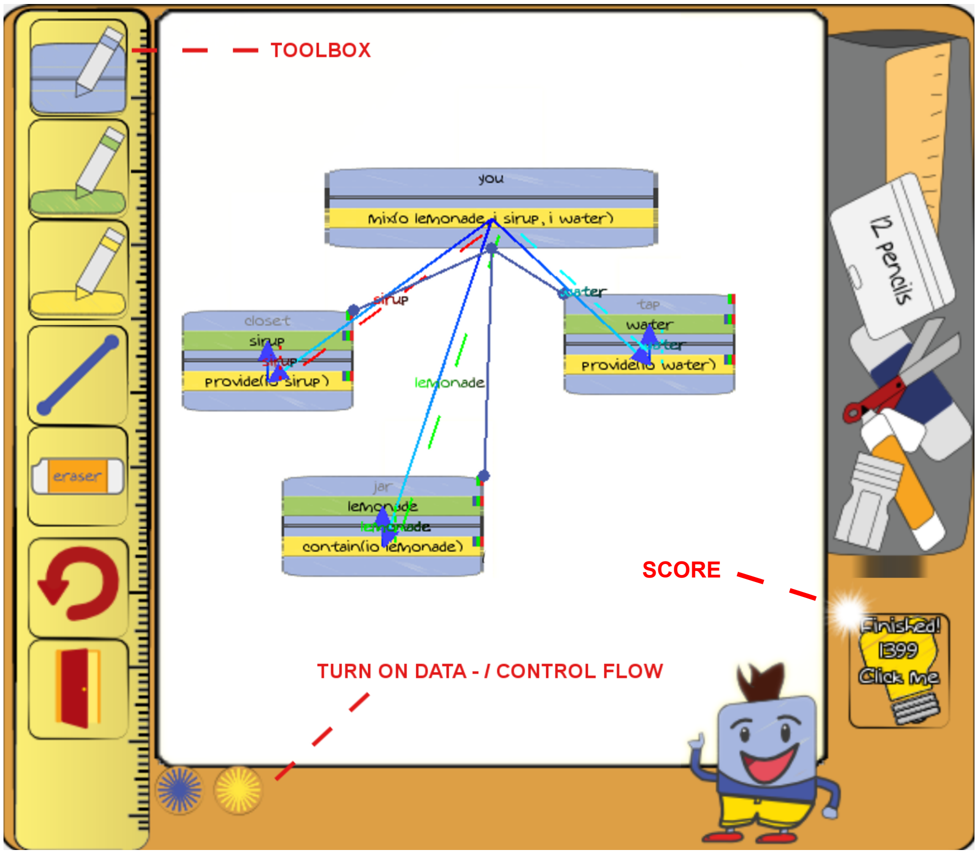


Figure 3.3: *GUI with activated data and control flows. Control flows are represented by an arrow. Data flows are dashed. GUI comments are made in red.*

- *the score indicator* – a light bulb that starts transparent. Whilst making progress the light bulb fills itself with a yellow colour. At the same time the score increases. We chose two indicators. One to give the player feedback about the process. The other one indicates the score of the level, but can vary: some solutions can be a right one, but better than the other.

3.4.6 Game Mechanics

There are several mechanics that we used in order to create an educational and engaging game. In most of the cases the mechanics mentioned below affect both educational and engagement goals.

Direct, Visual and Audio Feedback While feedback is a prerequisite for learning, the game offers feedback through two means:

- each level is scored through an evaluation mechanism supported by audio and visuals. The player receives feedback on the progress because of the intermediate evaluation scores. The player receives confirmational feedback when a solution is a right one.
- the user is given feedback through visualisation of control flow and data flow. To support a player in decision making, the data and control flow of a puzzle can be visualised. This can be seen in Figure 3.3. In this way a player can check the effect of placing an attribute or method in a certain class in comparison with the assignment of the puzzle. We think this is a valuable element of the game from the perspective of self evaluation.

When a player moves or modifies elements, feedback is provided directly and the score adjusts. In addition, sounds are played on user actions (e.g. connecting elements), starting a level and when a level is finished.

Level Unlocking To experience achievement, not all puzzles are playable from the moment the games starts. They have to be unlocked by finishing other puzzles before being able to play that particular one. In this way it is not possible to skip levels. A player needs to learn (by doing) a couple of subjects before they can work on dependent puzzles. By applying the unlock mechanism we aim to create awareness of the dependencies between the learning objectives and the concepts. An example: a player first has to finish the levels about classes and associations before the level about coupling can be played (Figure 3.4).

Choice of Path As mentioned in other research [92] [106] a user is more engaged if the freedom of choice increases. We designed the levels in a way that one can start from different starting points. Even when a player gets stuck he/she is always able to return and take another path.

Multiple Solutions Due to the context of the game, software design, the puzzles do not have one best solution. If a design respects the design principles there still can be a couple of different solutions. A design can be better than another considering the problem domain or other contextual factors. We preserved this real-life situation in the game. This is also expressed with the light bulb - score combination.

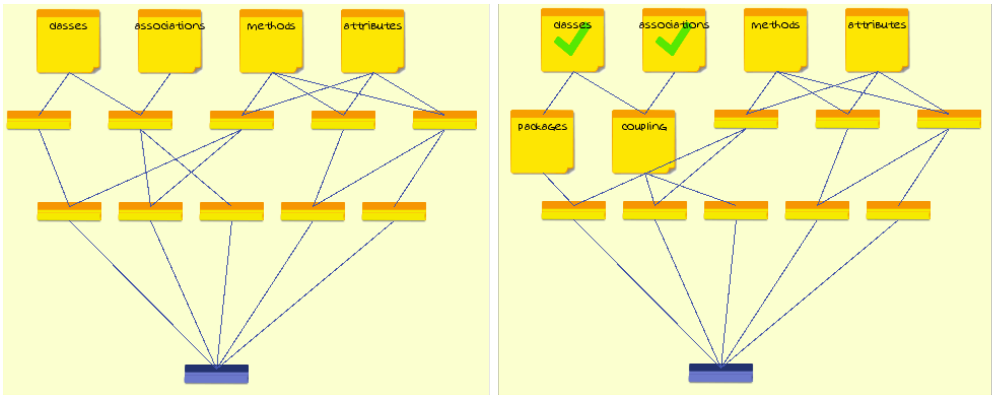


Figure 3.4: *Level unlocking – Left: no levels are unlocked. Right: levels are unlocked because classes and associations levels are completed.*

3.4.7 Scoring Metrics

The player's level score is determined by an evaluation script that scores the puzzle of the level. After every player action this script checks to what degree the created design matches possible solutions based on the design principles: cohesion, coupling, information hiding and modularity. These metrics are used in software design to determine the quality of a software design. The goal of the puzzles is similar: make the design (fragment) as good as possible (high quality). Dependent on the topic of the puzzle one ore more design principle metrics are used. This section explains the calculation of the score in more depth.

Coupling To determine coupling within a puzzle we used a simple approach. We used the CBO (coupling between object classes)[25] metric. This is a commonly used metric in software design. The CBO is a rough metric. There are more precise metrics. However, a more detailed metric would not improve the scoring system for the low to middle complex puzzles the game offers.

CBO per class is: the number of other classes it connects to

Per design the average would be the sum of all the CBO (indexed by *i*) divided by the total of classes (n):

$$averageCBO = \frac{\sum_{i=1}^{n} CBO_i}{n}$$

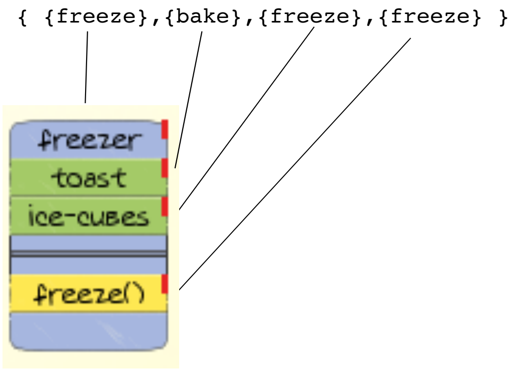


Figure 3.5: *Connected keywords to classes, attributes and methods*

Cohesion Every class, attribute and method has one or more keywords attached to it. We use these keywords to determine if two elements are related, i.e. whether they are cohesive.

The cohesion evaluation script compares if all items of a class (including the class itself) have similar keywords. In the example provided in Figure 3.5 the keyword of the class *freezer* is ***freeze***. The other elements (attributes and methods) also contain keywords: *bake*, *freeze* and *freeze*. In this example only one keyword is attached per element. This can also be two or more.

Cohesion of a class (CC) is determined as follows: i. per comparison (indexed by i), divide the number of keyword matches (m) with all the other elements by the number of keywords (k) that are being considered in the comparison. This gives us the match ratio per comparison. ii. divide the sum of these ratio's by the number of comparisons (n). This gives us the total index of the similarities of a class, the cohesion. The index is a value between 0 and 1. Where 0 means no cohesion and 1 means full cohesion. Below the calculation in formula form:

$$CC = \frac{\sum_{i=1}^{n} \frac{m_i}{k_i}}{n}$$

In the case of Figure 3.5 the CC is determined as follows:

$$CC = \frac{\frac{2}{3} + 0 + \frac{2}{3} + \frac{2}{3}}{4} = 0.5 \text{ (n=4)}$$

Information Hiding and Modularity To evaluate the application of information hiding and modularity we used general design patterns. The player has only a limited number of choices and is guided to a solution that uses this patterns. The evaluation script checks if the user applied the elements (classes, attributes, methods) in a way that matches the pattern.

3.4.8 Software Platform

The game is constructed with Gamemaker 8.1 Standard[1]. We made this choice so we could rapidly make fully functional prototypes. Gamemaker has a readily available engine with graphics, mouse events, scripting and other features that can be used in games.

3.5 The Game

In this section we provide an overview of the flow of a game session. The main aim of the game is to complete every puzzle and get the best score.

'The Art of Software Design' [2,3] starts with an opening screen and gives the player the opportunity to personalise the game by entering the name of the player. This name is used as a saved game and by clicking it, it will resume after the last finished puzzle. After entering the welcome screen a puzzle-tree appears. This tree consists of puzzles based on software design principles or combination of those principles. Players have to unlock upper level puzzles before they can do the next puzzles.

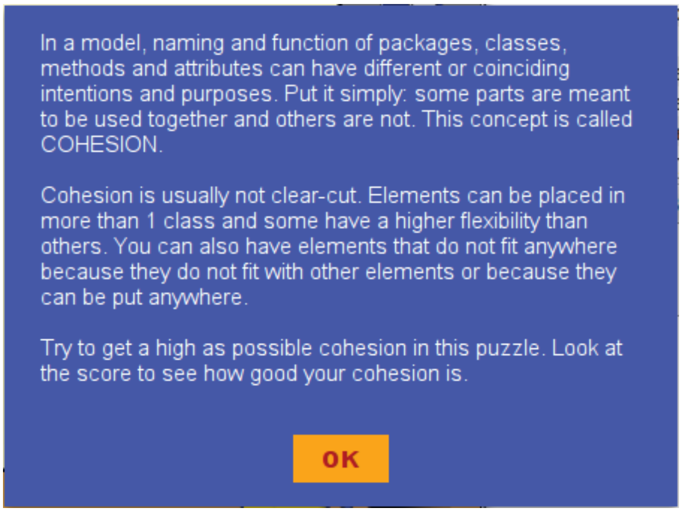
Inside a puzzle a welcome message is shown (as e.g. Figure 3.6). This message contains the assignment of the puzzle. Typical tasks a player has to fulfil are e.g. placing attributes and operations in the right classes (responsibility driven approach), connect the right classes (associate) given a typical design principle. A toolbox is present for adding these items when needed. A progress indicator shows how close the player is to the solution of the puzzle. As mentioned before the game offers the opportunity to complete a puzzle with different solutions. One can be better than the other. This is shown by the players' score as shown in Figure 3.7.

After completing the puzzle, the player returns to the main screen where certain puzzles are unlocked. From there new challenges are possible.

[1] https://yoyogames.com/gamemaker

[2] The Art of Software Design : https://gamejolt.com/games/the-art-of-software-design/21373

[3] Trailer : https://www.youtube.com/watch?v=xn1E2dU-_zg



In a model, naming and function of packages, classes, methods and attributes can have different or coinciding intentions and purposes. Put it simply: some parts are meant to be used together and others are not. This concept is called COHESION.

Cohesion is usually not clear-cut. Elements can be placed in more than 1 class and some have a higher flexibility than others. You can also have elements that do not fit anywhere because they do not fit with other elements or because they can be put anywhere.

Try to get a high as possible cohesion in this puzzle. Look at the score to see how good your cohesion is.

Figure 3.6: *Pop-up with assignment for the cohesion puzzle*

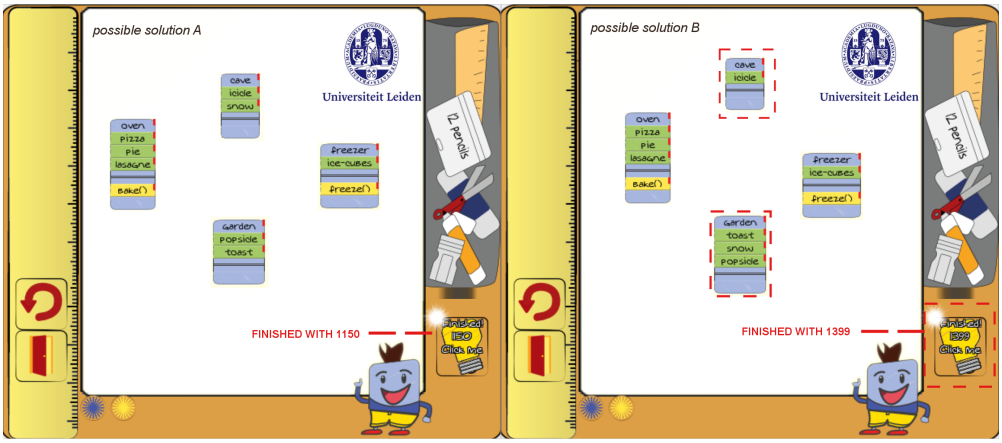


Figure 3.7: *Two possible solutions for the cohesion puzzle. The dashed red boxes show the differences.*

The game ends when all the puzzles about basic design principles have been fulfilled. After that, in future versions of the game, series of more complex puzzles can be offered.

3.6 Evaluation

We evaluated the game in two steps. First we conducted a think aloud session with 6 participants in order to get feedback for improvement. Our aim with these sessions was to: receive feedback on the accessibility of the user interface, finding bugs, checking if all text was read and for checking if the puzzles were understood. We selected 6 students with different backgrounds: from students with no software engineering experience to students who do (e.g. psychology and computer science). From first to fifth year students. Second we addressed a larger group for testing. Players with a software engineering background were approached within our own programs. General players were approached via the discussion website *reddit.com*[4] and via the gaming website *escapistmagazine.com*[5]. We recorded the answers on a survey that consisted of a pre-game and post-game questionnaire. 67 subjects responded to the pre-game questionnaire. 13 (of which 4 subjects were involved in the think aloud session) answered the post-game questions.

3.6.1 Think Aloud Evaluation

Players spent 30 (computer science student) to 90 (medical student) minutes.

Game Mechanics We noticed that giving freedom of choice was of value. When people got stuck on a puzzle, they tried out different solutions or tried another puzzle and returned to the more difficult puzzle later on.

User Interface Players paid more attention to the instructions prior to a puzzle when there were 3 paragraphs of 4 lines of text at maximum. The text was best understood if it had an introductory paragraph, an explanatory paragraph and an assigning paragraph in that order. Some puzzles seem too simple. They were solved without reading the instructions.

[4] https://www.reddit.com
[5] https://www.escapistmagazine.com

Observations For us it was satisfying to see, that after a while subjects started talking about the puzzles in terms of 'classes, methods and associations', instead of 'boxes, blocks and lines', which seems to indicate some unconscious learning.

3.6.2 Survey Group Evaluation

Players spent 35 minutes on average.

67 participants answered the pre-game questionnaire (shown in Table 3.2). Unfortunately only 13 participants answered the post-game questionnaire (shown in Table 3.3). From the 13 post-game participants 10 finished the game. We cannot explain the reason for the low amount of participants in the post-game questionnaire. We assume that participants lost interest after answering the pre-game questionnaire and did also not play the game.

Table 3.2: *Pre-Game Questionnaire Results*

N = 67	
Mean age	23 (min: 18, max: 62)
Working in IT	24
Familiar with classes	33
Familiar with methods	27
Familiar with associations	13
Familiar with coupling	10
Familiar with cohesion	7

3.7 Discussion

In this section we discuss the topics that were introduced in the introduction of this chapter: engagement, learning, interactive feedback for learning, games supporting 'learning by doing' and implementation issues.

3.7.1 Engagement

Supported by the answers on the questionnaire and player observation, we believe that a game can engage students into software design. The accessible platform invites students to start without any prior software design knowledge and gets the player

Table 3.3: *Post-Game Questionnaire Results*

N = 13			
Working in IT	4		
Finished all puzzles	10		
	yes	not	neutral
Enjoyed playing	7	2	4
Liked the graphics	8	3	2
Understood data flow	8	4	1
Understood control flow	7	4	2
Light bulb "feel I did right"	7	2	1
Scoring "feel I did right"	6	5	2
Recall of classes, methods and associations	8	2	-
Recall principles	5	5	-

engaged in learning the basic principles of software design. This positive experience should motivate them to learn about more in depth and specific knowledge.

3.7.2 Learning Yield

We acknowledge that further study is needed to demonstrate the learning effect, but we think the unconscious learning of concepts such as classes, attributes and methods at least indicates a certain learning effect. During the think aloud sessions we observed that our test players adopted the software design terminology and started reasoning about possible solutions using the appropriate terms. Further, most players that participated in the post-questionnaire mention that the understood the concepts of control- and data flow and recall classes, methods and associations. Half of them recall the software design principles.

3.7.3 Interactive Visual Feedback

Several authors emphasise the importance of the presence of feedback mechanisms in educational approaches [75] [49] and games in general [106]. We believe that the game introduces a valuable form of feedback. Next to the progress feedback that tells the player how well he/she performs, AoSD guides the students toward their solutions by supporting them in decision making. The visualisation of the control- and data flow gives students a better understanding of the relationship between the important elements (classes).

3.7.4 Specific Implementation Issues

coupling – Determining the average coupling of the design could be a too rough measure for a part of the quality score and thus the player's score. The average coupling can turn out to be relatively low, while a certain class in the design can have a very high coupling. It may be wise to indicate the user that the coupling of a certain class is too high. In addition having a class that has a very high coupling could lead to a penalty.

cohesion – Although we find the simulation of associative thinking with keywords a very plausible solution for determining cohesion, we have no data that validates that approach. It may be wise to explore correlation between metrics such as (L)COM [46] to validate our keywords method.

3.8 Conclusion

For addressing the challenge to get students motivated for learning software design and keep them engaged, we explored an additional educational approach with the help of an educational puzzle game. In this chapter we explored if a game could support students in learning software design. Although we were not able to completely validate our results, we think that the game *The Art of Software Design* supports students in learning. We offered the players a restricted solution space in which they can be trained in the topic of design principles, the basis of the complex balancing skill a good software designer should eventually master. By practising making design decisions, the players learn about balancing between the different software design quality metrics. Players are challenged by unlocking their levels of increasing complexity by choice of path and have the freedom of solving the puzzles with multiple solutions. An important instrument in the game is the visual feedback system that guides students towards their solution. The preliminary evaluation results support our conclusion.

Current courses on the topic of software design can adopt our game as an enrichment of their didactic approaches for offering students an interactive way in order to keep them engaged. With the help of the level editor lecturers can extend the levels and introduce additional topics of different complexity levels. We see the game as an opportunity to support students' self-assessment possibilities by using the built-in feedback mechanism.

Future research and improvement of this approach. A deeper study of the validity of the score metrics is needed. Further research is needed to demonstrate the learning effect of the game. To demonstrate the learning effect of the game we see a challenge in future research. One potential approach is to use the design skills test that was described in Chapter 2. We suggest to study both validation and learning effect in a case study that uses *The Art of Software Design*.

Chapter 4

WebUML - a UML Class Diagram Editor for Research and Online Education

In this chapter we discuss our web based educational UML class diagram editor: WebUML. WebUML was developed in order to support the study of i) students' design strategies ii) automatic evaluation of students' software designs iii) guidance of students during their software design activities. WebUML was used during multiple experiments. In this chapter we present an overview of the tool, discuss experiences and future extensions.

4.1 Introduction

Our research aims to observe and analyse students' software design activities. In order to gain knowledge about matters such as strategies or common difficulties we typically designed experiments with classroom settings in which students perform a design task with the use of a UML editor.

Using UML modelling tools often involves installing, configuring and learning to use the tool. What tool(s) to use is an ongoing discussion between educators [2]. Educators discuss the following issues when (considering the) use of tools:

- Tools should be designed specifically for pedagogy.
- There should be enough support available for the tool, either online or facilitated locally by the university.
- Tools can be used to encourage students to produce “good” models.
- Modelling tools could measure the quality of students’ software designs.
- Tools are often very difficult to install.
- Students get distracted by the complexity of tools.

For the development of WebUML we wanted to be able to address the points of discussion mentioned above, but also being able to use the tool as a research instrument. This means we want to measure and log every user action; something existing tools typically don’t allow you to do. There were three main objectives for the development of WebUML.

First, in our research there is a growing need to upscale experiments in order to address a larger amount of student subjects to observe and being able to do statistical analysis.

Second, we wanted to have a system without the need of installing software.

Third, the tool should be equipped with a system that eventually can give students feedback on the quality of their model and guide them through the process of software design modelling.

We formulated the following research questions:

- RQ1: how can we gather data about student software design activities on a large scale in order to perform statistical analysis?
- RQ2: how can we offer a UML editor that is not complex (easy to understand) that serves a pedagogical context?
- RQ3: how can we provide students the appropriate feedback during their design activities?

In order to answer the research questions we proposed that WebUML should meet the following requirements:

- the ability of drawing basic class diagrams, including packages.

- the ability of giving feedback during the making of, and after completion of a software design assignment.
- the ability of uploading assignment solutions.
- the ability of logging user actions in a comma separated file.
- The tool should be able to import and export files for exchanging diagrams with other tools.
- contains a common graphical interface.
- should be web-based and support off-line use.

Because we mainly focus on structured software design in our research, we choose to only implement class diagrams.

The remainder of this chapter is organised as follows: first we discuss related work in Section 4.2, second we present an overview of WebUML in section 4.3. We discuss WebUML in Section 4.5. At last we conclude and discuss future extensions in 4.6.

4.2 Related Work

Several researchers have proposed specific educational tools for software design. In this section we discuss the ones we explored during the research of this dissertation. In this section we present industrial tools and tools that were developed during research on educational tools. The tools listed in this section are distilled from related work and from discussions about the use of tools in education, in particular in a workshop session at the Educators' Symposium in 2013 [2].

4.2.1 Industrial UML tools

Most universities use industrial UML tools in teaching, this is a list of often used (UML)modelling tools:

- Visual Paradigm - Visual paradigm is a broad development tool that supports process modelling, ERD modelling and UML modelling. It also has project management features and users can build wireframes and story boards for previewing ideas for future systems. To explore possible paths in a model it has an option for animation. (source: `https://www.visual-paradigm.com`)

- IBM Rational Rhapsody - Rational Rhapsody supports SysML and UML and domain specific languages (DSL). It can generate code. From model validation there is the option to animate models. (source: `https://www.ibm.com/support/knowledgecenter/SSB2MU/rhapsody_family_welcome.html`)
- Bridgepoint - Bridgepoint is a UML editor that is capable of executing models and contains a set of model compilers (providing translation). It should support the model driven development approach. (source: `https://xtuml.org`)
- Magic Draw UML - Magic Draw claims to be 'Ease of use' and to have a short learning period. It supports code generation. (source: `https://www.nomagic.com/products/magicdraw`)
- StarUML - It started as an open source tool (StarUML 1). StarUML 2 supports UML and ERD modelling. It generates code. It is now a commercial tool. (source: `https://staruml.io/`)

4.2.2 Research and Educational UML tools

Because of our in interest in research tools we compare the tools listed below in Table 4.1. Below we describe the considered tools shortly:

- QuickUML, by Crahen et al. [29] offers a small and simple subset of UML. The authors claim most UML tools overwhelm students that are just learning UML for the first time. According to the authors, the Java based tool supports iterative design because of the possibility to switch between design- and source code view.
- CoLeMo, by Chen et al., is a collaborative learning environment for UML [24]. CoLeMo uses the concept of pedagogical agents [59], autonomous software components that support learning. The agents are capable of providing the student feedback on their collaboration process and advice them on UML modelling. All activities and chat messages performed by the student are logged by the tool.
- Ramollari et al. present StudentUML [109]. Their tool is tailored for education with focus on simplicity, correctness and consistency. The authors claim the tool supports the process the students go through by providing feedback on diagram checks and by offering automatic repairs.
- Baghaei et al. present COLLECT-UML [11] an educational tool for learning UML class diagrams that supports collaborative and individual learning. COLLECT-UML has a submission system that is evaluated by a pedagogical module. The module provides feedback about the submission on the syntax level as well as on the semantic level.

- Soler et al. [124] propose a web based UML class diagram editor as part of their ACME e-learning framework. The tool can check a student's solution and provides feedback messages with suggestions for improvement.

- Hiya et al. present Cloocal, a web based tool for domain specific modelling [55]. It offers a model compiler for generating source code.

- Ma et al. developed ASE [84], a web based modelling tool designed for touch screen devices. It is developed to be cross platform. With the growing application of web browsers as application platform the authors saw the need to develop a modelling tool that is web based. The tool handles different touch commands and specific gestures which uses a whole new human-machine interaction mode.

- Lethbridge's Umple [78] is a tool that supports programming and UML modelling in parallel. It offers feedback to users to guide them away from errors and let students correct them.

- Jolak et al. present their tool OctoUML [60]. The tool is capable to recognise hand drawn sketches and turns them into formal UML notation. Users can mix formal and informal notation at the same time on the same canvas. It can handle multiple users on a single input device which supports collaborative modelling.

Tool	Web based	Desktop	Feedback	Collab.	Logging	Active
QuickUML	no	yes	no	no	no	unknown
CoLeMo	no	yes	no	yes	yes	unknown
StudentUML	no	yes	yes	no	no	unknown
COLLECT UML	no	yes	yes	yes	yes	unknown
ACME	yes	yes	yes	no	no	unknown
Cloca	yes	yes	no	no	no	yes
ASE	yes	no	no	no	no	unknown
Umple	yes	yes	yes	no	no	yes
OctoUML	no	yes	no	yes	yes	yes
WebUML	yes	yes	yes	no	yes	yes

Table 4.1: *overview of UML editors for educational and/or research purposes*

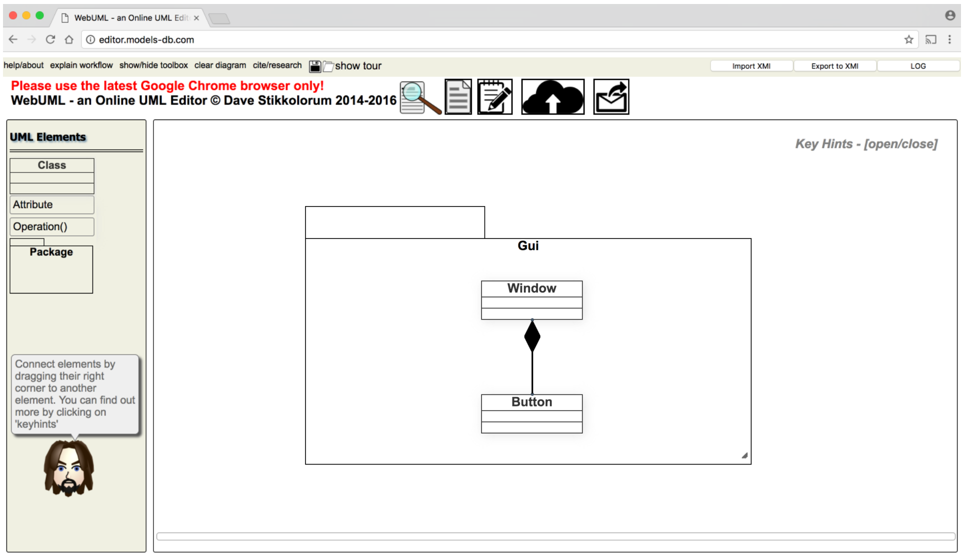

WebUML - an Online UML Edit
editor.models-db.com
help/about explain workflow show/hide toolbox clear diagram cite/research show tour
Import XMI
Export to XMI
LOG
Please use the latest Google Chrome browser only!
WebUML - an Online UML Editor © Dave Stikkolorum 2014-2016
UML Elements
Class
Attribute
Operation()
Package
Connect elements by dragging their right corner to another element. You can find out more by clicking on 'keyhints'
Key Hints - [open/close]
Gui
Window
Button

Figure 4.1: *WebUML in action*

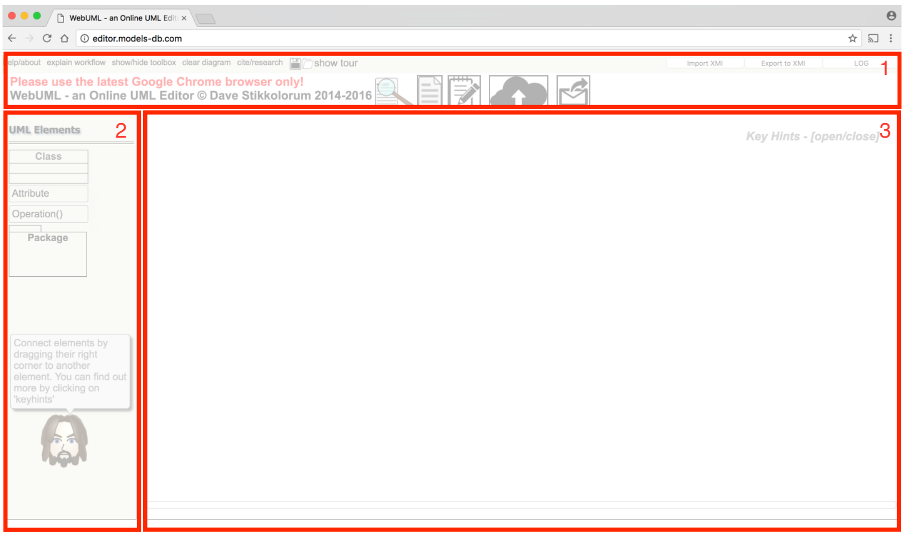

WebUML - an Online UML Edit
editor.models-db.com
elp/about explain workflow show/hide toolbox clear diagram cite/research show tour
Import XMI
Export to XMI
LOG
1
Please use the latest Google Chrome browser only!
WebUML - an Online UML Editor © Dave Stikkolorum 2014-2016
UML Elements
2
Class
Attribute
Operation()
Package
Connect elements by dragging their right corner to another element. You can find out more by clicking on 'keyhints'
Key Hints - [open/close]
3

Figure 4.2: *WebUML's divisions*

4.3 Overview of WebUML

In this section we present an overview of the WebUML class diagram editor. We explain which functions we included to fulfil our research and education needs.

Figure 4.1 shows a screenshot of WebUML during a class diagram modelling session. Basically WebUML can be divided into 4 area's (shown in Figure 4.2): 1) navigation and control, 2) the toolbox, 3) feedback agent and 4) the drawing canvas. This section discusses WebUML's most important features organised per area.

4.3.1 Area 1: Main Control and Navigation

Area 1 is meant for the main navigation of the application and houses launcher icons for different kinds of services. We discuss the most important ones.

- *Import and Export XMI Button*: WebUML is capable of importing class diagrams that are described in the XMI[1] format. Through a dialogue it is possible to paste XMI code and import it into the editor. The canvas (area 4) will show a class diagram of the imported code. When exporting to XMI, WebUML downloads the current diagram to the users computer packed in a XML file [2].
- *Export Log Button*: Users' modelling activities are logged during a session with WebUML. It is possible to download this log as a comma separated file. For further reading about the logging system, see 4.3.5.
- *Evaluate Button*: This button activates the feedback agent. The result of the agent is displayed in the text balloon of the avatar (area 3) on the left bottom of the screen.
- *Mode Button*: Switches between reading an assignment text and modelling.
- *Register Question Button* Users' questions during modelling can be recorded by clicking a button. The remarks and/or questions will be added to the log.
- *Upload Assignment Button*: When users use the upload possibility, a zip file is generated and uploaded to a server. For own use the same zip file is downloaded to the users computer.

[1]The xmi specification is found at: `https://www.omg.org/spec/XMI`

[2]XMI is a specific application of XML

4.3.2 Area 2: Toolbox and Feedback Avatar

Area 2 consists of the standard UML elements Class, Attribute, Operation and Package. They can be dragged onto the canvas (area 4). We choose to draw associations between classes by drawing lines directly from the dropped classes instead of offering a association icon in the toolbox.

4.3.3 Area 3: Feedback Agent

The avatar that represents the feedback agent can be used to communicate to the user. It has a text balloon that can be filled with messages for giving hints or evaluation feedback. Chapter 10 discusses the feedback agent.

4.3.4 Area 4: Drawing Canvas

Area 4 is the drawing canvas. All dropped elements appear here. The user is able to move UML elements, associate classes and, change the types of the associations, adding association names and adding multiplicity elements. Element names can be edited and elements can be deleted.

4.3.5 General Services

WebUML provides two core services: logging user actions and providing (textual) feedback. In this section we discuss both.

- *Logging of Students' Design Activities* WebUML is equipped with an automatic logging service. The service logs different kinds of activities of students during their design sessions. Currently we log the following activities for UML elements: creation and removal, movement of the element on the canvas and changing names. Other activities that are logged are: submission of assignment, switching from reading to modelling and vice versa. All log items are timestamped using Unix time [3] (also known as POSIX time or epoch time).

  An activity log can be exported to a comma separated file in order to import the data into applications that are capable of visualising the data or support statistical analysis.

[3] https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

With the analysis of the logs we aim to gain knowledge about typical design strategies, measure design time, identify design decisions, measure corrections and discover unknown patterns of student behaviour.

- *Feedback and Diagram Evaluation* WebUML aims to develop to a powerful educational tools that is capable to guide students through the process of software design and is able to evaluate students' models. Currently WebUML is equipped with a feedback agent that is capable of evaluating a diagram based on an example solution. The feedback agent is inspired by the concept of pedagogical agents [123, 85]. Agents, often used in games, are autonomous entities that have different behaviour based on events from the environment. Pedagogical agents react on actions of learners and/or the state of their products - in our case, diagrams - with the aim to support their learning process.

4.4 WebUML in research activities

We used WebUML in several experiments and classroom research. In this section we briefly describe the aim of the research and the role of WebUML in this research. The different research topics are explained in more detail in the following chapters in this dissertation.

4.4.1 WebUML and LogViz

In [136] we used WebUML in combination with another tool: LogViz [107] in order to investigate different strategies that students use for solving software design tasks. With the help of visualising WebUML's log files we were able to discover the dominant breadth-first and depth-first strategies. More about this research can be read in Chapter 5

4.4.2 WebUML as online tool

In [137] we were able to scale up the classroom setting by conducting an online experiment with 120 student pairs. Because of the web-based character of WebUML we were able to address a larger group of students. More about this research can be read in Chapter 6

In this research we also enabled and explored the assignment submission feature of

WebUML and the possibility to ask questions via an online form. More about this research can be read in Chapter 6

4.4.3 Teaching Agile Modelling

In this research we use WebUML as tool for doing an incremental analysis and design assignment that would fit in an agile development process. We asked student pairs to reflect on each other's challenges. More about this research can be read in Chapter 7

4.4.4 Automatic Grading and Feedback

We explored the possibilities of automatic grading and giving feedback to the students while making a design. Preliminary results are discussed in Chapters 9 and 10.

4.5 Discussion

Based on the experience with the use of WebUML in different experimental settings we answer and discuss the research questions in this section.

4.5.1 RQ1: how can we gather data from student software design activities on a large scale in order to perform statistical analysis?

We were able to equip WebUML with a logging function that produces interchangeable log data. The data was analysed in two experiments and statistical analysis was possible. Different activities such as movement of UML elements or edits of names can be logged. We were able to recognise different strategies students use to deliver a software design. The online mode enable us to address and research larger groups easily.

4.5.2 RQ2: how can we offer a UML editor that is not complex (easy to understand) that serves a pedagogical context?

WebUML is less complex than the industrial software design tools. It contains a slimmed down amount of functions, focused on relevance to the learning task.

By choosing for a web-based architecture WebUML does not need to be installed on the operating system of the students' computer. The web-based approach also has the benefit students can use any operating system they want. WebUML aims to be cross-platform, although, at the moment of writing this dissertation Google's Chrome browser seems to best support our tool. Chrome can be installed on all popular desktop operating systems (Windows, MacOS, Linux).

4.5.3 RQ3: how can we provide students the appropriate feedback during their design activities?

We explored the possibility of using a module that is based on the concept of pedagogical agents. The first pilot study shows encouraging results and suggest further research on this topic (further explained in Chapter 10).

4.5.4 Limitations

Currently WebUML only serves the students to create class diagrams.

4.6 Conclusion and Future Work

In this chapter we presented our educational web-based UML class diagram editor. WebUML was created because of the need of a UML tool that could be used as an educational tool as well as a research tool. The web-based architecture combined with the logging system enabled larger scale classroom studies that could be analysed with statistical analyses (RQ1). To avoid the complexity of most industrial tools we implemented a tool that does not require installation and uses a small subset of UML (RQ2). We were able to use the tool in our research in order to collect data about student activities during class diagram analysis and design. In pilot studies we used the feedback system that was implemented in WebUML in order to investigate the value of providing guidance during modelling activities (RQ3).

At the moment of writing this dissertation WebUML2 (version 2) is being developed. WebUML2 is going to serve a wider range of UML diagrams, to start with sequence diagrams and use case diagrams. Also the following functions are being added and researched:

- Replay - the replay function enables the user to replay the construction of a model. In this way a lecturer can discuss possible errors they made in previous steps. At the same time it helps with students' self-reflection as well. Students can continue their work from the point they made the mistakes.

- Collaborative modelling - this extension is meant for user to collaborate and work on a model together. The collaboration is threefold: i) the users can work on the model at the same time, the model synchronises on the peers' screen when a change is made ii) the users can chat together via a text chat window iii) a video chat window enables users to have a live conversation about their progress.
- Feedback agent - the improved version of the feedback agent is guiding the student through a software design task. The agent is going to be configurable. The students should be able to decide on what items they want to receive feedback.
- Assignment mode - students will have the possibility to work on assignments provided by WebUML. Assignments can be uploaded (which was already part of WebUML version 1), but also receive direct feedback on the quality of their work.

The above mentioned functions should be tested in experimental settings in the future. We aim to have university software design programs involved to understand learners needs even better.

Part IV

Student Guidance and Feedback

Chapter 5

Strategies of Students Performing Design Tasks

This chapter shows the most common strategies students use to create class designs. We demonstrate our approach of logging students' modelling activities while doing a software design task. We developed our own online modelling editor 'WebUML' and visualisation tool 'LogViz' for the logging and interpretation of the log files. As follow-up, students filled-out a brief questionnaire about their time spent and difficulties in performing the task. The results show that the students use different strategies for solving the tasks. We divided these strategies into four main categories: Depthless, Depth First, Breadth First and Ad Hoc. Our results show that the Depth First strategy supports students to deliver models with a better layout and richness (detail). The questionnaire shows that students find that choosing the appropriate UML elements is a difficult and time consuming task. We want to use our insights to improve our educational programs and tools. In the future we want to test WebUML and LogViz in larger educational contexts.

This chapter is based on the following publication: Dave R. Stikkolorum, Truong Ho-Quang, and Michel R.V. Chaudron. Revealing students' UML class diagram modelling strategies with WebUML and LogViz. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*, pages 275–279. IEEE, 2015

5.1 Introduction

Lecturers are familiar with the fact that students learn in different ways. These learning styles are categorised by authors in various ways, such as: being a 'do-er','dreamer', a 'thinker', a 'decision maker' [67]. Others use: being 'meaning-oriented', 'reproduction-oriented', 'application-oriented' or 'non-oriented' [154]. By acknowledging the fact that student groups consist of a mixture of representatives of these styles, lecturers can organise their education to serve this combination of different learning styles. Especially in classroom settings, that usually consist of students with different styles, this is of course difficult to achieve, while outside the lecture rooms students would choose strategies that fit their style better.

In software design courses students deal with problem solving challenges. A substantial number of students have difficulties with software design tasks. It is plausible that, based on their learning style, students will use different strategies to solve their software design problems.

By understanding students' difficulties in designing software, we can develop better teaching methods and tools. In order to understand what type of problems students have during modelling tasks we monitored students' modelling activities during an average UML class diagram assignment. To analyse students' activities while working in these tasks, we developed an online UML class diagram editor that can log the actions students perform during a modelling task. In addition, we developed a tool that is capable of visualising the different actions a student performs, which makes the analysis easier and enables us to discover patterns that are likely more difficult to see without the visualisation.

In this chapter we answer the following research questions:

- RQ1: *Can we identify strategies that students use for creating software designs?* We assume students approach their problem-solving strategies in different ways. We asked ourselves if it is possible to identify and categorise students' strategies.
- RQ2: *Can we learn which steps students find difficult in creating designs?* From experience in the classroom we learned there is a substantial part of students in a group that copes with difficulties in software design. We are interested in verifying/assessing whether it is possible to identify typical steps a student makes that are related to his/her difficulties.
- RQ3: *Do students that perform better on modelling tasks use different strategies than those who perform worse?* Is it possible to identify strategies that are more successful than others?

The contribution of this research is a description and/or classification of students' modelling behaviour revealed by our tools : WebUML (online UML class diagram editor) and LogViz (visual log analyser).

When we use the term 'model' in this chapter we mean that we explore modelling behaviour of just one view of the model: the structured view, the class diagram.

The remainder of this chapter is organised as follows: in Section 5.2 we explore related work. In Section 5.3 explains the method we used and includes the most important features of both tools. We show our results and discuss them in Sections 5.4, 5.5 and 5.6. We conclude and suggest future work in Section 5.7.

5.2 Related Work

Avouris et al. mention the value of logging students' learning activities and argue that logging is not enough [9]. They propose a combination of log info and video observations or snapshots in order to reveal patterns the log files do not show. In the same workshop Gibert-Darras et al. propose a set of design patterns for tracking students' problem solving performance [40]. They show patterns related to automated grading and the analysis of student answers. The patterns, found as result of answer analysis, do not reveal hidden learning strategies but can be used for monitoring students' learning. Blikstein reports on the use of a logging module for programming tasks [18] and identified student strategies that could help lecturers identify student problems in an early stage of the task.

Ko et al. conducted a study in which they used the combination of a logging file and visual interpretation tool to analyse the behaviour of software developers during a maintenance task [66]. They successfully identified different strategies the developers used.

Westera et al. found predictors in the logs of their serious games in the field of consultancy for learning [160].

Claes, Pingerra et al. [27][103] logged students' events during business modelling sessions. They used visual analysis and found different styles and related them to model quality.

Agarwal and Sinha conducted an empirical study [1] on the usability of UML. Novice developers rated UML low. Some difficulties with class diagrams were identified.

We are not aware of cases in the specific task of software modelling where students'

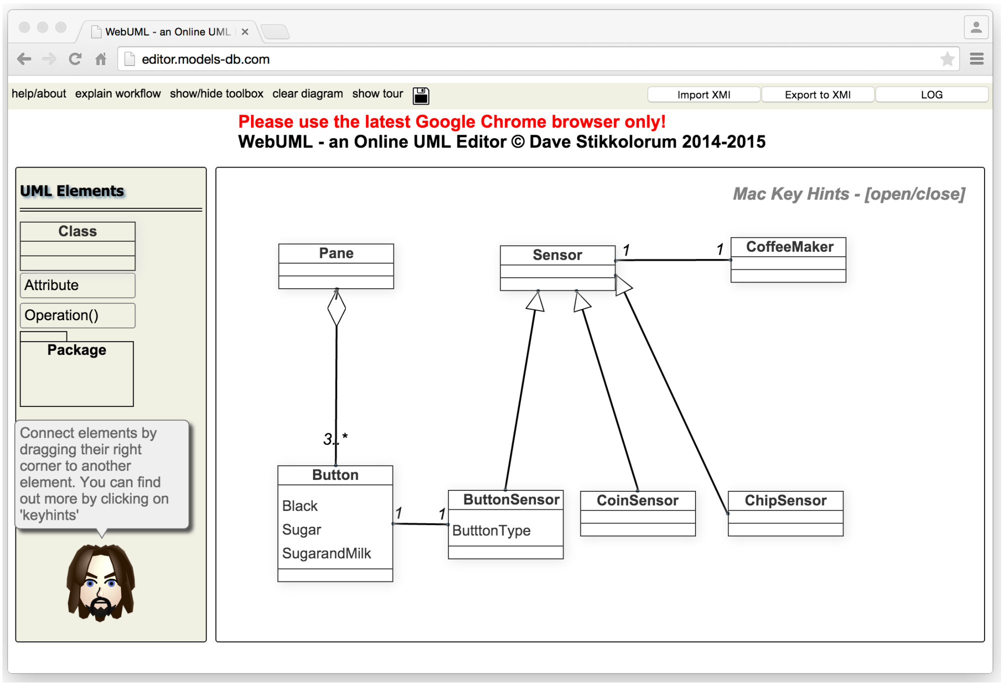

WebUML - an Online UML
editor.models-db.com
help/about explain workflow show/hide toolbox clear diagram show tour
Import XMI
Export to XMI
LOG
Please use the latest Google Chrome browser only!
WebUML - an Online UML Editor © Dave Stikkolorum 2014-2015
UML Elements
Class
Attribute
Operation()
Package
Connect elements by dragging their right corner to another element. You can find out more by clicking on 'keyhints'
Mac Key Hints - [open/close]
Pane
Sensor
CoffeeMaker
1
1
3..*
Button
Black
Sugar
SugarandMilk
1
1
ButtonSensor
ButttonType
CoinSensor
ChipSensor

Figure 5.1: *Student's work in progress using the online editor*

logs were analysed to find task strategies.

5.3 Method

In this section we explain the method we used. We show the participant group, explain the modelling tool and the log format it delivers. We show the features of the visualisation tool that is capable of filtering the modelling tool's log files. We conclude with explaining the follow-up questionnaire that we conducted.

5.3.1 Participants

We observed 20 bachelor students, studying software engineering, during their 2nd semester course (Analysis and Design). The experiment was conducted in the third week of the course. The students were not experienced with UML or software design. They just learned the theoretical basics of class diagrams. Prior to the course the students followed a course on Java programming.

a Coffee Machine A coffee machine can make different types of coffee (one at a time): black, with sugar, with sugar and milk. The different types of coffee are poured into a cup. The front of the machine contains a pane with buttons. The buttons are used for selecting the desired drink. One can pay with a chip card or coins. The chipcard is read by a chip sensor and the coins by a coin sensor.

Figure 5.2: *Coffee Machine Modelling Assignment*

5.3.2 The Task / Assignment

We asked each student to use our tool 'WebUML' and to model a class diagram of a coffee machine. The short (76 words) case text was digitally offered to the students (Figure 5.2 / Appendix B). The students had to extract the details from the text for delivering a basic class diagram using classes, attributes, operations, association, aggregation and inheritance. There was no time constraint.

5.3.3 Online Modelling Editor

Because we wanted to have a simplistic UML editor with a minimum subset of the UML and the possibility to embed it in different online education software environments, such as serious games or gamified courses (e.g. specific MOOCs), we developed our own modelling tool : WebUML[1] (Figure 5.1, see also Chapter 4). At the moment of writing WebUML is only targeted at UML class diagrams equipped with the most used relationships, attributes, operations, labelling and multiplicity elements. The editor supports XMI import and export. The editor is capable of logging students' activities while doing modelling activities. Specifically for UML class diagrams we categorised the following activities for logging:

- CREATE - creation of UML elements, such as Class, Operation, Attribute, Association etc.
- MOVE - the movement of already created elements to another location in the diagram.
- SET - set value of attributes of elements e.g. names of classes, attributes etc.

[1]WebUML - `https://webuml.drstikko.nl`

- REMOVE - removal of an element from the diagram.

All items are logged with a unix timestamp[2]. For simplicity reasons we chose to save the log file in a comma separated file. For example CREATE :

```
<TIME>,CREATE,<UML_ELEMENT_TYPE>,<UML_ELEMENT_ID>, <UML_ELEMENT_NAME>,<CONTAINER>

Log fragment: 1423143012689, CREATE, CLASS, cl_1_1, Class, Diagram
```

5.3.4 Visualisation Tool

The visualisation tool, LogViz[3] is capable of reading one or more WebUML log files. It can filter on activity and UML element level. Figure 5.3 shows the main screen of the tool. Area 1 contains the controlling buttons of the tool; area 2 consists of the log's visualisation properties such as activity colours, symbols of UML elements, etc; area 3 is the most essential component of the visualiser where graphs and activity points are shown. Logging activities and UML elements are represented by different colours and symbols.

5.3.5 Follow-up Questionnaire

Our editor (WebUML) only records students' direct usage of the tool, not what students do when they are not using it (e.g. thinking about the modelling task, discussing). In order to anticipate on missing this data, a brief questionnaire was conducted. The questionnaire consists of 11 questions. We asked students to estimate the total time they spent on the assignment, time spent on different (sub)activities, and steps that they found difficult. Four of the questions are aimed at tool-feature development. We did not include them in the research that this chapter describes.

5.4 Results

In this section we present the different results: i) the statistics derived from the online modelling editor's log files; ii) the identified strategies of the students' modelling

[2]Unix Timestamp - https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

[3]LogViz - https://gitlab.com/truonghoquang/LogVisualizer

Table 5.1: *Overview of measured modelling activities*

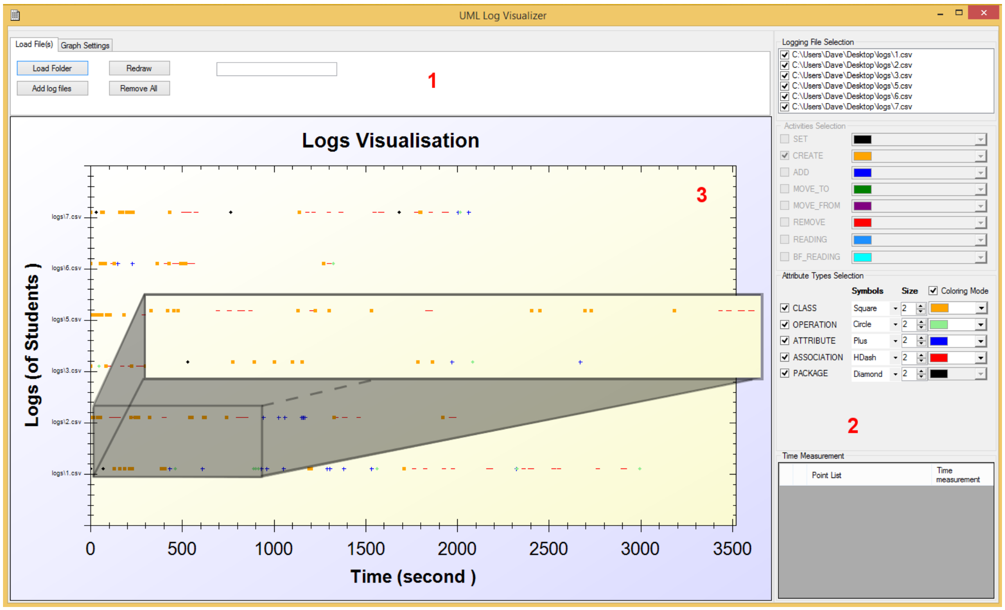
Student	Time (secs)	CREATE	MOVE	SET	ADD	REMOVE
1	395	11	2	11	0	0
2	1709	7	15	15	0	0
3	3580	29	115	40	5	4
4	2111	41	17	51	12	6
5	1688	50	73	78	23	6
6	703	30	2	14	0	0
7	1334	33	16	17	2	1
8	5579	93	150	43	28	2
9	225	9	18	0	0	0
10	2099	40	110	20	3	9
11	2299	59	210	55	41	10
12	3003	51	100	28	20	2
13	5607	96	270	71	40	6
14	1603	70	24	57	37	6
15	3123	28	40	21	14	6
16	1400	46	96	18	2	13
17	11357	59	360	16	4	12
18	689	22	54	12	2	3
19	2618	73	80	34	0	15
20	2087	48	49	26	7	3

task we analysed using the visualisation tool and iii) the response on the follow-up questionnaire.

5.4.1 Statistics

Table 5.1 shows the measurements of the different activities we logged. Each row in the table represents the log of the assignment task of one student. The numbers in columns 'CREATE' to 'REMOVE' represent the amount of occurrences of this particular activity during the performance of the assignment. The columns Time and MOVE show a large spread: Time (M = 2660.37, SD = 2515.05), MOVE (M = 90.05, SD = 95.09).

5.4.2 Visualisation



Logging File Selection

- ✓ C:\Users\Dave\Desktop\logs\1.csv
- ✓ C:\Users\Dave\Desktop\logs\2.csv
- ✓ C:\Users\Dave\Desktop\logs\3.csv
- ✓ C:\Users\Dave\Desktop\logs\5.csv
- ✓ C:\Users\Dave\Desktop\logs\6.csv
- ✓ C:\Users\Dave\Desktop\logs\7.csv

Activities Selection

☐ SET	⬛
✓ CREATE	🟧
☐ ADD	🟦
☐ MOVE_TO	🟩
☐ MOVE_FROM	🟪
☐ REMOVE	🟥
☐ READING	🟦
☐ BF_READING	🟦

Attribute Types Selection

	Symbols	Size	✓ Coloring Mode
✓ CLASS	Square	2	🟧
✓ OPERATION	Circle	2	🟩
✓ ATTRIBUTE	Plus	2	🟦
✓ ASSOCIATION	HDash	2	🟥
✓ PACKAGE	Diamond	2	⬛

Figure 5.3: *LogViz visualisation tool*

Since the task was a 'creating' task we filtered the logs' CREATE activities. Figure 5.3 shows a 'zoom in' of two student logs. As explained in Section 5.3.3, actions are logged in time. Figure 5.3 shows that the upper student associates (horizontal dash) classes early in the task, while the other first identifies attributes and associates later.

We examined all the log files by analysing the students' creation steps (create class, create attribute, create operation, create association) and identifying creation patterns by looking at the, in time, clustered creation actions. We identified 4 different main strategies (Table 5.2) and variations on these strategies. We indexed them 1 - 4 and are shown in Table 5.3. The last column in that table shows the amount of occurrences in the observed student group.

Explanation of the main strategies:

1. **Depth-less Strategy** - students don't add detail in the form of operations and/or attributes to come to a complete diagram.

2. **Depth First Strategy** - students, after they create classes, first add detail in the form of operations and/or attributes, then associate the classes.

3. **Breadth First Strategy** - students, after they create classes, first associate the classes, then add detail in the form of operations and/or attributes.

4. **Ad Hoc Strategy** - students don't use any form of structured approach to solve a class diagram task.

Table 5.2: *Class design strategy types*

1	Depthless Strategy	class → associate
2	Depth First Strategy	class → add detail → associate
3	Breadth First Strategy	class → associate → add detail
4	Ad Hoc Strategy	no structured approach

Table 5.3: *Different strategies students use while making a class diagram*

Strategy Index	Strategy Description	Occurrences
1	Create all classes → associate classes	3
1A	Loop (create classes→ associate classes)	1
2	Create classes → add operations and attributes → associate classes	
2A	Loop (Create classes → add operations and attributes → associate classes)	4
2B	Loop (Create classes → add operations and attributes) → associate classes	4
2C	Loop (Create classes) → Loop (add operations and attributes) → associate classes	2
3	Create classes → associate classes → add operations and attributes	1
3A	Loop (Create classes → associate classes → add operations and attributes)	1
3B	Loop (Create classes → associate classes) → Loop (operations and attributes)	3
4	No clear strategy: mix of classes, attributes, operations in random-like order	1

5.4.3 Follow-up Questionnaire

From the survey that was conducted after the assignment we received 8 responses. Although not all students responded, we show their answers in Table 5.4. It shows that the time spent on the task varies among the students (from 30 - 280 minutes). Students spent their time in three main activities: i) understanding the assignment and the related domains; ii) making modelling decisions; iii) usage of the tool. The responses indicate understanding and making decisions are the most time-consuming tasks. 5 out of 8 respondents spent more than 50 percent of the assignment time on understanding the assignment or making decisions on their models.

Half of the students reported doing the task continuously, while the other half inserted breaks for thinking over the case or looking for supporting information in books or online.

Most students found it difficult to decide class, attribute and association type (5 out of 8). 5 out of 8 observed students mentioned that they changed their diagrams layout to make them more reasonable.

5.5 Discussion

In this section we discuss the research questions we listed in Section 5.1. We do this based on our interpretation of the results presented in Section 5.4

5.5.1 RQ1: Can we learn strategies that students use for creating software designs?

From the visualisations we clearly found three types of strategies (summarised in Table 5.2) that students use for developing their diagrams. We also identified a fourth type which one could name a rest group (Ad Hoc). We could not determine a logic strategy in that case. We only identified one person for this category. The most interesting is the difference between the Depth First and Breadth First strategies where students differ in the moment they add attributes and/or operations in their class diagrams.

Table 5.4: *Summary of the reponses on the follow-up questionnaire*

ID	U (%)	M (%)	D (%)	O (%)	Time	Cont.	Difficulties	Revs	Revision Type
1	25	25	50	0	30	1		3	Normally I re-check my designs to be aligned.
3	29	33	33	5	60	1	Attributes and operations.	3	Organise classes.
4	54	18	29	0	280	0	How to split each element up into either a class or have it as an attribute.	3	Association should not be dependency
7	50	33	17	0	60	1	Whether to use packages or not, and about what to include in the model.	20	Mostly from scratch.
8	33	33	33	0	60	1	The payment and make coffee part were difficult to make.	2	Basically on design to make it more reasonable.
9	30	50	20	0	60	0		3	Moving things around trying to make the model cleaner.
10	20	50	30	0	50	0	How to connect the diagram between the classes. If there should be attributes/methods involved or not.	20	Re-making whole diagram, new classes, add/remove attributes and methods
16	10	60	30	0	30	0	Deciding what UML associations to use was difficult for me.	3	UML associations and naming classes.

ID=Diagram ID - U=Understanding - M=Making Decisions - D=Drawing
O=Other - Time = Time Spent - Revs=Number of Revisions

5.5.2 RQ2: Can we learn which steps students find difficult in creating designs?

From the log statistics we have the impression students use a great amount of MOVE actions. This could indicate they have difficulties in making a layout. Layout should help them to get 'the big picture'. This could be due to the fact that students don't have experience in software design and have to 'rethink' their designs over and over again. There is a large spread in time used to complete the designs. This could be due to the fact that some students have to think longer about decisions that they make about their designs. It could also mean that they went for a cup of coffee during the assignment. As said before, there was no time constraint or class setting that forced them to work continuously on the task. We found out from the logs of the modelling tool that it is difficult to accurately measure the exact time students spent on a particular activity. On the other hand, looking at the gaps in the visualisations between activities, one will find several gaps where a student 'does nothing'. Large gaps of, for example, 1000 seconds are difficult to interpret. They could mean anything, but gaps of 60 to 200 seconds could indicate thinking time. Assuming that, it is remarkable most of the times this is followed by a class MOVE action. The follow-up questionnaire has shed some light on our understanding about the gaps and what students found difficult about their task. For example students mention: *'It is hard to identify design elements from the assignment text'* and *'How to split each element up into either a class or have it as an attribute?'*.

Students spent most of their time in understanding the assignment and making decisions on what the model should look like. Some students reported that they needed time to get themselves familiar with the modelling tool. This resulted in a lot of intuitive sketching and deletion at first, then delaying for quite a long time before actually start drawing. They don't appear in Table 5.4 because we don't have their logs.

5.5.3 RQ3: Do students that perform better on modelling tasks use different strategies than those who perform worse?

To answer this question we only looked at the Depth First and Breadth First group. We evaluated the diagrams of the Depth First group and the Breadth First group. We took into account that the students are novices and that they looked at the layout (good readability) and richness (e.g. applying aggregation in stead of an 'empty' association for explaining a relationship better) of the diagrams. From this evaluation the Depth First group performed best. This could be explained with the idea it helps students to associate classes when classes have more detail (operations/attributes). When leaving these elements out of the classes first, the Breadth First approach, they will have more difficulties with associations.

5.5.4 Other Observations

From the results it seems a common trend to add attributes and operations at the same time.

There were two students (16 and 9) that did the task at once after spending time researching on the assignment and sketching on papers. Their comments where:

- "Two days, first 20-30 minutes was spent reading up on the material and think of a possible outline. Created the diagram the next day"
- "First I made a sketch on paper and practised a little bit with the given tool in order to get familiar with the tool environment. After that I did the assignment at once."

Their logging files do not completely cover the process.

5.6 Threats to Validity

We are aware of the difficulty in generalising the interpretations we presented in Section 5.5. A statistical approach would help here, but the amount of subjects is not ideal for a statistical analysis. Our first aim was to find possible strategies.

The time recorded in WebUML does not cover the 'real' total time, because of the fact students can spend time before they start modelling or when the tool is not used. We tried to capture this difference by asking them in the follow-up questionnaire.

5.7 Conclusions and Future Work

In the experiment reported in this chapter we revealed typical strategies that students use for a software design task. We developed our online UML editor: WebUML. WebUML is designed to log the different activities students perform during their tasks. Together with the editor we developed LogViz, a visual log analyser, in order to visualise WebUML's logs.

This chapter showed that, by visualising the students' activities, we were able to identify typical strategies students use to make class designs: Depth Less, Depth First, Breadth First, Ad Hoc. Two approaches stood out: the Depth First approach and the Breadth First approach. (RQ1)

From our logs students seem to use a lot of time to get an organised layout, which is likely to be a typical problem for novices. Some of the students take a long time to finish their assignment. On the one hand this could be related to all the moves. On the other hand they mention themselves it is hard to transfer from text to UML elements or to decide between attribute or class. (RQ2).

From a statistical point of view we could not prove if one strategy leads to better designs than the other. We did however find indications that a Depth First strategy could lead to a better layout, and therefore a class diagram that is better to understand. (RQ3)

Our tools open up new ways of studying the process of class design. Having the possibility to analyse students' modelling behaviour enables us to develop better educational programs and tools. Recognising certain strategies could help us to give better feedback during a task in stead of post-task.

In future research we will increase the amount of subjects to also statistically investigate the strategies and find possible correlations between model quality metrics and strategy actions. We want to test our tools as part of other educational suites, i.e. e-learning or MOOC-like environments.

Chapter 6

Uncovering Common Difficulties of Students Learning Software Design

This Chapter further investigates the prelimary study that was conducted in Chapter 5. It describes an online experiment with the aim to reveal common difficulties and modelling strategies of students during class diagram design.

To gain more insight in their difficulties while performing a software design task, the students were asked to register their arising questions using a form in WebUML (Chapter 4). To gain more insight in the overall class design approach we compared students that use Breadth First strategies with those that use Depth First strategies in terms of grading overall assignment performance and diagram layout. Based on statistical analysis and diagram observations we noticed i) students seem to introduce noise by misunderstanding the assignment text ii) students have difficulties in choosing the right abstractions iii) good layout seems to lead to a good overall grade iv) the difference in grade between the Breadth First and Depth First strategy groups is not significant, however comparing the number of element moves, as possible measure for efficiency, indicates significant difference. We suggest follow-up studies to investigate the results in more detail.

This chapter is based on the following publication: Dave R. Stikkolorum, Truong Ho-Quang, Bilal Karashneh, and Michel R.V. Chaudron. Uncovering students' common difficulties and strategies during a class diagram design process: an online experiment. In *Educators Symposium 2015, co-located with the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, 2015

6.1 Introduction

Novice software designers, students or newly-qualified professionals, struggle with different problems during their training tasks or first software development assignments. Especially in the context of software design. Several studies show students' difficulties that are related to UML syntax [80] (or other modelling languages) and reasoning, in special abstraction skills [69][113][141]. Educators are all aware of the different learning styles [67][154] students have. In order to adjust study programs to be more suitable for different types of learning styles we need to understand why students make the mistakes they make.

While doing an assignment, lecturers are not always present to give feedback on the steps students take or questions they have. Furthermore, students will not always pro-actively ask a question the moment they have one. We assume there is a relationship between the questions they ask themselves during a modelling task and how the design process takes form. Recording students' questions would provide more insight into their design processes - insight in students' considerations while creating a design or what typical decisions they make or think they have to make.

In Chapter 5 we described how students used our online UML editor WebUML[1] during a class diagram design task. WebUML is capable of logging UML class design activities (such as creating elements, movements, deletion etc.). We introduced our way to categorise the approach into depth first (DF) and breadth first (BF) strategies. These two strategies are part of a set of four strategies (see Table 6.1) that explains students' overall approach of making a software design. BF and DF were the dominant approaches. Students seem to construct their design by building classes with detail first and then associate (DF) or having an overall class framework first and then add detail (BF).

In order to do more in-depth research on the design steps students take and the problems they face we extended WebUML with a 'register-your-question-button' to record questions and comments students have during their modelling task.

A part of the students' approach probably consists of how they organise their diagrams, the quality of the layout. In this chapter we want to follow up on our previous study on a large scale to show whether the strategies we identified are common. We used 98 student-pairs to achieve this. Our main research questions were:

- RQ1: Do students have typical questions that arise during the development process of a software design?

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

Strategy name	Activity sequence
Depthless Strategy:	class → associate
Depth First Strategy:	class → add detail → associate
Breadth First Strategy:	class → associate → add detail
Ad Hoc Strategy:	no structured approach

Table 6.1: *Different strategy types*

- RQ2: Does the Breadth First or Depth First strategy leads to a better grade for a class diagram design task?
- RQ3: Does the layout of the diagram influences the grade of a student's work?

The remainder of this chapter is organised as follows: in Section 6.2 we explore related work, in Section 6.3 we explain the method we used. After showing the results in Section 6.4 we discuss them in Section 6.5. Validity threats are discussed in Section 6.6. We conclude and identify future work in Section 6.7.

6.2 Related Work

Leung and Bolloju [80] relate common mistakes novice modellers make to 3 quality categories: syntax, pragmatic and semantic. And suggest this knowledge could be used for training purposes. They don't relate the categories to the design process or grading. Although syntax is important, our study focuses more on the semantic category.

The visualisation of log files is close to the process of mining research. There are a number of tools that provide different ways of presenting event-based logging. The following two are most related to our visualiser.

Song et al. [125] introduced the tool Dotted Chart that displays the events of the instances of a process as coloured dots on its time-lines. However, the tool is limited in its ability to present different steps of processes (e.g., creation, movement, renaming of a particular activity) on the same chart.

Claes et al. [26] followed up on the research in [125] and introduced a way to improve the visualisation. The tool PPMChart visualises modelling operations of one modeller (one person) in the construction of a single process model as different coloured and shaped dots in a horizontal time-lines chart. The authors concluded that this approach of visualisation would provide audiences with different views at different levels of abstraction on the process modelling operations. However, the tool is not able to present multiple logging files concurrently or to detect modellers' patterns automatically.

Störrle addresses the relation between the understandability of a model and how well a design is organised. Making a good layout seems to be correlated with cognitive load. He mentions novice modellers benefit more than experts from good layout. The size of the layout seems to stand out. The larger, the more difficult the design is to understand. [144] [145]

Our approach of registering questions is arguably a 'think-aloud' [82] kind of approach. We are not aware of other research that uses that approach.

6.3 Method

In this section we describe our experimental approach. First we show the overall framework. Then we discuss the student participants and the class design task they had to model. Furthermore we explain the tools we used for collecting the data and performing the analysis. Subsequently, we discuss our approach in grading the students' models and analysing the results. More information is found in our technical report [130]

6.3.1 Overall Framework

Figure 6.1 shows the overall framework that was used for our online experiment.

Data collection was done online through a class design assignment performed by students from Uganda. The students were asked to submit their solution to a model task by using the pre-discussed web-based UML editor (WebUML). Besides the model file, students' modelling activities and questions during the modelling session were logged. After submitting their assignments the students' diagrams were graded by three experts in terms of overall task performance and layout.

The data analysis phase consisted of: i) an analysis of the registered student questions ii) an analysis of students' modelling strategies with a focus on the two emerged approaches Breadth First and Depth First. We developed a string pattern matching method to automatically detect the two approaches in the students' logging files iii)

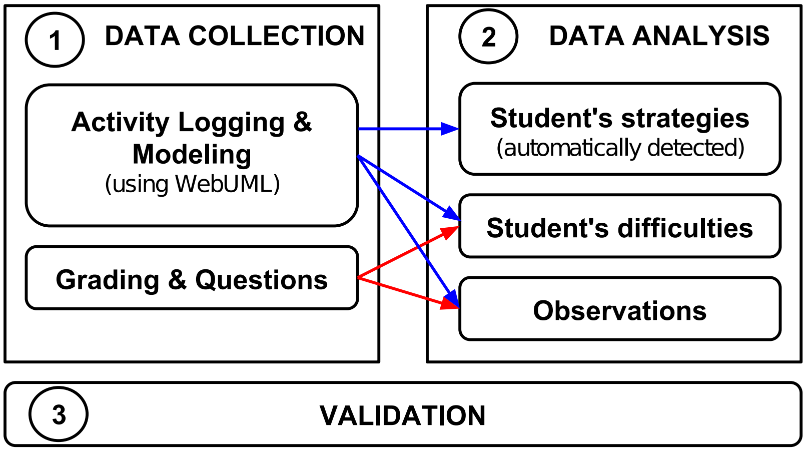


Figure 6.1: *Experimental framework*

observation of students' diagram solutions.

6.3.2 Participants

120 student-pairs were invited to perform a design task with our online class diagram editor. The students involved were 3rd year Software Engineering students following a Bachelor of Science degree program. They already followed courses in software design principles, UML and programming. The task was graded but not used as part of the course grade. The native language of the students is English. We did not choose to work with pairs on purpose. This is the students' university's approach for practical assignments.

6.3.3 Task

The students were asked to make a class design of a game that was presented online in a pdf-file. The text was a short (153 words) one paragraph description and was written in English:

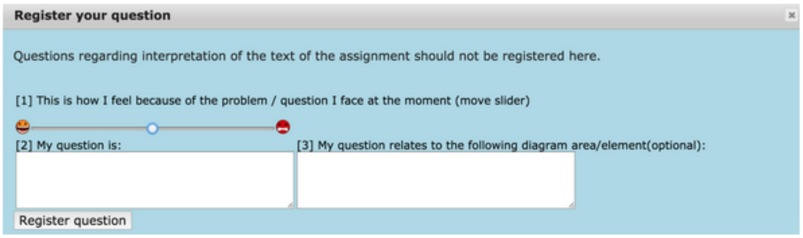
The Modelling Task – a Tank Game[2] *In this task you will design a game with use of the UML class diagram. You do not need to use packages in this assignment. The description of the game is as follows: A player (user) controls a certain tank. This tank is a Panzer Tank, a*

[2]text: B. Karasneh

Centurion Tank or a Sherman Tank. They fire bullets and Tank shells. Bullets can be Metal, Silver or Gold bullets.

A tank moves around a world (level). The aim is to destroy all other tanks in the world. After a world has been completed the tank advances to the next world. A list of all the worlds visited is kept.

An entire game consists of 8 levels. A world contains a maximum of 20 tanks that compete for victory. Each tank remembers which tanks it has destroyed in the past. The score for each level is kept by a scoreboard that gets notified by the individual tanks each time an opponent is shot. The players control their tanks through an interface allowing for steering, driving (reverse / forward), switching ammo and firing.



Register your question

Questions regarding interpretation of the text of the assignment should not be registered here.

[1] This is how I feel because of the problem / question I face at the moment (move slider)

[2] My question is:

[3] My question relates to the following diagram area/element(optional):

Register question

Figure 6.2: *Form used to register students' questions*

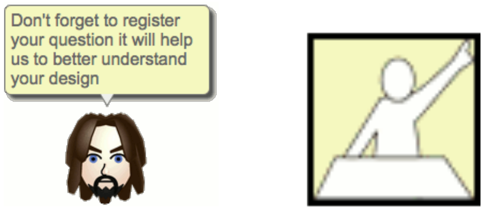

Don't forget to register your question it will help us to better understand your design

Figure 6.3: *Reminder and question button in WebUML*

The students were asked to press a button whenever they ran into difficulties or had a question or remark. With the web-form that popped up they were able to i) explain how frustrated they were at the moment that the difficulty arose and ii) record the question. Figure 6.2 shows this web-form.

The web-form was not meant for getting answers from a lecturer or assistant instantly, but only for registering the student's questions and/or difficulties. The online editor reminded the student every 5 minutes in a non-intrusive way: a message on the left side of the screen appeared and the question icon was highlighted (Figure 6.3). The experiment was part of a regular practical class of two hours, but they were allowed to use more time. The students were asked to upload their work when they were finished.

6.3.4 Instrumentation

For the experiment we used three of our own tools. They were meant for creating the class designs, logging the activities and analysing the log files.

Modelling & Logging tool: WebUml is an online class diagram editor. WebUML is capable of logging students' activities in a comma separated file (explained in [136]) and saves the last version of the diagram in a xmi file and a png picture file. The files are compressed in a zip file and uploaded with an upload button. WebUML was designed to address a larger number of students independent of location or time. For this experiment we extended the log capabilities with the addition to register student questions. Students can register questions by filling in a form (explained in Subsection 6.3.3)

Analysis tools: For visual analysis we use LogViz[3]. LogViz is capable of displaying the designers' activities in WebUML over time. We can compare different log files (in this case different student-pairs) and measure times between activities. The tool is able to auto-classify the log files by the students' design strategies (Depth First, Breadth First, Depthless and Ad-hoc).

For gathering statistics, such as the number of creations we use StatLog[4]. StatLog reads WebUML's log files, counts all design activity occurrences in the logs and saves a table with the data that was found.

The registered student questions that were recorded in the log files were filtered out using general command line tools and regular expressions and then combined into one file.

6.3.5 Assignment Evaluation

The students' work was evaluated in three ways: i) every model was graded for overall task performance ii) every model was graded for layout quality iii) every activity log was automatically labelled with a strategy.

Grading: the grading was done during 4 grading sessions by 3 experts having more than 6 years of experience in software design and education. The experts considered

[3]LogViz - `https://gitlab.com/truonghoquang/LogVisualizer`
[4]StatLog - `https://gitlab.com/stikkolorum/StatLog`

two aspects to grade: i) task grade, how well does the diagram reflect the problem of the assignment? ii) the layout, how well is the diagram organised? For both of the grades a rubric was used (see Table 6.2).

Table 6.2: *Class Diagram Rubric for Grading Design Modelling*

Grade	Judgement, criteria description
1	The student does not succeed to produce a UML diagram related to the task. He/she is not able to identify the important concepts from the problem domain (or only a small number of them) and name them in the solution/diagram. The diagram is poor and not/poorly related to problem description with a lot of errors: high number of wrong uses of UML elements mostly no detail in the form of attributes or operations.
2	The student is not able to capture the majority of the task using the UML notation. Most of the concepts from the problem domain are not identified. The detail, in the form of attributes or operations, linked to the problem domain is low. Some elements of the diagram link to the assignment, but too much errors are made: misplaced operation / attributes non cohesive classes few operation or attributes are used.
3	The student is able to understand the assignment task and to use UML notions to partly solve the problem. The student does not succeed to identify the most important concepts. A number of logical mistakes could have been made. Most of the problem is captured (not completely clear) with some errors: missing labels on associations missing a couple important classes / operations / attributes Logical mistakes that could have been made: wrong use of different types of relationships wrong (non logical) association of classes.
4	The student captures the assignment requirements well and is able to use UML notations in order to solve the problem. Almost all important concepts from the problem are identified. Some (trivial) mistakes have been made: Just one or two important classes / operations / attributes are missing Design could have been somewhat better (e.g. structure, detail) if the richness of the UML (e.g. inheritance) was used.
5	Student efficiently and effectively used the richness of UML to solve the assignment. The problem is clearly captured from the description. concepts from the domain / task are identified and properly named The elements of the problem are represented by cohesive, separate classes (supports modularity) with a single responsibility In the problem domain needed attributes and operations are present Multiplicity is used when appropriate Naming is well done (consistent and according to UML standard) Aggregation / Composition / Inheritance is well used No unnecessary relationships (high coupling) are included.

The rubric consisted of a 5 point scale and the experts agreed on the rubric before the experiment started. In advance of the actual grading, a set of possible ideal solutions was discussed for calibration. The grading was done in two steps: first the assessors graded all diagrams separately, then they discussed the differences in grading and gave the diagram the final marks (task, layout) after consensus. Grading was done in batches of 25 class diagrams.

Student's strategy: students' strategies were automatically classified using a string pattern matching approach. Extracted from the log files, the creation activities of class diagram elements were constructed as a string of the 4 letters: C (represents CLASS), O (represents OPERATION), A (represents ATTRIBUTE) and R (represents

CCCCCCCAOCCCCCOAOOOOCRRRAORRRRRRRRRRRRRRCRCRRRCRCROOAOAA

Figure 6.4: *String fetched from the log to determine a student's strategy*

associations/relationships between classes). Figure 6.4 shows an example of such a creation string. As an example, we show the regular expressions of the two most used strategies in Table 6.3. A log is labelled as the strategy that matches the longest string in the creation string.

Strategy	Activity order	Regular Expression
Breadth First	class → associate → add detail	[C]+[R|C]+[O|A]+
Depth First	class → add detail → associate	[C]+[O|A]+[R|C]+

Table 6.3: *Regular expressions used to fetch a strategy from the log file*

6.4 Results

In this section we present the results of our online experiment. First we explain the overall response and the questions students asked. Then we present the statistics of the class design log files combined with the experts' grades and modelling strategies that were identified. Finally, we summarise the observations that were made during the grading process of the class diagram designs.

6.4.1 Recorded Log Files - Overall Response

We recorded useful log files of 98 student-pairs. Although 100+ student-pairs participated in the experiment, some files were corrupted or incomplete.

6.4.2 Registered Questions

Out of the total response (N=98) 31 questions from 24 different student-pairs were registered during the experiment. From 7 student-pairs we received 2 questions. 1 pair asked 3 questions. The others asked 1 question. We divided the questions into the following 5 categories (questions can fit in multiple categories):

Category	Occurrences	Example question from respondents
Task Comprehension	7	How many users are allowed to play at a given time?
Tool Usage	16	How do I draw associations?
Tool Feedback	9	Why doesn't the tool support adjusting of the class in the event when the operation name is too long to fit in the fixed size?
UML/OO Comprehension	3	Could the different types of tanks be modeled as specializations of the tank class or as an attribute in the Tank class?
Notation/Syntax	7	How do you represent inheritance?

Table 6.4: *Categories of questions students asked*

- Task Comprehension: how well the student understood the task
- Tool Usage: Questions about the usage of the online tool.
- Tool Feedback: Remarks about or suggestions for improvement of the tool.
- UML/OO comprehension: Related to the UML and/or object orientation comprehension of the student.
- UML Syntax/Notation: questions about graphical representations of UML or other elements the student wanted to draw.

Each question was rated in sense of relevancy to the task on a 1 to 5 scale (no relevance - high relevance). For 1 question it was impossible to determine the relevance because of the unclear wording of the question. Both category overview and relevance rates are shown in tables 6.4 and 6.5. Most questions (16) were related on how to perform certain actions in the tool (Tool Usage). The lowest number was related to the comprehension of OO concepts and/or UML.

On the 'feeling' indicator 22 student-pairs responded neutral, 7 student-pairs felt bad/angry while recording questions, 2 pairs felt happy. Remarkable is that questions about UML syntax only links to neutral or positive feelings. They don't relate to angry feelings.

Relevance index	Occurrence	Example question from respondents
1	2	How do you connect many arrows together for inheritance?
2	8	It is hard to delete an attribute once its written
3	7	Is a scoreboard an attribute?
4	9	Where should we note our assumptions?
5	4	If scoreboard falls under class, what attributes can it take in this case?

Table 6.5: *Relevancy of recorded questions*

6.4.3 Statistics of the Logs and Evaluation Data

The log files consist of the recorded user data: time spent (in minutes) and the different modelling activities (in frequencies) such as creating UML elements (classes, attributes etc.). The dataset was extended with a number for the grade and a number for the layout evaluation. Table 6.6 shows the statistics summary of the logged data and evaluation grades. The total time row contains some extreme values. This is due to some students uploaded their work late in the evening or the next morning or due to technical errors. If we discard these cases we find a range from 11 - 400 minutes with mean = 92.66, sd = 83.06 and N=87.

Statistic	N	Mean	St. Dev.	Min	Max
grade	98	3.06	0.84	1	4
layout	98	3.27	0.65	2	5
totaltime	98	1,215,771	5,267,869	11	23,819,063
creates	98	70.54	28.71	16	168
sets	98	42.36	17.07	3	96
adds	98	25.59	15.18	3	87
moves	98	77.08	63.44	7	364
removes	98	11.10	11.05	0	71
readings	98	3.09	5.18	0	36
modellings	98	2.91	5.15	0	36
comments	98	0.32	0.65	0	3

Table 6.6: *Descriptives of all logged variables and evaluation*

6.4.4 Modelling Strategies

The extracted creation strings from the logs that were mentioned in Subsections 6.3.4 and 6.3.5 were explored using the regular expressions and manual analysis. We identified the two major groups: Depth First (N=43) and Breadth First (N=45). Also, a number of student approaches could be described as: 'Ad Hoc' (N=5) - they don't use a clearly observable pattern - and a 'Both' (N=2) group that seems to switch between Depth First and Breadth First. Some logs were labelled with 'U' (N=3). In this case there was no complete pattern recorded or just missing.

6.4.5 Group Comparison

To compare the performance of the student-pairs (the grade) grouped by the strategy they use (Depth First, Breadth First) we performed a Wilcoxon rank sum test[5] . We only compared the groups Breadth First (N=45) and Depth First(N=43). Although the mean of BF (grade=3.13) is higher than DF (grade=2.88) the Wilcoxon test did not show a significant difference between the two strategy groups (W = 1122, p = 0.1735).

If we compare the student-pairs in terms of the amount of moves grouped by strategy the Wilcoxon test does indicate a significant difference between BF and DF strategy student-pairs (W = 1354.5, p = 0.0013, meanBF = 92.38, meanDF = 59.26)

6.4.6 Correlation

At first glance no remarkable correlations were found in the dataset of the experiment. However, excluding very poor models did yield an interesting correlation. During the grading process we came across several class diagrams that were poor in the sense of a low number of classes and did not use richness of the UML (such as inheritance). These kind of diagrams most of the time scored high ($\geq$3) on layout. There cannot be a lot of things wrong with the layout of poor diagrams, except from the alignment. The examination of the subset (N=66) that excluded such cases resulted in a correlation coefficient of 0.32 (p=0.009) between layout and grade. Which can be seen as a moderate positive correlation.

[5]https://stat.ethz.ch/R-manual/R-devel/library/stats/html/wilcox.test.html

6.4.7 Student Diagram Observations

During the process of grading the diagrams we discussed the common mistakes or additions students seem to make. In this subsection we summarise our observations.

Mistakes or Unidentified Elements *Reflexive associations (self associations)*: the task description should have triggered the students to use a reflexive association or with the help of an intermediate class. It seemed common not to notice this.

Wrong use of UML elements: a typical mistake for students is to use the wrong element for a certain purpose, such as using aggregation or composition when inheritance was meant.

Misplaced operations or attributes: students seem to have difficulties to identify the responsibility of a class and only save this class for this purpose (cohesion).

Forgotten elements: information that appears in the task is not represented in the solution (such as classes, operations, attributes etc.).

Concept as attribute instead of class: students have difficulties deciding between classes and attributes (abstraction).

'Loose' classes: a number of diagrams consisted of classes that did not have any relation with another class.

Addition of Elements Sometimes students felt the need to include non standard notation in the attribute fields, such as code notation or numbers instead of using the multiplicity element. Although it was not needed to include types (such as Int or String) a number of students added this to an attribute or operation. They also tend to include an 'id' as part of the class attributes. Although expected, from experience in classrooms, students seemed not to use too many associations per class (high coupling).

Grading While grading, some criteria seemed to be more important than others. For example, missing a clear relationship between two main classes was considered as a major design flaw, while the misdirection of relationships or missing a label are not considered as a big problem. At the moment our rubric does not contain this distinction per level although the experts used it unconsciously.

During our discussions we realized we could add some more layout criteria for layout quality. From the experiment experience we propose: symmetry, the spread-out of the elements and diagram size. We did not take these into account while grading layout.

6.5 Discussion

In this section we discuss the results by answering the research questions.

6.5.1 Do students have typical questions that arise during the development process of a software design?

It is not in every student's nature to ask questions and it is often considered as something to avoid. Because of the number of questions (31 registered by 24 student pairs), we could not do any statistical analysis. We did identify typical categories: Task Comprehension, Tool Usage, Tool Feedback, UML/OO comprehension, UML Syntax/Notation. We observed that most of the questions were related to the tool. Tool use is discussed widely and often considered difficult in use [2]. For some students this likely seems to distract them from their tasks. We assumed our tool is very easy to use. The students were able to use it without any training other than a little practice in advance.

Students occasionally seem to seek for extra information that is not in the text, but at the same time it is not needed for the solution. 5 out of 7 task related questions were considered to be of low (2) relevance.

Surprisingly the number of questions related to UML/OO comprehension was low (3). This could be explained by the fact that the students had prior knowledge and were already in their 3rd academic year.

Based on the relevance number, most student-pairs seem to be capable of asking relevant (index 3-5) questions. Which points out, that doing this by the means of an online tool is a good approach.

Derived from our diagram observations, and supported by the students questions, choosing between attributes or classes (which addresses OO comprehension) seems to be a general difficulty for students.

6.5.2 Does the Breadth First or Depth First strategy leads to a better grade for a class design task?

From the statistical analysis we cannot conclude one of the approaches leads to a better grade. There is no significant difference. We are not yet convinced that both strategies can lead to diagrams of the same quality. Deeper investigation must be done. We could rerun the experiment with a different grade scale and see if it has a better spread. At the moment in the 5 point scale grades set the grades 1 and 5 were not represented that much which may has lead to the results we have now. Another option is to run the experiment with professionals and investigate if there is a bigger share of BF or DF strategies.

According to the statistical test there was a significant difference between the two strategies in terms of number of movements. If we interpret this as a measure for efficiency it suggests the DF could be more efficient than the BF approach.

In case both strategies can result in comparable quality models it still can be the case certain strategies cause certain difficulties. In our dataset we have not enough questions recorded to perform such an analysis.

6.5.3 Does the layout of the diagram influences the grade of a student's work?

Based on the moderate correlation of 0.32 we advise students to pay attention to create a nice layout in their overall design approach. 32% of a grade can explained by the layout. We assume a good layout not only helps the student to understand his/her own model, but also provides the lecturer better insights.

6.6 Threats to Validity

Although we evaluated the model and layout according to a rubric, they are still graded manually which introduces a bias. On one hand we tried to reduce this bias to determine the grade through discussion. On the other hand this same discussion could have lead to a milder evaluation.

To be sure the automated process of labelling the strategy worked the results were checked by hand by two experts.

We are aware of the fact we used pairs of students. The results might not represent individual students.

6.7 Conclusion and Future Work

In this chapter we presented our approach to uncover students difficulties and strategies while making a class design. We recorded the solutions, design activities and questions students have in a log file. A small number of students registered their questions from which a majority was relevant to their assignment. Based on the questions and diagram observations students introduce noise by adding unneeded elements and have difficulties in choosing between attributes and classes as representation of certain concepts from the assignment text.

In general the student-pairs clearly seem to use a DF or BF strategy but do not significantly differ in terms of grades. Both strategies could profile students and different profiles could lead to different difficulties during modelling. Reruns of the experiment in different settings could gain more insight. Also, collecting more student questions that are related to certain strategies could help us to explain our questions.

Comparing the amount of movements between BF and DF, the results suggest DF to be more efficient than the BF strategy.

Based on the moderate correlation we found between grade and layout, we assume that paying attention to a nice layout helps students to perform better on their modelling assignments.

In future research we continue to explore students' strategies and difficulties. Based on the results of the experiment in this chapter and future research we aim to develop educational programs in the field of software design that fit students' profiles better.

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

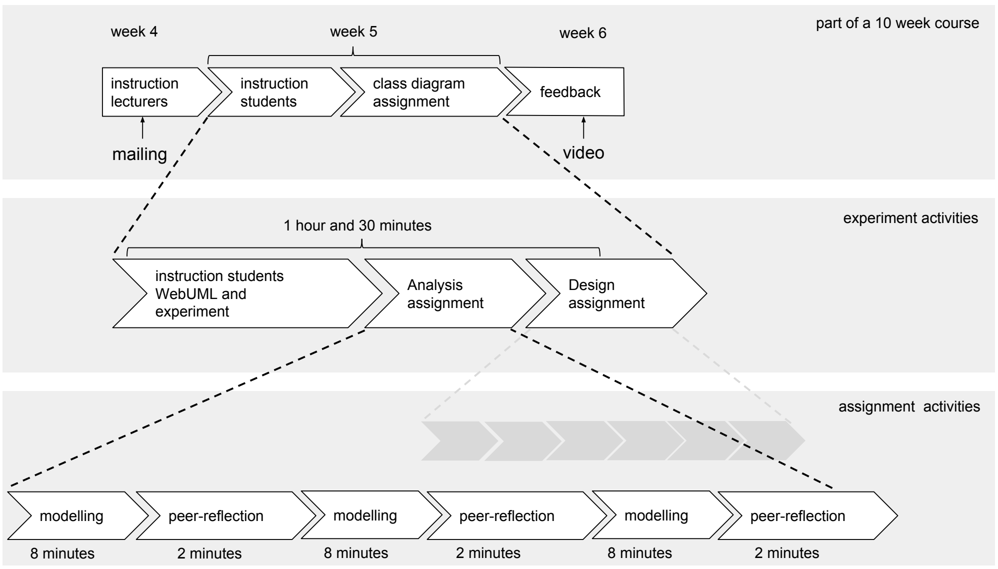


Figure 7.1: *overview of the teaching method*

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 difficulties 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.

Chapter 8

Evaluating Didactic Approaches used by Teaching Assistants for Software Analysis and Design using UML

Teaching assistants (TAs) are employed for supporting various activities of a course, from assisting with course material, grading and, predominantly, bridging the interaction between teachers and students. This role is even more significant in large courses with dozens of students, particularly during lab sessions with practical assignments. Therefore, here we explore how the role of the TA supports students to overcome the challenges they face when learning software design. During a bachelor course on Analysis and Design, we collected data from i) weekly interviews with TAs, ii) student's assignment hand-ins and iii) the corresponding feedback reports written by TAs to students. By analysing these three different sources we propose guidelines and practices for effective deployment of TAs.

This chapter is based on the following publication: Dave R. Stikkolorum, F. Gomes de Oliveira Neto, and Michel R.V. Chaudron. Evaluating didactic approaches used by teaching assistants for software analysis and design using uml. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 122–131, New York, NY, USA, 2018. ACM

8.1 Introduction

Teaching assistants (TAs) support various tasks in a course, such as grading assignments, lecturing, supporting students during lab sessions, advising students, supervising projects, etc. Consequently, assigning TAs to these different activities inherently becomes part of the course planning activities performed by a course coordinator (usually faculty members at a university). When addressing student's needs related to feedback and guidance in courses with practical activities (such as programming and modelling), the responsibilities of a TA involve lecturer-like duties in order to guide students towards their deliverables.

In our practice we often use TAs to support students during practical group work. We believe that the use of teaching assistants in a software analysis and design (SAD) course can be of great value. We see the fact that their reasoning approach is close to the target student, hence bridging further the lecturers and the students to improve knowledge transfer. TAs are easy to approach and can relate to students' study experiences [95].

The aim of this report is to study the didactic approaches of TAs during the practical sessions in a software design analysis and design course. We aim to obtain insights into what particular elements of the TAs intervention help students further with the development of their software design skills. With this insight we would have a better understanding of students' reasoning and common difficulties during their learning process. In addition it can support the development of future training programs for TAs. Based on our experience from a software analysis and design course, we aim to answer the following research questions:

- RQ1: What are typical challenges in 'Software Analysis and Design' for which students seek TA support?
  - RQ1.1: Do students focus on different matters than the topics in the lectures?
- RQ2: Which didactic approaches do TAs use to support the challenges students have?
  - RQ2.1: Are there approaches that are specifically useful for Software Design contexts?
- RQ3: Which expectations do TAs have about students and vice versa?
- RQ4: Do feedback and guidance of the TAs focus on the same matters as the lecturers?

In this chapter we present and discuss the results of a case study that was conducted during an 'Analysis and Design' course in the second semester of the first bachelor study year. We collected data from student surveys, the TA feedback in the online learning system and from the weekly TA-teacher interviews during the course.

By answering the research questions we contribute to the understanding of the interaction between students and TAs. In addition, we contribute to the existing investigation of student's common difficulties in UML Analysis and Design assignments (i.e. [79] [132]). With our research we confirm and extend these studies. Moreover, this research also provides a basis for checking whether the TAs address the students' problems.

The remainder of this chapter is organised as follows: in Section 8.2 we discuss related work, in Section 8.3 we present the research method that we used, subsequently in Section 8.4 we present the results that are discussed in Section 8.5. Section 8.6 consists of recommendations for future deployment of TAs. Threats to validity are discussed in Section 8.7. We conclude and suggest future work in Section 8.8.

8.2 Related Work

In this section we discuss related work. In our research we aim to reveal the benefits of the intervention of a TA in a Software Engineering (SE) lab context. To our knowledge there is little published about the use of TAs during SE lab sessions. Specifically, research about what specific interventions are observed in the context of Software Analysis and Design, is not available to our knowledge.

To provide more background on the use of TAs we explore further in this section: i) general use of TAs and ii) use of TAs in a CS context.

8.2.1 General Application of TAs

Because of a growing employment of TAs in higher education in the UK, Park [99] distils lessons for future TA usage in the UK. He studies North American publications about the use of graduate students for teaching undergraduates. The literature explored different TA activities including lab demonstrations, practical classes, leading tutorials and seminar groups, and lecturing. Benefits that the use of TAs could bring include: reduction of teaching load, funding for postgraduate research students, teaching experience for the TA, apprenticeship model for future professors. The author presents a summary of lessons learned with a special focus on the following issues: i) selection and preparation, ii) training, iii) mentoring, iv) practical/personal issues, and v) professional development. Interesting to mention is that the summary also includes the remark that some students have difficulties with the ambiguity of their role – teacher versus student; employee versus apprentice.

Hardré et al. state that institutions need effective and efficient ways for the the development of TAs teaching skills, but that many TAs are not prepared enough for teaching responsibilities [44]. They propose the use of instructional design (ID) as a tool for the development of a TAs teaching expertise. ID should support TAs in thinking and planning like expert teachers, which would help them improve instructing undergraduates.

Hardré et al. explore what contributes to the development of a teaching assistant through a two-day professional development workshop [45]. The study involved 210 TAs and focused on general TA development. The study involved three clusters (hard sciences, social sciences, and arts & humanities). They found that TAs are aware that training can contribute to their learning and development of teaching. Features of the workshop that TAs value the most in relation to their development were: expertise of speakers, structural design of events, and quality of support materials. Eighty percent of TAs reported to be motivated to continue learning about instructional theory and practice. The authors suggest future work related to the degree of autonomy of the TA, the role, influence and effort of the supervisor and suggest to compare experience between different disciples.

8.2.2 TAs in Computer Science

Danielsiek et al. present a preliminary competence model for the training of undergraduate TAs (UTA) for computer science students [30]. Next to practical benefits, such as dealing with the increasing numbers of students, the authors point out the positive influence on student retention. The authors also point out there is little attention to UTA training in the community so far (2017). With the evaluation of their prototype training course they expose the different competences and perceptions UTAs have. In some of the observed courses more teacher-centered beliefs are observed whilst in others more student-centered are observed. Their analysis of applying their training course shows a positive change on the student-centred beliefs of the students.

Patitsas discusses a case study of TAs in computers science lab sessions [101]. The author comments there is not much literature on lab TAs (as of 2013). The author found that the following factors contribute to TA development: practice, teaching multiple courses, mentoring, staff meetings, team teaching and feedback. The team teaching is valued highly by the TAs, since it enabled them to learn from each other. For labs, the TAs most benefit from course-specific experience from others than general teaching experience.

8.3 Research Method

In this section we explain our overall research approach, how we collected the data, the course and assignments we used in our case study, the participants of our case study and how we analysed the data.

8.3.1 Overall Research Approach

In order to answer our research questions, we conducted a case study during a course for a bachelor programme in Software Engineering and Management (SEM) at the University of Gothenburg (Gothenburg, Sweden). The course has 63 registered students and is taught in the second term, for first-year students. The students have a basic knowledge about object-oriented programming from a previous course in the first term. The course covers object-oriented analysis and design, has two teachers and five TAs. A summary of the course activities is illustrated in Figure 8.1. Given that our main interest is related to TA activities, we rely on particular course elements to conduct the study, in particular: (i) feedback by the TAs on assignment submissions, (ii) supervision sessions with everyone involved in the course, and (iii) weekly meetings with TAs to verify course progression and students' performance. Besides the activities described above, we use two instruments to collect data: (i) interview sessions with the TAs, and (ii) questionnaires answered by students. Consequently, we perform qualitative analysis in our data to gather insights on the trade-off of using TAs in a software design course. Our data analysis and collection approach is based on triangulation. The collection and analysis of the data is discussed in Sections 8.3.2 and 8.3.5. An overview of the overall research approach is presented in Figure 8.1.

8.3.2 Data Collection

For our research we collected data from the following three sources: An *online survey* – four times, after every assignment, we asked the students about their experiences with the TAs. We focused the questions on particular topics of the specific assignment and on how the TA that was involved helped to understand the topic better. The survey consisted of a non-free text part and a free text part. The questions are shown in Table

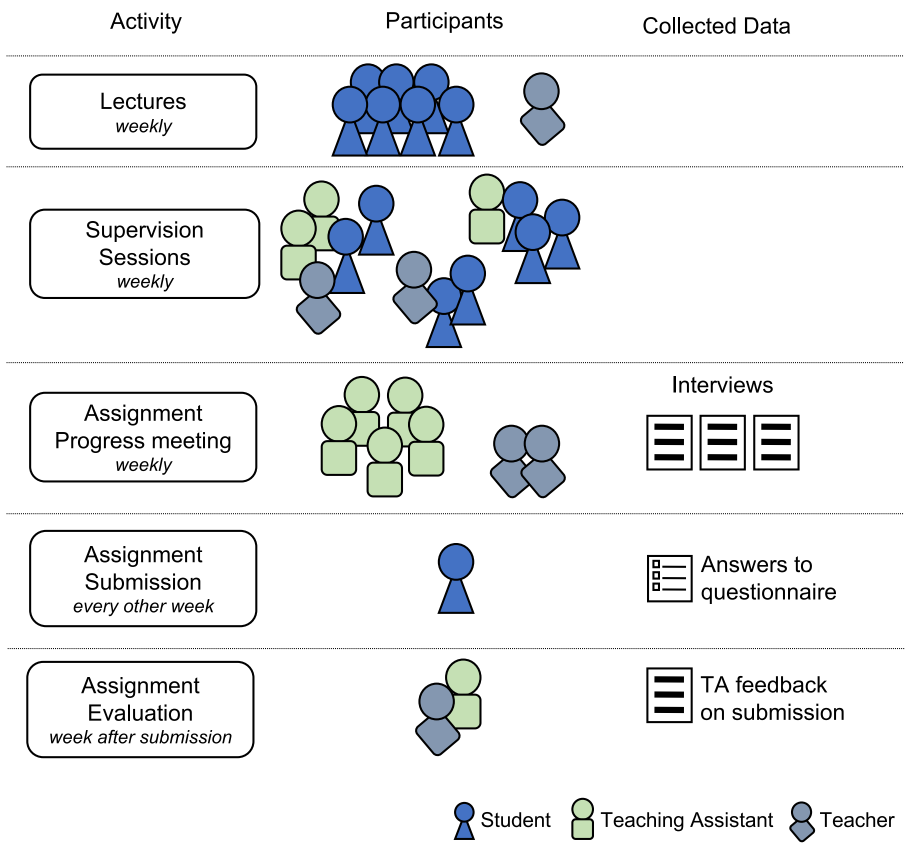


Figure 8.1: *Summary of course activities and data collection.*

8.1. Limesurvey[1], an online tool, was used for this purpose.

A *structured TA interview* – every week, during the course, the lecturers interviewed the TAs. This interview was structured targeting topics such as: i) problems with students, ii) observations and iii) discussions about the approach. We used an audio recording application on a mobile phone to record the interviews.

TA feedback text on hand-in – according to feedback guidelines provided and defined by the lecturer, each TA wrote down feedback on the hand-in for two groups per assignment. The online learning platform was used to give feedback on the hand-in's. This was written feedback and was given every other week.

[1]https://www.limesurvey.org

Table 8.1: *Overview of students' post assignment questions – non free and free text*

Non-free Text Questions
During the assignment
Participation in supervision sessions
Q1: *Did you participate in supervision?* [Yes ∣ No]
Students' opinion about the TA support
Q2: *How do you feel about the support of the teaching assistant(s) during this assignment?* [Likert scale 1-5: not helpful - very helpful]
Difficulties with which the students were helped by the TAs.
Q3: *With what difficulties did the teaching assistants help you with?* (for each assignment a more specific, task related, version) [list of choices, multiple answers allowed]
After handing in the assignment
Usage of the feedback by the students
Q5: *Did you look at the feedback the teaching assistant gave with the grade?* [Yes ∣ No]
Opinion about the learning fullness of the TA support
Q6: What do you think about the feedback of the teaching assistant after grading? [Likert scale 1-5: did not learn - learned a lot]
Free Text Questions
During the assignment
Q1: *Can you provide detail about how a teaching assistant helped you out with a specific topic?* (for each assignment a specific, task related, version) [free text]
After handing in the assignment
Q2: *Can you give an example of feedback of the evaluation that really helped you to understand the topic better?* [free text]

8.3.3 Course and Assignments

The course that was used for the case study was a 1st year bachelor's object-oriented analysis and design course. The course targets analysis and design techniques using UML. The typical UML diagrams that were taught were: Use Case Diagram, Class Diagram, Sequence Diagram and State Machine Diagrams. The course consisted of lecture hours (2 hours per week) and supervision hours (2 hours per week). During supervision students can ask questions about the practical assignments they have to hand in and also about the previous lectures.

Three of the practical assignments were based on a software development project that was running in another course in the same teaching period. The fourth assignment was part of only the analysis and design course. The practical assignments had to be handed in every other week using an online learning platform for submission.

The case that was handled in the project course was a complex case (development of a robot car control system) that required a large amount of modelling. For the practical assignments the students delivered reports in which they presented their UML models for the different development phases of the software development project. In total there were four assignments:

- Assignment 1: Use Cases and Domain modelling with class diagrams – Students create an analysis report, using use case diagrams and class diagrams, that explained and motivated analysis decisions.

- Assignment 2: System Sequence Diagram and Use Case Realisation – Students create an initial design report, using sequence diagrams. Students motivated their design decisions.

- Assignment 3: State Machine Diagram and Sequence Diagram Simulation – Students create a report that focused on detailed behavioural design and testing by simulation.

- Assignment 4: Refactoring – Students delivered a report covering the refactoring of a game. The source code was given. The students had to redesign the reverse-engineered class diagram. The new design should be improved by using design patterns. Eventually the students implemented the code and delivered a running game.

8.3.4 Participants

Students – the students that participated were 1st year bachelor students following the 2nd semester of their SE program. In the 1st semester they learned about object-oriented programming in Java. The students worked in a group of 6 or 7 students during the whole course. During the course 10 groups were active.

TAs – Five teaching assistants were elected from the second year of the bachelor program. They were selected based on their grades for prior courses and information from the lecturers that experienced them in the classroom. TAs should apply for the position. The TAs task was to support the students during supervision hours and give feedback on the hand–in's that students uploaded on the online learning platform. They receive a small payment for the job.

Lecturers – 2 lecturers were involved, teaching the course and having the weekly interviews with the TAs.

8.3.5 Data Analysis

For analysing the data we used both a quantitative and a qualitative approach. First, for the survey questions, we used descriptive statistics for analysis. Second, the open ended answers on the survey, the transcriptions from the TA interview meetings and the written TA feedback on the student hand-ins were analysed by labelling the text using codes. The codes were distilled from proof reading the recorded text with the research questions in mind. The coding was done by one person. Examples of these codes were: *challenges using TA* (challenges identified during TA supervision), *observation* (remarkable observation) and: *approach* (approach a TA uses to help a student). After labelling we merged the labelled text from the different sources (survey,interview, TA feedback) into categories. The results of both approaches are presented in Section 8.4.

8.4 Results

In this section we present the results we obtained from the student survey, the written teaching assistant assignment feedback, and the interviews with the TAs. First we present the descriptive statistics. Second, we present the data for the open ended questions in the form of tables organised in categories.

8.4.1 Descriptive Statistics

In this section we show the statistics that were distilled from the non-free text questions in the student survey (Table 8.1). After each of the 4 assignments we conducted a post-assignment questionnaire. There were 3 moments in time. Due to scheduling challenges, the questions of assignment 3 and 4 were combined into 1 questionnaire. Each questionnaire consisted of a set of similar questions. A questionnaire consists of a part *during the assignment'* and a part *after grading*. We present the results of the 3 questionnaires below in order of the assignments. The number of responses (N) can differ. This is because of questions where multiple answers were allowed or where students skipped questions.

Assignment 1 – After assignment 1 36 students (N=38) responded that they participated in the supervision sessions. Most students (N=29) value the support of the TAs higher than 2 (neutral or helpful) in this assignment (see Figure 8.2).

Table 8.2 shows the challenges where students asked for help from a TA for the Use Cases tasks of assignment 1. Table 8.3 shows the challenges for the domain modelling with class diagrams task.

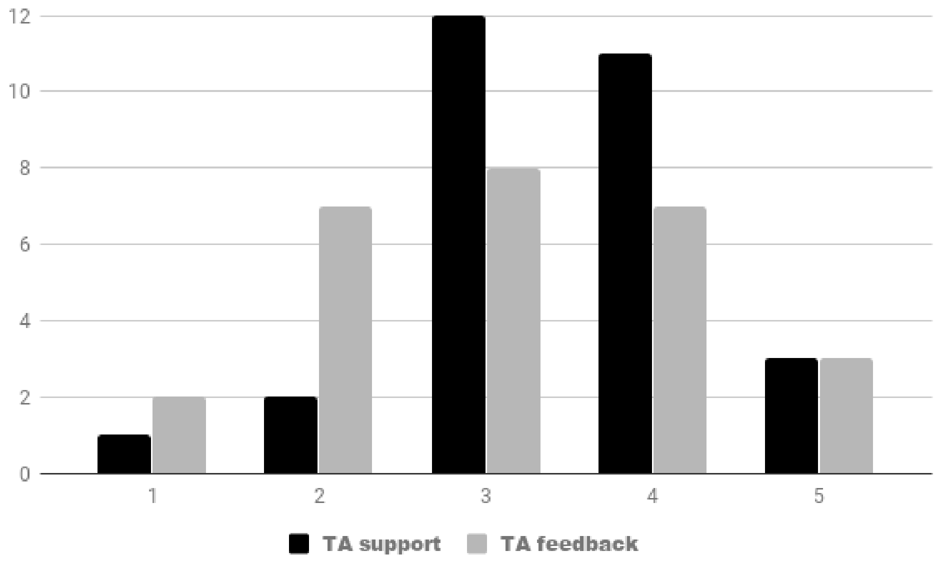


Figure 8.2: *Students' judgement about the quality of the TA support and quality of the TA feedback for assignment 1. A 1 indicates low quality, a 5 high quality.*

Table 8.2: *Difficulties During the Use Case task – assignment 1*

Difficulty	frequency
Apply Include or Extend	11
How to apply an include or extend relationship in a use case diagram	
Finding Use Cases	8
Finding use cases from the assignment's case text	
Finding Domain Concepts	4
Finding concept from the domain from the assignment's case text	
Finding Actors	4
Finding actors from the assignment's case text	
Using the Tool	3
The use of the UML modelling tool	
Other	10
Removing actors from the diagram	
Understanding the tasks and what to do/hand in (7x)	
Reviewing the diagram	
Deleting some extra actors from the diagram	

Table 8.3: *Difficulties During the Domain Modelling Task (Class Diagram) –assignment 1*

Difficulty	frequency
Finding the problem domains	6
Determining the problem domains in the assignment	
Respresent concepts with classes	5
Determining the main concepts and represent them in the diagram	
Finding Relationships	9
Finding relationships between concepts from the assignment's case text	
Choose Between Class or Attribute	5
Determining whether something is a main concept or property of a concept	
Finding attributes	1
Finding operations	1
Using the tool	2
The use of the UML modelling tool	
Other	4
Explained how thorough it was supposed to be	
Understanding the assignment	
Understanding that in this case they represented hardware connections	
The overall finished diagram needed help to be refined	

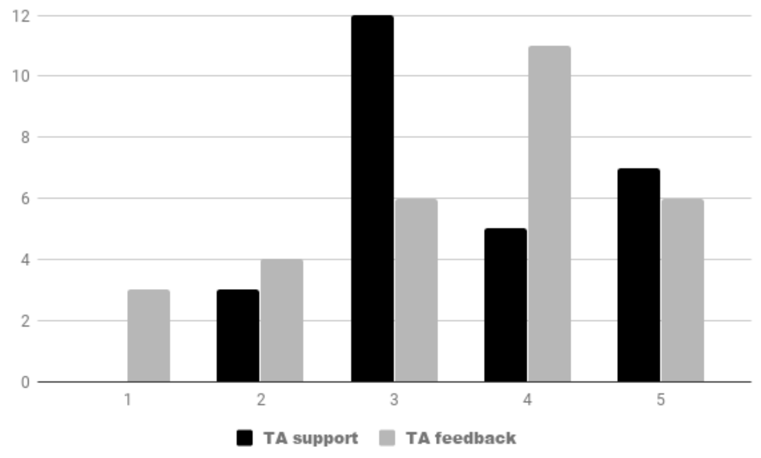


Figure 8.3: *Students' judgement about the quality of the TA support and quality of the TA feedback for assignment 2. A 1 indicates low quality, a 5 high quality*

Table 8.4: *Difficulties During the System Sequence Task – assignment 2*

Difficulty	frequency
Seeing the system as a black box	10
Making a top level diagram	9
High abstraction at system level without implementation detail	
Use case to sequence	8
Identify sequences from the use case that was distilled from the case	
Using the tool	8
The use of the UML modelling tool	
Finding methods	7
(A)synchronous messages	4
When to use (a)synchronous messages	
Apply reply messages	4
When to use reply messages	
Finding the central lifeline	3
Identify the central lifeline that represents the system	
Other	3
what was exactly required (2x)	
overall structure	

Table 8.5: *Difficulties During the Use Casse Realisation Task – assignment 2*

Difficulty	frequency
Finding the important objects that collaborate in a use case realisation	11
Identify concepts that play a role in the solution (design)	
Finding methods for internal system sequence	6
Assigning responsibilities to the concepts that were found	
Creating an Initial Design	6
Updating the Initial Design	6
Finding relationships	5
Having a white box view	5
Understanding how the internal system should work	
Apply Reply Messages	4
When to use reply messages	
Using The Tool	3
The use of the UML modelling tool	
(A)synchronous Messages	1
When to use (a)synchronous messages	
Other	3
Deciphering requirements	
Understanding the purpose of the diagram fully and its abstraction level	
Explaining what they were	

After grading, 25 of the students (N=28) responded that they looked at the feedback of the TA. In addition, 2 were informed about the feedback by a group member. They (N=27) seem to be more critical on judging that they learned from the TA feedback, compared to their judgement about TA the support. On average they judged neutral (see Figure 8.2).

Assignment 2 – After assignment 2, 37 students (N=42) responded that they participated in the supervision sessions. Most students (N=27) value the TAs neural or helpful (>2) in this assignment. (Figure 8.3).

Table 8.4 shows the challenges were students asked for help from a TA for the System Sequence Diagrams task of assignment 2. Table 8.5 shows this for the Use Case Realisation task.

27 students (N=30) stated that they read the feedback on the assignment. In addition, 3 were informed about the feedback by a group member.

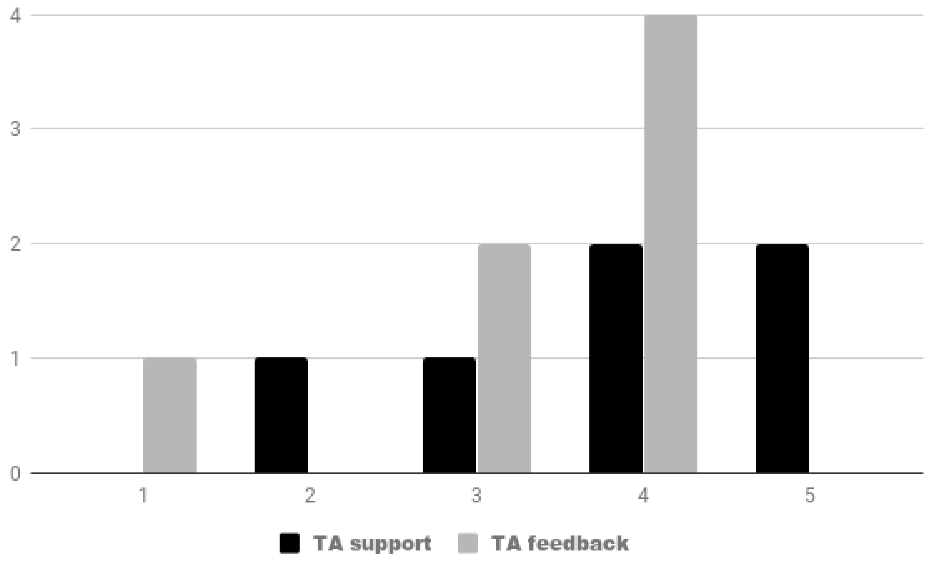


Figure 8.4: *Students' judgement about the quality of the TA support and quality of the TA feedback for assignment 3. A 1 indicates low quality, a 5 high quality*

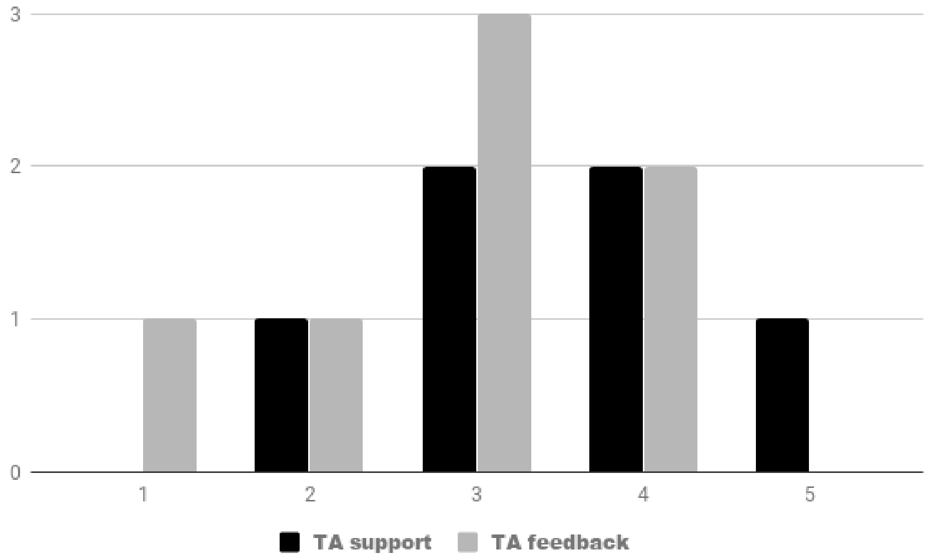


Figure 8.5: *Students' judgement about the quality of the TA support and quality of the TA feedback for assignment 4. A 1 indicates low quality, a 5 high quality*

Table 8.6: *Difficulties State Machine Diagram–assignment 3*

Difficulty	frequency
Identify State behaviour	3
Recognize state behaviour from the assignment text	
Identify events	2
Recognize events from the assignment text	
Identify States	1
Recognize states from the assignment text	
Identify Transisitions	1
Recognize transisitions from the assignment text	
Apply Guards	1
When to use guard notation (for conditions)	
Apply Proper Naming	1

Table 8.7: *Difficulties During the Sequence Diagram Simulation Task – assignment 3*

Difficulty	frequency
Using (A)synchronous Messages	3
When to use (a)synchronous messages	
Identify Important Methods	2
Recognize implementation responsibilites in the design	
Apply reply messages	1
When to use reply messages	
Identify Collaborating Objects	1
Recognize implementation concepts for the design	
Placing States on Lifelines	1
Adding state behaviour to a sequence diagram	

Most students (N=27) value the TAs' support as helpful or as very helpful (>3) in this assignment. (Figure 8.3).

Assignment 3 and 4 – The survey for assignment 3 and 4 were conducted combined. 10 students (N=14) confirmed they have participated in the supervision hours. 6 students judged the support during the supervison hours of assignment 3 or 4. Most judgements were neutral or positive. There was no very negative judgement (1) (Figures 8.4 and 8.5).

Table 8.6 shows the difficulties during the State Machine Diagram task. Table 8.7 for the Sequence Diagram Simulation task. Table 8.8 shows the challenges were students

Table 8.8: *Difficulties During Refactoring–assignment 4*

Difficulty	frequency
Apply a Class Design Pattern *How to apply a design pattern on class level (non-architecture level)*	3
Apply an Architecture Design Pattern	1
Getting Runnable Code *Update the source code based on design, compile and run*	1
Other *Suggestions for further patterns that could be applicable with our current one*	1

asked for help during the Refactoring task.

7 students (N=8) stated to have read the feedback on the assignment. Most students (N=7) value the TAs' support neutral or helpful (>2) in both assignment 3 and 4. The students judged that they have learned more from the TA feedback of assignment 3 (Figures 8.4 and 8.5).

8.4.2 Open Questions, TA Feedback and Interviews

This section presents the answers to the open questions in the students' post questionnaire, the written TA feedback on the hand-in's of the students and the data that was distilled from the interviews we had with the TAs. This section shows results on mainly two matters: i) an inventory of common difficulties were TA support was involved and ii) explore what type of approaches on guiding and feedback are used by the TAs.

Table 8.9 shows an overview of the difficulties that were mentioned by the students in the questionnaire (free text in Table 8.1). Next to what the students mentioned, we also list the topics that the TAs observed.

Table 8.10 shows a categorised list of the approaches that were used by the TAs during the 4 different assignments. An 'X' indicates the presence of that particular approach.

Table 8.9: *Overview of common difficulties mentioned in free text – an 'x' indicates that the students mention the difficulty in the text of the survey responses or that the TAs mention this as an issue in the interviews or the feedback*

	Student				TA			
Common topic	A1	A2	A3	A4	A1	A2	A3	A4
P Process / Motivation / Confidence								
P.1 Confirmation of direction	x							
P.2 Clarification of task and deliverable	x							
P.3 Individual participation					x	x		
S Software Development Related								
S.1 Analysis vs Design	x	x						
S.2 User Stories vs Use Cases	x				x			
U UML Related								
U.1 Abstraction / level of detail	x	x		x	x	x	x	x
U.2 Structure of diagram	x				x			
U.3 Responsibility (identify elements)	x	x	x		x	x	x	
U.4 Use of notation / concepts	x	x			x	x	x	x
U.5 Layout					x	x		
U.6 Pre / Post conditions					x	x		
U.7 (In)consistency			x			x	x	
D Deliverable / reporting								
D.1 Improve documentation	x				x	x	x	x
D.2 Clarification of task / deliverable	x				x	x	x	x

Student = from student questionnaire - TA = from TA interviews and written feedback
An = assignment n

8.4.3 Mapping Students' Difficulties and TA Approaches

In this section we relate the common difficulties that students have to the approaches TAs use to help students.

From the student questionnaires we identified the common difficulties (Table 8.9). The TA approaches were distilled from the interviews and the written feedback to the students (8.10). We present a mapping in Tables 8.11 and 8.12.

The rows in Tables 8.11 and 8.12 represent students' difficulties and the columns correspond to TAs approaches. An 'x' in a cell indicates that we see a potentially use for a TAs approach for a particular difficulty a student has. A 'ⓧ' indicates a difference between the students view and the TAs view. For example, when a 'ⓧ' occurs in the TA view this means that a certain TA approach is not mentioned by the student, while

Table 8.10: *Overview of the different approaches TAs use*

TA Approach Related	A1	A2	A3	A4
I Informative				
I.1 Invite students for next activity	X			
E Examples (comparison/clarification)				
E.1 Explain problem with the use of an example	X	X	X	X
E.2 Showing study material (slides, book, etc.)		X		
E.3 Suggest to read literature, slides and instructions		X		
E.4 Compare with programming		X		
E.5 Rephrase the assignment grading criteria		X	X	
E.6 Let the student think aloud			X	
G Guiding (not directly give the answer)				
G.1 Suggest direction for improvement	X	X	X	X
G.2 Point out errors/omissions	X	X	X	X
G.3 Point out particular notation features	X	X		
G.5 Ask questions to get student in right direction	X	X	X	
G.6 Give confirmation to student when on the right track		X		
G.7 Discuss the problem in an intense way		X		X
G.8 Suggest to use the richness of UML to improve readability			X	
G.9 Tips on how to approach the assignment				X
A Active / passive support				
A.1 Approach student actively by asking questions	X			
A.2 Invite student to contact a TA or lecturer when insecure		X	X	
A.3 Send extra feedback after group meeting		X		
R Revealing the student's problem				
R.1: Backup with other TA	X			
M Medium used				
M.1 In person communication		X		
M.2 Written feedback		X	X	X
C Character of TA				
C.1 Always willing to help out		X	X	X
C.2 Pointing out the TA's job is support not to solve the problem		X		
C.3 Thorough Explanation			X	
C.4 Make assignment him/herself for understanding the student				X
T Time				
T.1 Suggest follow up meeting after evaluation feedback	X	X	X	X
T.2 Look at deliverable before actual hand-in	X			

An = assignment n

Table 8.11: *Mapping of difficulties and possible approaches – student view*

	E1	E2	E3	E4	E5	E6	G1	G2	G3	G5	G6	G7	G8	G9
P1	ⓧ						ⓧ	ⓧ	ⓧ	ⓧ				
P2	ⓧ									ⓧ				
P3														
S1	ⓧ	ⓧ	ⓧ	ⓧ			ⓧ	ⓧ	ⓧ	ⓧ	ⓧ	ⓧ		
S2	x						x	x	x	x				
U1	x	x	x	x			x	x	x	x	x	x		
U2	x						x	x	x	x				
U3	x	x	x	x		x	x	x		x	x	x		
U4	x	x	x	x			x	x	x	x	x	x		
U5														
U6														
U7	x					x	x	x		x			ⓧ	
D1	x						x	x	x	x				
D2	x						ⓧ	ⓧ	ⓧ	x				

Table 8.12: *Mapping of difficulties and possible approaches – teaching assistant view*

	E1	E2	E3	E4	E5	E6	G1	G2	G3	G5	G6	G7	G8	G9
P1														
P2														
P3	ⓧ	ⓧ	ⓧ	ⓧ	ⓧ		ⓧ	ⓧ	ⓧ	ⓧ	ⓧ	ⓧ		
S1														
S2	x						x	x	x	x				
U1	x	x	x	x		ⓧ	x	x	x	x	x	x		
U2	x						x	x	x	x				
U3	x	x	x	x		x	x	x		x	x	x		
U4	x	x	x	x		ⓧ	x	x	x	x	x	x	ⓧ	
U5	ⓧ	ⓧ	ⓧ				ⓧ	ⓧ	ⓧ	ⓧ	ⓧ	ⓧ		
U6	ⓧ	ⓧ	ⓧ	ⓧ	ⓧ		ⓧ	ⓧ	ⓧ	ⓧ	ⓧ	ⓧ		
U7	x	ⓧ	ⓧ	ⓧ		x	x	x	ⓧ	x	ⓧ	ⓧ		
D1	x	ⓧ	ⓧ				x	x	x	x	ⓧ		ⓧ	
D2	x	ⓧ	ⓧ		ⓧ					x		ⓧ		ⓧ

from the TA's perspective the approach is actually identified.

We mapped common difficulties with the approaches of categories E (Examples) and G (Guiding) because those categories are more specific about how the approach was performed.

When looking at difficulty D2 across tables, we notice that students seem to have a preference to behave passively, while the TA is concrete suggestions of next steps. Examples of these approaches are G1 and G2. In these cases students aim to finish the task without to much mental effort. Instead, the TAs are expected to literally state the next step. In contrast, the TAs do not mention approaches were they provide concrete solutions to be helpful.

When looking at difficulties D1 and D2 across tables, we find that TA's use approaches in which they try to motivate and activate students, such as E2 and E3. For example, they provide suggestions for reading or studying a particular topic rather than concrete solutions. The TAs also mention that discussing the problem helps. During a discussion the TA and the students elaborate a certain topic. Instead of leaving the group after giving them guidance, the TA stays for discussing a particular topic.

When comparing tables on difficulty U7, a technical oriented challenge, it can be noticed that from a students view the TA approaches E2, E3 and E4 are not mentioned as helpful. This is contrast to what the TAs have used or discussed in the interviews to be helpful. E2, E3 and E4 are TA approaches in which examples are used to guide the student further in the assignment. The students don't mention this as helpful support or wanted to have more concrete answers.

Also across tables, we notice that at difficulties P3, U5 and U6 students did not made remarks about any supporting TA approach. This appears to be in contrast to what the TAs intended.

8.5 Discussion

In this section we discuss with the aim to answer the research questions listed in the introduction of this chapter.

8.5.1 What are typical challenges in 'Software Analysis and Design' for which students seek TA support? (RQ1)

Based on the results from the questionnaire we identified the challenges where students asked for TA support. On one hand, it seems that the most dominant topics are the ones that require more abstract reasoning, such as 'Finding relationships' or 'Seeing the system as a black box'. On the other hand, it is worthwhile to mention two other matters. First, in the 'other' option students often mention that they needed help to 'understand the task' or 'what we were supposed to do'. The students, who are novice in software designing, need guidance and confirmation. They expect the TAs to pick up this (lecturer) role. Second, we noticed that the 'using the tool' option was not chosen that much. From research [2] we know tools can frustrate students a lot (installing, configuring, crashes etc.). We did not put special attention to the use of (industrial) tools [83]. Therefore we expected to have collected more tool-related issues. It could be that the TAs, that were walking around during the supervision sessions, helped with small tool use problems at a time and it was not noticed as a TA support challenge in the questionnaire.

From the analysis of the open ended questions we found similar challenges. Although these challenges were more described in an abstract way than specific. For example: 'Use of notation' versus 'Apply guards'.

We noticed differences between which challenges the TAs noticed and which ones the students noticed (RQ1.1). Table 8.9 shows, that for some of the topics, the TAs kept noticing the need for support. At the same time the students do not notice this in the open questions in the questionnaire. Although students do not mention this in their feedback, this does not mean they did not receive support for this topic. It still gives the impression that TAs try to continue focusing on important matters such as 'Improve documentation' and 'Use of notation', while at the same time the focus of the students seem to shift.

We did not find any unexpected challenges or reasoning, such as matters that wouldn't arise in a lecturer-student conversation. The identified challenges and the fact that students seek for confirmation, confirm and extend our previous research [132]

8.5.2 What kind of didactic approaches do TAs use to support the challenges students have? (RQ2)

From the free text answers on the questionnaires and the transcribed interviews with the TAs we were able to identify and categorise approaches TAs use when helping out students during the supervision sessions as presented in Table 8.10. We see that the guiding approaches are quite dominant in every assignment. Especially 'suggestion of direction for improvement' and simply 'point out errors' are mentioned in every assignment as helpful. This is in line with findings from Hattie et al. They describe feedback to be most useful when one combines direct task feedback with feedback that is focused on the process [49]. As most of the approaches could be expected in different types of programs (i.e. non IT), we see that typically 'explaining with the use of an example' is useful for SE students (RQ2.1). This also counts for the approaches that help the students 'in the right direction' We see software design as an iterative activity. It is important that we, and in this case the TAs, help students out by reflecting on their intermediate results. Especially novices are insecure about their solutions and need some or more guidance in the design process.

There seems to be a mismatch between the activating approach the TA most of the times chooses and the more passive attitude that a student can have. While TAs try to encourage and activate students by showing examples and pointing to lecture materials, the student wants to have a 'yes, this is good' or 'no, this is wrong' answer. This is probably due to the novice stage of the student. This means that a student is not yet capable of associating the fresh material with examples or other sources. The TAs themselves give a suggestion to cope with this problem. They mentioned discussion helped the students to discover bits of the solutions. Maybe a brief discussion prior to suggestions for further reading or looking at examples could be a good approach. We believe that the TAs' active learning approaches could benefit students' learning. Freeman et al. state active learning increases exam performance [39].

8.5.3 Which expectations do TAs have about students and vice versa?(RQ3)

From the results that summarise the attitude of the TAs and the students we identified dominant issues that are addressed from both the TAs and the students.

First, from the TAs perspective we noticed that TAs mention different matters about the study behaviour of the students. They noticed the students do not use the scheduled supervision time to it's full extent. They expect students to take part in all the supervision hours. They explain it by the fact that "students underestimate the time" and that some students "are more into programming at the moment". The TAs were critical on the students by noticing some students had questions that should have been addressed in the lecture and stated they did not visit that lecture. TAs expect students to elaborate more on possible solutions before saying something is good or bad. In some cases TAs also had difficulties stating something was good or bad because of the variety in possible answers. With respect to the time spend, the TAs noticed students allocated more time half way the course.

Second, from the students' perspective, the students mention that they have difficulties with the different answers TAs give when asking for help. For the students the TAs sometimes could have different interpretations of the assignment goals. This seems to be typical for software engineering. Also students are critical on the feedback of the TA. They have a need that TAs say if it is good or bad. While at the same time the TA tries not to give them the answers right away.

8.5.4 Do feedback and guidance of the TAs focus on the same matters as the lecturers?(RQ4)

The students' difficulties that the TAs act on (Table 8.9) and approaches TAs use (Table 8.10) in this study reflect on what was discussed in the weekly meetings and instructions. There seems to be a shared focus with the lecturers. Interestingly for some topics there was no particular instruction, but they seem to be important for the TAs. We also find these topics of importance. The TAs mention the following:
Layout TAs value to spend time on the quality of the layout of a diagram. For them to give proper feedback, it is necessary to have a layout that is easy to read.

UML Syntax Students seem to not look into syntax / read about UML (because of less questions about it).

Prior programming knowledge TAs explained the root of some student struggles:"Probably students make mistakes because of knowledge of Java classes." In our previous research [132] we also found evidence that prior programming knowledge also can lead to difficulties in understanding software design.

Quality of Reporting although instructed properly and discussed during supervision TAs notice the quality of the reports.

Responsibility TAs mention different matters that worry them in relationships to the students' progress: the difference in individual participation; a coming deadline while students focus on - in their eyes - wrong priorities.

8.6 Recommendations for Future Deployment of TAs in SAD Courses

In this section we present recommendations for future use of TAs in courses similar to the 'Analysis and Design' course that was observed. Based on the analysis we performed in this research we list several recommendations.

8.6.1 Elicitation / TA Profile

When selecting the TAs lecturers should be aware of the different skills that are needed next to the technical (UML) skills. As Table 8.10 presents, TAs use different methods to support the different questions that come from the students. Having a mix of TAs that use different approaches is an ideal situation. Interview TA candidates and address the different approaches. For example TAs could react on challenges in a mini case.

8.6.2 Training and Support

We endorse the need for training TAs [30]. In our opinion, training depends heavily on the level of the course, the TAs and what (educational) task the TAs should fulfil. From a course content perspective one could use and extend our categorised challenges that are listed in the figures in Section 8.4 Table 8.9. From an approach perspective Table 8.10 can be extended and used to train student support approaches.

We recommend to support the TAs during the course by having weekly meetings and discuss student problems, solve organisational issues and discuss example solutions.

8.7 Threats to Validity

In this section we discuss the threats to validity of this research.

Construct Validity Closed ended questions. The questionnaire could be biased because of the pre-filled question answers. We tried to avoid this by adding an 'other' option amongst the answers.

Internal Validity Recorded free text and transcribed interviews. It is not proven that we recorded everything that was relevant. While answering the questionnaires, students may have forgotten some details of the TA interactions that could have been significant to report. Those issues can possible be missed. Also there could be an interpretation validity threat when we were analysing the data because of incomplete or unclear student statements.

External Validity Matters such as cultural background, student level (TA and course student) and the university's methods, could influence the generalisability of this study.

We plan to address these threats as we refine the study framework in future studies.

Conclusion Validity We base our conclusion on the evidence that was distilled from the questionnaire (closed ended and open ended questions) and interviews. There is a risk of not having recorded all the relevant information. We tried to avoid this by having multiple sources that contain relevant information. Most of the observations were found in multiple sources.

8.8 Conclusion

This chapter reflected on the use of TAs in order to: i) reveal interventions (approaches) TAs commonly use ii) extend our knowledge about common difficulties students have with analysis and design topics and iii) what believes are shared and valued useful by TAs and lecturers.

By continuously observing the TAs and students during an analysis and design course we were able to distil and categorise the common challenges students have recorded where TAs supported them. Further we have an overview of the typical interventions TAs use when helping students during their assignment tasks.

With this research we now have an approach that can be used to further research the use of TAs. What is learned can be used for improving training programs for TAs. Future work comprises further studies with other instances of the courses, and correlate with connected courses that use OO analysis and design knowledge. Moreover, we plan to investigate whether the same findings can be applied to TAs of different courses in Software Engineering.

Chapter 9

Towards Automated Grading of UML Class Diagrams with Machine Learning

Description of contributions of the authors
The research in this chapter was done in collaboration with Caroline Sperandio. At the time of writing Caroline did a research project on machine learning as guest student from ESIEE Amiens, France. Michel Chaudron and Dave Stikkolorum supervised her. The experiment was designed and conducted in collaboration. With the support of Peter van der Putten, Dave re-analysed the data and wrote the publication.

This chapter describes an exploratory study on the application of machine learning for the grading of UML class diagrams. Bachelor student pairs performed a software design task for learning software design with the use of UML class diagrams. After experts had manually graded the diagrams, we trained a regression model and multiple classification models. Based on the results we conclude that prediction of a 10 point grading scale can't be done reliably. Classifying with trained data using expert consensus and a rubric increases the performance, but is still not good enough (a precision of 69%). Future work should include larger training sets and an explore other features.

This chapter is based on the following publication: Dave R. Stikkolorum, Peter van der Putten, Caroline Sperandio, and Michel R.V. Chaudron. Towards automated grading of uml class diagrams with machine learning. In *BNAIC/BENELEARN*, 2019

9.1 Introduction

Many university programs, including ours, support a learning by doing approach for learning software design. Software design is known to be a discipline that needs to facilitate students with a lot of of exercise time. For large groups of students, lecturers face an enormous amount of assignments to grade. In classical lectures, accompanied with practice labs, 200 students or more are not an exception. Grading a single class diagram can easily take 10 to 15 minutes per student. Hence, the grading of practical assignments and running practice labs has a high demand on the presence of lecturers and teaching assistants. Moreover, in cases of massive online courses, manual grading is not an option anymore. Overall automated evaluation can decrease the workload of lecturers.

In addition, it is a challenge for teachers to provide students with immediate feedback when the need arises. Moreover, most software design assignments do not have a single right answer. There are multiple solutions that solve a particular problem. Novice software designers lack the experience to recognise the solution space. Therefore it is difficult for students to perform self-evaluations of their solutions. This is a pity, because self-evaluation, supported by automated evaluation, would enable students to practice at home more often in addition to their university hours, at their own pace. Further, it enables students to take small steps while practising their task. In summary, a system grades students' design assignments would decrease the demand on the lecturers' time and potentially offer faster feedback.

This chapter reports on an early exploration of the possibility of using machine learning for the automatic grading of students' work. The main research question is: How can machine learning contribute to the automated grading of software design class diagrams?

We approached our research as follows. Student pairs were given a UML class design task, and in total 99 pairs were graded by 3 experts. Features were extracted from these UML models. These features were input to machine learning algorithms which produced models for producing grades. To our knowledge the research in this chapter is the first to study how machine learning could be used to grade UML class diagrams that for software designs. Given that this work is exploring new avenues, our preference is to first try out a simple approach over using more technically sophisticated approaches.

The remainder of this chapter is structured as follows: Section 9.2 discusses related work. Section 9.3 presents the research method we used. The results are presented in Section 9.4 and discussed in Section 9.5. Section 9.6 concludes the chapter.

9.2 Related Work

For the automated evaluation of software analysis and design assignments with the use of UML modelling there are several approaches that use an example solution for comparison with the student's solution.

Hasker et al. present their tool: *UMLGrader* [47]. This tool compares student solutions against a standard solution. In this approach, students are offered a very constrained assignment, and therefore minimise the solution space. By repeating their submissions students are guided towards an acceptable solution. Despite the name, the UMLGgrader systems does not grade, but only produces a binary pass-fail decision. The grading is based on predefined criteria about the solution and does not employ machine learning. *UMLGrader* is based on UMLint, a UML diagram defect identifier [48], especially developed for students to learn the UML notation and does not focus on issues of software design [47].

Striewe et al. present a static analysis approach for automated checks on UML diagrams with the help of rules that are executed on graphs that represent the content of a diagram [146]. The focus of this work is on behavioural diagrams, which contrasts with our focus on structure diagrams. Their approach for evaluating activity diagrams is described in [147].

Bian et al. propose a grading algorithm that uses syntactic-, semantic- and structural-matching against a template solution [17]. Their approach uses meta-models to map the teacher's solution with the student solutions and to grade model elements. In a case study, 20 students were automatically graded with a 14% difference from the lecturer's grades.

In our own research with our educational UML tool WebUML [136], we introduced a pedagogical feedback agent [6]. The feedback agent also compares the solution with a standard solution. We use synonyms for providing some flexibility. We did not use the feedback agent for grading but for guiding feedback during software design tasks.

For the automated evaluation of student solutions for programming assignments there are examples of the application of machine learning. A big advantage of using source code is that it can be tested. This is not the case for software designs when using a non-executable specification language, such as UML.

Srikant et al. demonstrated an approach for grading practical assignments for programming assignments [127]. Manual grading was done based on a 5 points scale rubric. The precision range of their research results is 55%-77%, based on experiments with a sample size of 84-294 students. Based on the findings of Srikant et al., Boudewijn explores the use of machine learning for predicting the manual grading of Java assignments in relation to the automation of hiring processes [21]. In addition to Srikant et al. semantic features were added to the test set. She did not find this to be significant better than without the semantic features. However, it seems that the features have a positive influence on the test case accuracy.

To our knowledge, tools for the automated grading of software design assignments are scarce and there are no tools that use machine learning for predicting the grades.

9.3 Method

This section describes the method we used for our research. First we present the overall research approach. Second, we present the participants, explain the students' design task and the manual grading process of the assignments. Subsequently we explain the features that were extracted from the data we use. We close with explaining how we build and analyse our models.

9.3.1 Overall Approach

We performed two types of grading approaches: one in which the graders divided the grading work (experiment 1) and one in which the graders individually looked at all the work based on a rubric and had to come to a consensus (experiment 2).

Figure 9.1 shows the overall approach which consisted of the following steps:

1. Task Performance – Students performed a practical learning task, designing a class diagram for a game.
2. Expert Grading – three experts manually graded the produced UML class diagrams. For experiment 1 the experts divided the workload by grading their own part of the student submissions. For experiment 2 all experts graded all submissions and a final grade list was composed based on the consensus of the experts.
3. Defining and extracting the features – Next to an image representation, the class diagrams' actual values (e.g. class name, attribute name, multiplicity value etc.) and the UML element type themselves (e.g. class, operation, association etc.) were recorded. A selection from these characteristics was made and used as features for conducting the machine learning experiments.

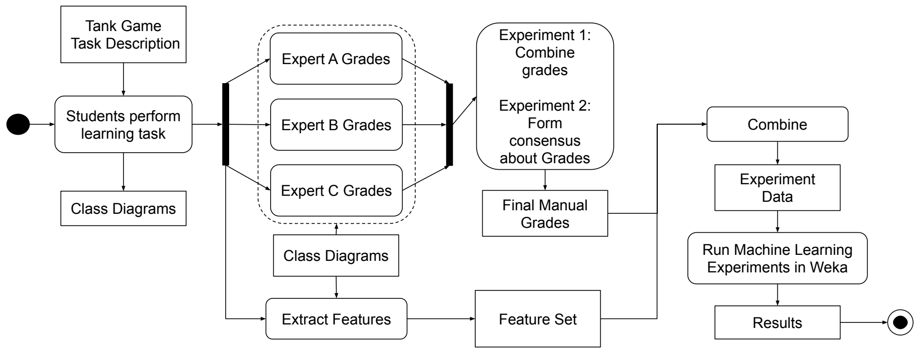

Tank Game Task Description
Students perform learning task
Class Diagrams
Expert A Grades
Expert B Grades
Expert C Grades
Class Diagrams
Extract Features
Experiment 1: Combine grades
Experiment 2: Form consensus about Grades
Final Manual Grades
Feature Set
Combine
Experiment Data
Run Machine Learning Experiments in Weka
Results

Figure 9.1: *Overall Research Approach*

4. Experiments with various machine learning algorithms – In Weka's Experimenter we ran classification and regression experiments with the use of several well known methods and algorithms, such as the random forest algorithm.

9.3.2 Participants

We invited 120 student pairs to perform a class design task. The students followed the third year of a bachelors program in Computer Science. Through prior courses, they were familiar with programming and had basic understanding of UML and software design principles. The lecturers of the university embedded our experiment in the practical part of their course. The students were used to working in pairs. The cleaning consisted of removing incomplete assignments and outliers. After cleaning the data, 99 pairs of students remained.

9.3.3 Assignment

All participants worked on the same assignment. The assignment was to model a class diagram for a tank game (Appendix C). The solution contains 10-12 classes using attributes, operations, named associations, inheritance and multiplicity. The assignment is chosen because it is representative for the students' study level.

The class diagram is created online with our WebUML editor [136]. The editor is capable of saving the diagram to a XMI file, a standard file format for UML diagrams, and an image file in png[1] format.

[1] https://fileinfo.com/extension/png

Table 9.1: *Conversion table grading*

10 points	1	2	3	4	5	6	7	8	9	10
5 points	1		2		3		4		5	
3 points	fail				pass			good		

9.3.4 Grading

Three experts, one lecturer and two PhD students, performed the grading of the students' work. The grading scales that were used, were a 10 points[2] (experiment 1) and a 5 points (experiment 2) scale. The reason to choose for the exploration of different grading policies is that we anticipated that we could not expect that a machine learning model will be as accurate as the precision of a 10 points scale.

For experiment 1 an expert graded a part of the students' work. The three expert grades were combined later. In addition, for comparison with experiment 2, the grading of experiment 1 was done with two extra scales: a 5-point – and a 3-point (out of curiosity) scale. The extra scales were derived from the 10-point grades, and were not the result of an independent grading activity. For all the scales, see the overview in Table 9.1.

For experiment 2 the experts graded based on a rubric that uses a 5 point scale (Appendix D) and formed a consensus grade later. Consensus about the grade was discussed live or by (video) conference calls. There is no 3 points scale translation, because in experiment 1 this is derived from the 10 points scale, which is not used in experiment 2.

9.3.5 Feature Extraction

The features that are used for training our model are extracted from the XMI file that contains the student's solution. Together with the experts' grades the features are stored in a database. In this research we used two type of features:

- a generic set, which focused on the presence of important elements and the use of specific structures (such as e.g. inheritance or aggregation)
- a specific set that focuses on the specific use of elements that carry a specific name (or variation thereof).

[2]This is the conventional scale for grading exams in The Netherlands.

The data sets that were used for building prediction models consisted of a feature set in combination with one of the grading scales used as classifier. A feature set consisted of all features (generic + specific) or only the generic features. The features are listed in Table 9.2.

9.3.6 Modelling

For analysing the data and running the machine learning experiments we used Weka[3] (version 3.8.3). We built classification and regression models using a range of algorithms. We evaluated the classification models using accuracy and AUC, through ten runs of five and ten fold cross validation (five fold results omitted for brevity). The regression models were evaluated by analysing the mean absolute error (again, five fold results omitted for brevity).

9.4 Results

First, we present the distribution of the grades for the various scales. Next, we will present the results for the various classification and regression models. For both the classification model-table as well as the regression models, the results are compared with majority vote and predicting the overall average (ZeroR), as a baseline benchmark for the other models.

9.4.1 Grade distribution

Figures 9.2, 9.3 and 9.4 show the distribution of the grades for experiment 1 using a 3-, 5- and 10-point scale. Figure 9.5 shows the distribution for experiment 2. For the second experiment, the experts were able to come to a consensus for 85 student submissions. As can be seen, the distribution is non-uniform and skewed, and this will be taken into account when interpreting the model results.

9.4.2 Classification models

For the classification models, we evaluate the performance of the models by two measures: accuracy (percentage of correct predictions) and AUC (area under the ROC

[3] https://www.cs.waikato.ac.nz/ml/weka/

Table 9.2: *List of features*

Attribute code	Description
GENERIC FEATURES	
ICC	Important Class Count
IATC	Important Attribute Count
IOC	Important Operation Count
IASC	Important Association Count
IIHC	Important Inheritance Count
IAGRC	Important Aggregation Count
ICOC	Important Composition Count
IINC	Important Realisation Involving Inheritance Count
SPECIFIC FEATURES	
TANK_C	Class Tank Present
BULLET_C	Class Bullet Present
SHELL_C	Class Shell Present
WORLD_LEVEL_C	Class World/Level Present
SCORE_C	Class Score Present
PLAYER_USER_C	Class Player Present
TANK_AT	Attribute in Tank Present
BULLET_AT	Attribute in Bullet Present
WORLD_LEVEL_AT	Attribute in World/Level Present
SCORE_AT	Attribute in Score Present
TANK_O	Operation in Tank Present
BULLET_O	Operation in Bullet Present
WORLD_O	Operation in Bullet Present
SCORE_O	Operation in Bullet Present
PLAYER_O	Operation in Bullet Present
TANK_BULLET_AS	Association between Tank and Bullet
TANK_TANK_AS	Self-association of Tank
TANK_PLAYER_AS	Association between Tank and Player
GAME_LEVEL_AS	Association between Game and Level
TANK_SCORE_AS	Association between Tank and Score
TANK_IH	Inherits from Tank
AMMO_IH	Inherits from Ammo
BULLET_IH	Inherits from Bullet
TANK_WORLD	Aggregation - Tank part of World
SCORE_WORLD	Aggregation - Score part of World

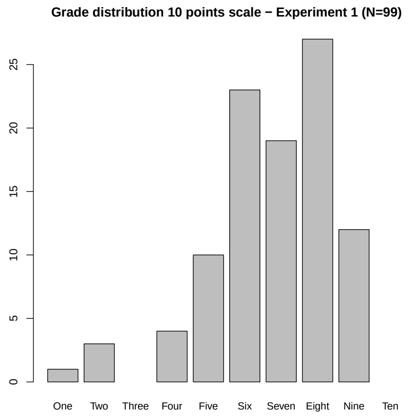


Figure 9.2: *Grade distribution 10 points scale - Experiment 1*

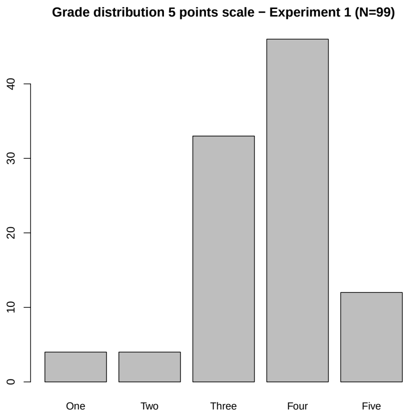


Figure 9.3: *Grade distribution 5 points scale - Experiment 1*

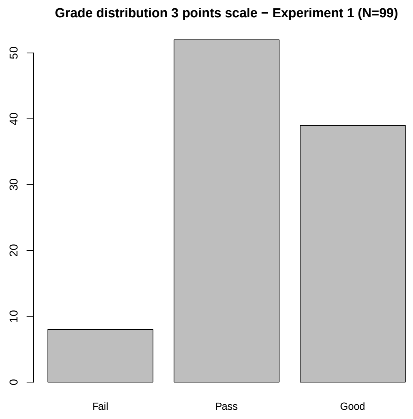


Figure 9.4: *Grade distribution 3-pnts scale - Experiment 1*

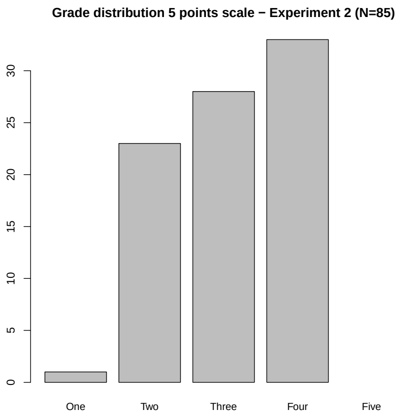


Figure 9.5: *Grade distribution 5-pnts scale - Experiment 2*

curve). To take the skewedness and non uniformity of the outcome (grade distribution) into account, accuracies are compared against majority vote, and it was also one of the reasons to include AUC.

Table 9.3 shows the analysis of the accuracy of the different prediction models. What can be observed is that the coarser the grading scale gets in the first experiment, the better the performance of the model is – which is to be expected. The best accuracy is found in the second experiment. This experiment delivers a significant accuracy of 69.42. Compared to the majority vote baseline (ZeroR) of 38.82 this gives the best result.

In addition to the accuracy, Table 9.4 shows the AUC. We did to double check our findings for the accuracy. The AUC values support our finding that the random forest algorithm has the best performance to build the model.

Table 9.3: *Classification experiment 1 and 2 - accuracy (10 runs, 10 fold). Values in italics are significantly better than ZeroR at the 0.05 level.*

Dataset	ZeroR	logistic	simple-log	1-nn	dec. table	dec. tree	rnd forest	rnd tree	naive bayes
all 10 1st	27.30	30.89	33.57	32.98	*37.31*	29.07	37.44	32.71	34.92
generic 10 pts 1st	27.30	32.2	32.88	*42.30*	28.31	33.32	*42.76*	*39.30*	*39.02*
all 5 1st	46.47	45.78	*59.66*	54	*58.41*	52.73	*58.02*	47.38	40.51
generic 5 pts 1st	46.47	49.61	53.34	51.06	54.64	51.89	53.97	46.62	51.52
all 3 pts 1st	52.58	57.06	62.03	*62.82*	*68.79*	59.76	*66.30*	59.89	*64.46*
generic 3 pts 1st	52.58	*64.14*	62.33	*64.66*	60.46	60.57	*65.24*	*64.30*	*65.20*
all 5 pts 2nd	38.82	*51.49*	*56.12*	*58.88*	*50.92*	*60.33*	*64.00*	*55.19*	*57.89*
generic 5 pts 2nd	38.82	*61.33*	*59.08*	*54.49*	*46.76*	*66.10*	*69.42*	*59.01*	*55.76*

all: specific and generic features

Table 9.4: *Classification experiment 1 and 2 - AUC (10 runs, 10 fold). Values in italics are significantly better than ZeroR at the 0.05 level.*

Dataset	ZeroR	logistic	simple-log	1-nn	dec. table	dec. tree	rnd forest	rnd tree	naive bayes
all 10 1st	0.50	0.62	*0.66*	0.52	*0.72*	0.59	*0.73*	0.56	*0.64*
generic 10 pts 1st	0.50	*0.63*	*0.63*	*0.68*	*0.57*	*0.67*	*0.76*	*0.65*	*0.69*
all 5 1st	0.50	0.61	*0.65*	*0.64*	*0.65*	0.59	*0.74*	*0.60*	0.59
generic 5 pts 1st	0.50	0.61	*0.62*	0.59	*0.62*	0.62	*0.67*	0.57	*0.63*
all 3 pts 1st	0.50	*0.70*	*0.79*	*0.71*	*0.80*	*0.70*	*0.84*	*0.67*	*0.85*
generic 3 pts 1st	0.50	*0.81*	*0.81*	*0.71*	*0.72*	*0.73*	*0.84*	*0.71*	*0.83*
all 5 pts 2nd	0.50	*0.72*	*0.82*	*0.77*	*0.70*	*0.75*	*0.87*	*0.69*	*0.83*
generic 5 pts 2nd	0.50	*0.86*	*0.85*	*0.74*	*0.58*	*0.76*	*0.86*	*0.67*	*0.84*

all: specific and generic features

9.4.3 Regression models

Table 9.5 shows the results for the regression models. We analysed the mean absolute error (MAE). Which means that if we use regression as a prediction model, using the linear or random forest algorithm, the best models will be between 0.47-0.56 point off at best. Although significant, in comparison with just giving everyone the average grade (ZeroR, MAE = 0.71) this is not a large difference.

Table 9.5: *Regression models - MAE (10 runs, 10 fold). Values in italics are significantly better than ZeroR at the 0.05 level.*

Dataset	ZeroR	linear	simple linear	random tree
all 10 1st num	1.34	*1.00*	1.28	1.08
generic 10 pts 1st num	1.34	*0.99*	1.24	1.09
all 5 1st num	0.74	0.62	*0.63*	0.62
generic 5 pts 1st num	0.74	*0.57*	*0.62*	0.62
all 5 pts 2nd num	0.71	*0.55*	*0.56*	0.58
generic 5 pts 2nd num	0.71	*0.48*	*0.56*	*0.47*

all: specific and generic features

9.5 Discussion

In this section we answer our main research question: How can machine learning contribute to the automated grading of software design class diagram tasks?. In order to do this we reflect on the models of Section 9.4. In addition we discuss threats to validity and discuss future work.

9.5.1 Reflection on models

In general, based on the results, we can say that for now it is not reliable enough to apply the model to grade students class diagrams automatically on a 10 point scale. The highest accuracy we have achieved is 42.76%. What we can say is that the rougher the grading scale gets, the higher the accuracy, which is to be expected. In the first experiment this leads to a top accuracy of 59.66% using a 5 point (46.47% ZeroR) and 68.79% (52.58% ZeroR) using a 3 point scale.

But how applicable would such a coarse scale grading then be? It could be used in online training systems where the emphasis lies on getting a rough indication of quality of the solution. Also, often at the universities, practical parts of SE courses only require a Pass-Fail. That said, it would lack feedback about the errors made for the students. In addition, compared to the majority vote we can say that the models do not perform that much better.

In the second experiment we achieved a accuracy of 69% using a 5 point scale and trained by a manual 5 point scale grading. Also, the graders looked at all the models and came to a consensus. In this case not only the accuracy is higher, but also the delta with ZeroR (39%) is larger.

From the results it is not clear that adding specific features is a better approach than just using the generic features. Looking at the random forest column we observe that the generic features perform better in the 10 points scale of experiment 1 and in experiment 2. However, this is not consistent throughout all results. Even if there is a significant difference, it is not a big difference.

9.5.2 Threats to validity

We are aware of the validity threats in this research. We discuss the following:

The experts graded a part of the submitted assignments twice in the second experiment. This could introduce a bias. We expect that the use of the rubric and the consensus of the experts lead to a high accuracy and that it did not negatively influenced the experiment.

The features in this case are specific to the assignment. On one hand one could argue we cannot generalise the approach presented is this chapter. On the other hand the process that delivers the features could be repeated in other cases with other assignments. Future work should confirm the generalisability of the approach.

In general we are aware that a larger number of subjects would yield a more accurate model when using machine learning.

9.5.3 Future work

The goal of this work was not to build the best model possible, it was to explore the feasibility of this use case. If the grading task is very well defined, for instance through a rubric, there would be no need to use a machine learning approach as the rubric could simply be automated. However, it becomes more interesting if the rubric is not well known, elements may be missing, or the weighting of various elements is not well defined.

One angle to future work would be to repeat these experiments with more instances, and different features. Also grading tasks that are less well defined, such as the quality of the layout of the diagram, could lend itself better to a machine learning based approach. What will remain challenging is to develop models that would generalise and validate across different exercises, this could be evaluated with a separate validation exercise.

In this chapter we constrained the evaluation of students' work to grading. Future research should explore the application of machine learning for providing feedback to students during the performance of software design tasks.

9.6 Conclusion

In this chapter we presented an initial approach for using machine learning for building models for the prediction of grades for software design assignments. We collected the data of submitted software design assignments of bachelor student pairs. With this data we performed two experiments in which we built models for prediction with machine learning software.

Based on the results of the first experiment we conclude that using machine learning for a classification that uses the same precision as a 10 point grading scale is not accurate. Classifying in a 3 points (fail, pass, good) or maybe even using a 2 points (fail, pass) comes closer to accuracy, but is still not good enough (an accuracy of 65.24 for 3 points). The models of the second experiment performed better. The best model had an accuracy of 69.42 using a 5 points grading scale. Again, this model is not accurate enough, but heads towards an acceptable value.

Currently we could integrate a prediction model in online training software to give users a rough indication of the quality of their models. Future work should include larger training sets and the exploration of the application within e-learning environments and for other features such as providing feedback.

Chapter 10

An Online Educational Agent: Automated Feedback for Designing UML Class Diagrams

Description of contributions of the authors
The research in this chapter was done in collaboration with Helen Anckar. At the time of writing Helen had conducted her bachelor project at the joint department of Computer Science of the University of Gothenburg and Chalmers University of Technology in Sweden. Her topic was to design and implement a proof of concept of an educational agent that gives students feedback when they are creating a class diagram.

The contributions of Dave Stikkolorum were: introducing the topic, taking part in the discussions about how to apply the concept of pedagogical agents in the field of software design education, supervising the research process, supporting the software design and programming process and writing this chapter.

This chapter describes our exploration of the automatic feedback to students during their process of designing a class diagram. To this end, we extended our online educational design tool WebUML with a feedback on demand feature. This feature

This chapter elaborates and extends on the bachelor thesis of Helen Anckar: Helen Anckar. Providing automated feedback on software design for novice designers. BSc thesis, Goteborg University, 2015

was implemented by means of a pedagogical agent. The feedback agent was evaluated by five students and resulted in future direction for research. We conclude the agent is still immature and further research is needed to be conducted in order to make such a technique useful for educators.

10.1 Introduction

As mentioned in the previous chapters of this dissertation, several studies point out that software design and its modelling activities are challenging topics for novices [80][69][113][141]. Given the fact that software design is a practical skill, we believe educational programs should offer students a 'learning by doing' approach (as discussed in Chapter 3) and facilitate the students with much exercise time to overcome their difficulties. At the same time, this requirement creates a challenge in organising and staffing a course:

- The need for much practical hours demand a large group of lecturers or teaching assistants (TAs) that have to be involved.
- It is a challenge to support the students at the right time. A lecturer or TA can only support a couple of students at one point in time. If the other students are stuck, they have to wait for help.
- Students could benefit from being guided during their design process. This enables the possibility to take smaller steps, instead of only having a evaluation at the completion of their task.

The above mentioned requirements are not easy to fulfil in the typical classroom. Both in classic lectures and online courses, high numbers of students challenge the process of evaluation and grading. In classic lectures, accompanied with practical workshops, 200 students or more are not an exception. Online, this number can even be higher. To overcome scalability problems, an automated approach of more tailored feedback is needed. Moreover, this enables students to practice at their own pace and in their own time.

The research in this study focuses on feedback. We investigate how the integration of a pedagogical agent in our online WebUML tool can support overcoming the aforementioned challenges. We aim to design an agent that is capable of guiding students towards their solution by giving feedback during the performance of their task.

There are several categories of criteria that lecturers use when giving feedback on the design of class diagrams (also available as checklist in Appendix E):

- Syntax – does the diagram follow the formal notation rules of the modelling language? At the time of writing UML 2.4 for example.
- Layout – Is the diagram readable? Does it not mix styles? For example: is an inheritance relationship drawn from top to bottom across the whole diagram?
- Consistency – is the diagram consistent with the other diagrams in the model? For example: does a class name correspond with a lifeline in a sequence diagram?
- Semantics – Does the diagram represent what it should? For example: a '1 .. *' multiplicity is applied. Which means '1 to many'. Was this meant?
- Requirements satisfaction – does the diagram support the (functional) requirements? For example: Does the class diagram have a set of classes that cover the responsibilities of the system that is designed?
- Design principles – are common design principles applied? For example: did the student take coupling and cohesion into account?
- Design decisions – were the right design decisions made and applied? For example: a pattern is applied to support a quality requirement.

Of these criteria, Syntax and Layout and - to some degree - Consistency and Design Principles can be addressed using well-defined formal- rules. The difficulty with evaluating students' designs lies mainly in evaluating the criteria: 'Semantics', 'Making design decisions' and 'Requirements satisfaction'.

Design principles can be considered in isolation and then evaluated as a rule. For example metrics for coupling and cohesion can be computed [25, 46] and compared against some threshold. However, a difficulty that arises is the balancing between the right amount of coupling versus the right amount of cohesion. Making such trade-off's are not easy to capture as a rule. Indeed, such decisions depend on knowledge in addition to the (technical) solution context, such as domain knowledge.

For constructing a diagram's layout certain styles can be followed. A modelling language's syntax is defined by a formal grammar. Modern modelling tools, such as Visual Paradigm [1] or Papyrus [2], are already equipped with systems for checking the syntax of diagrams and the consistency of these diagrams within a model. A large overview of consistency rules for UML is reported by Torre [150]. For layout, most modelling tools also provide functionality to guide the construction of the layout. Also [16] demonstrated good results in automatically evaluating layout of class diagrams using a machine-learning approach.

[1] https://www.visual-paradigm.com

[2] https://www.eclipse.org/papyrus

As mentioned above, the key challenge lies in evaluating the semantics, design decisions and requirements satisfaction. In this chapter we explore the possibilities for addressing these challenges.

As an extension of our WebUML editor, we developed a module that can be seen as a pedagogical agent that is capable of providing feedback to students by evaluating their class diagram. The aim of the agent is to provide students with feedback while they are working on a software design task in order to guide the student to the correct solution. The feedback agent evaluates the correctness of a student's design in order to give guidance for improvement. In our approach, the student solution is compared to a model solution and the feedback the student receives focuses on semantics, requirements satisfaction and the design decisions.

In this chapter we explore the possibilities and limitations of a feedback agent that guides students towards the solution of their software design task. The primary focus is on the role of the agent as a *guide*:

> the agent supports the student with hints about the quality of his design and suggests steps in the right direction. The agent aims to let the student discover the direction until a final situation is achieved.

In our approach the agent needs to *evaluate* some aspects of the design before giving feedback for guidance. However, the agent does not grade the student's task:

> as evaluator, the agent evaluates the quality of the student's software design. Evaluation takes place at the end of the student's task.

By researching this topic we aim to: i) improve online learning approaches, and ii) decrease the workload of lecturers.

The remainder of this chapter is organised as follows: Section 10.2 discusses related work. Section 10.3 discusses the research method of the study. Section 10.4 presents the results. We discuss the findings in Section 10.5 by answering the research questions. We close this chapter by concluding and proposing future work in Section 10.6

10.2 Related Work

This section describes existing research that is related to the evaluation of students' modelling products. We mainly focus on the following categories:

- tooling – the use of modelling tools in an educational context for supporting the development of the students modelling skills

- help seeking – the way in which students seek help, at what typical moments, and what kind of topics they need help for
- providing feedback – providing the right type of feedback (task or self related) and the right amount (prevent cognitive overload) at the right time
- pedagogical agents – a digital, automated solution that recognises students' struggles, provides feedback and guides them towards solutions.

10.2.1 Interactive UML Learning Environments

Baghaei et al. present COLLECT UML, a tool for individual and collaborative learning of UML software design [10]. Their web based tool is capable of giving ondemand feedback by comparing submitted individual solutions with an ideal solution provided by the lecturer. The tool also supports a group phase in which the group solution exposes the differences between the individual solutions in order to enable group discussions for improvement of the group model. Their constrained-based tutor shows supporting messages on one side of the screen in order to guide them towards a correct solution. The feedback message are of different nature: Simple Feedback (the solution is right/wrong), Error flags (indicators for missing elements, such as classes, attributes/operations or types), and Hints (guiding from violated rules). The more help students need, the more detailed the feedback is (Hints).

Chen et al. designed CoLeMo, an online learning environment for collaborative UML modelling of geographically distributed students [24]. The PC tool uses pedagogical agents (see 10.2.4) for providing domain related advice and collaboration related advice during the activity of the design task students perform. By monitoring students in a chat area, responses to questions (agreement buttons) and activities in a shared workspace CoLeMo provides task related feedback and feedback on collaboration.

The modelling task related feedback is limited to the UML domain level, such as explaining a violation of a UML diagram construction rule. For example wrong use of the abstract class concept. The feedback on the collaboration is based on a weighted activity log. For example: additions of classes are assigned with a higher weight than renaming. Based on the logs the agent provides feedback about participation and warns when elements that were created by others are about to be deleted.

Tourtoglou et al. [151] designed AUTO-COLLEAGUE, a collaborative UML environment specifically designed for the formation of groups. They use models of prior performance of the users for creating an advice for the formation of groups. With the help of defined stereotypes for the level of expertise, performance types and personality the tool generates the advice. The UML knowledge is observed by a human trainer or recorded by a wizard drive test module. It seems that the tool does not provide feedback on the UML designs.

The above mentioned works show that with interactive learning environments additional possibilities for providing feedback to students are created. At the moment of conducting our research we noticed that there was more work that relates to the feedback on the design process than actual software design and software design decisions. This motivated to conduct more research on feedback on actual software design and, in the future, include design principles in the feedback mechanism.

10.2.2 Help Seeking in Interactive Learning Environments

Aleven et al. propose to use Nelson-Le Gall's help seeking model [96] for children's problem solving tasks in an interactive learning environment [4]. The help seeking model consists of 5 stages:

1. Become aware of need for help.
2. Decide to seek help.
3. Identify potential helper(s).
4. Use strategies to elicit help.
5. Evaluate help-seeking episode.

Although this model originally was designed for classroom contexts it should also be applicable for interactive learning environments [4]. The difference is the 'helper' role. In a classroom context this role is fulfilled by the lecturer, in an interactive learning environment, next to asking help from the lecturer, this could be a computer program or glossary.

Aleven et al. conducted a study [3] in which 15-year old students used a glossary and an intelligent on-demand hint system for help seeking. They concluded that the students rather use the hints than the glossary. For the hints, they tend to search for the more specific hints (close to the solution) than the high level (guiding) hints. Learning from the right answer, just as learning from examples, requires that the student has an understanding of why the answer is right [7]. Also, the students seem to wait long before using the help facilities. Even when making a high number of errors per step.

In our research, we want to take the stages of Nelson-Le Gall into account when designing the pedagogical agent. We want to design a help system that is guiding and will not use a glossary kind of approach. Eventually we want to be able to recognise common students' design mistakes in order to teach them by reflecting on these mistakes. Our future modelling tools should be equipped with smart feedback systems that guide beyond reflection on syntax errors. Rather they reflect on students' decision behaviour and trigger and improve self-evaluation.

10.2.3 Providing Feedback

Hattie et al. propose a model for applying the right type of feedback that has the greatest impact in a certain situation [49]. It is based on three questions: *Where am I going? How am I going? and Where to next?* The levels of feedback that they identify are:

- feedback about the task – performance on a task,
- feedback about the processing of the task – understanding how to do the task,
- feedback about self-regulation – self monitoring and regulation, and
- feedback about the self level (personal level) – evaluation and affect on personal level unrelated to the task.

Interesting to mention is that, in the analysis of the studies they report, the largest effect of the feedback was via feedback on the task. This effect was larger than that of praise, reward and punishment. For applying feedback Hattie et al. emphasise that timing should be considered. Also, they explain there is a choice between negative and positive feedback. Which can both have a positive influence on learning. For example, the choice depends on the level of feedback and the attitude a student has to the task (does the student *want* to do a (learning) task or does he/she *have* to do it?). Moreover, in classroom situations a safe environment should be created to make the learning caused by the feedback effective, e.g. students should feel safe to make errors.

Kalyuga explains that the cognitive architecture of our brain should be taken into account when designing interactive learning environments [63]. Our cognitive system can be easily overloaded when offering too much information to process. Kalyuga warns that poorly designed feedback can lead to split-attention. The students then are driven away from the original task. Providing feedback or guiding on demand is one of the examples that could help to reduce the amount of unnecessary cognitive load. Another suggestion is to provide feedback sequenced with an increase of provided details (amount of information) in the feedback message.

10.2.4 Pedagogical Agents

In general a pedagogical agent is an autonomous piece of software that supports learners by interacting with them. Pedagogical agents consist of a software agent (typically using some sort of AI) and a virtual character. The characters impersonate a person that has a specific role to the students, e.g. a tutor, teacher or learning companion. The character can be static (e.g. a picture) or dynamic in the form of an animation to benefit the interaction between the agent and the student and the motivation and engagement of the student.

Gulz and Haake argue that the visual and aesthetic aspects of the animated agents should be taken seriously as they have an important impact on the user *Look is too important to be left or just happen* [42]. In another study they provide an overview of the different roles an agent can have [43]. The distinguish between:

- More authoritative roles – tutor, coach, instructor etc.
- Less authoritative roles – competing co-learner, collaborating learning companion, etc.

In their studies Gulz and Haake emphasise that *pedagogical agents are not a digital monsters threatening to replace teachers – they are only a tool with the potential to help teachers in daily classroom activities and hopefully help them to personalise the learning during the everyday activities*.

Mørch et al. see an important role for pedagogical agents as conceptual awareness mechanisms in their Computer-Supported Collaborative Learning environments. [91] The propose a design space that supports the adoption of pedagogical agents in interactive learning environments. The mention four dimensions that support the design of a pedagogical agent:

- presentation – how an agent should present itself to a user
- intervention – when the agent should present itself to a user
- task – the student's task (work / assignment)
- pedagogy – the kind of information that is presented.

In a meta-analysis of 43 studies Schroeder et al. [120] showed that pedagogical agents have a small, yet positive significant effect on learning. The effect was higher for K12 students than post-secondary students. Dincer et al. [34] conducted a study on multiple-pedagogical agents: the use of multiple types of agents. They report that pedagogical agent use has a very important effect, especially on academic success. Based on their findings they suggest the use of pedagogical agents in their computer assisted software.

The effect of the pedagogical agent on cognitive load is still being discussed. Schroeder did not find a significant difference between groups (with or without the use of an agent) related to their training efficiency or instructional efficiency in his experiment [120]. Müller-Wuttke et al. see the pedagogical agent as an instrument to reduce cognitive load [93]

In our design we combine the findings of the works mentioned above. We aim at an agent that:

- acts as a guiding tutor,
- evaluates the solution of the student based on a lecturer's solution,
- gives feedback on the different task related and self regulation levels, and
- takes a student's cognitive load into account.

10.3 Research Approach

In this section we explain our research approach. We explain our choice for design research. We explain how we iterated over different (design) research steps and how we analysed the data.

10.3.1 Research Method

For developing the feedback agent we applied a design research approach [54]. Because of the involvement of designing, implementing and evaluating a software artefact as part of the study, this approach was found most useful. Figure 10.1 shows the overall process of the study. The different phases are described in the following subsections. The design evolved by iterating over the requirements, design, implementation and evaluation phases. The improvements are discussed in Subsection 10.4.1

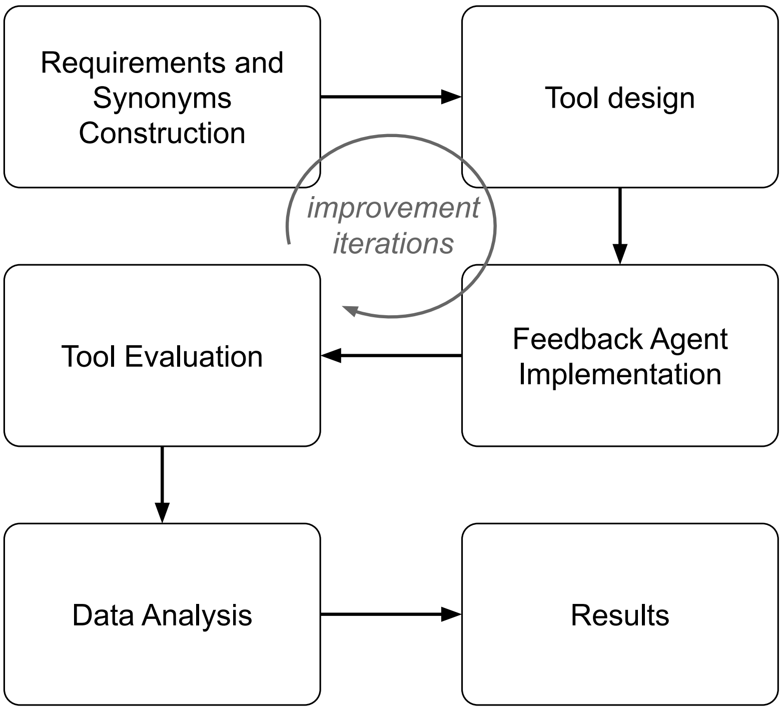


Figure 10.1: *The overall research approach for the feedback agent study*

10.3.2 Requirements Collection

First we collected the requirements for the construction of the feedback agent. To this end, we analysed the experiences of students designing UML diagrams from the study that was discussed in Chapter 6 [137]. In this study we asked students to record the questions that arose when facing a difficulty while performing a class design task in order to uncover common difficulties. The students created a class design for a game (see Appendix C for the game description). We used the game designs for further analysis of the research that is described in this chapter. The aim of the analysis was to investigate what particular software design mistakes students make in an assignment. These mistakes point to the topics that students could use feedback on during their tasks. The following requirements that the feedback agent should fulfil were collected based on the analysis of the designs of Chapter 6:

Naming and Principles

– Offer domain-level feedback on UML naming conventions.

Satisfying requirements

- Inform designers of missing interface classes.
- Provide advice on missing classes.
- Provide advice on missing attributes.
- Provide advice on missing operations.

Paradigm and generic design concepts

- Provide advice regarding inheritance.
- Feedback on general design principles.

10.3.3 Design of the Feedback Agent

The feedback agent was build as an extension for WebUML, our self made online UML editor (discussed in Chapter 4). For extending and future possibilities, such as connections with other sub modules or the integration in other systems, the agent is designed as a separate module. Figure 10.2 shows a high level overview of WebUML, including the feedback agent.

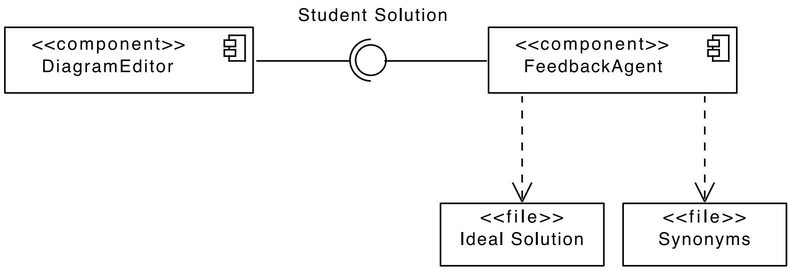


Figure 10.2: *The high level overview of WebUML including the feedback agent*

We choose to check a student's diagram against an ideal solution. Although we will continue exploring other approaches such as using machine learning, we choose to focus on a proven (less flexible) approach that delivered us a prototype for further investigation.

The feedback of the agent is shown to the user in a text balloon of an avatar. Using such a simulated human-like interface is common when using pedagogical agents.

In addition to the feedback agent a progress bar is shown at the bottom of the WebUML editor. The progress bar supports students' need for confirmation that was also found in Chapters 3, 7 and 8.

The general work flow for receiving feedback with the feedback agent is as follows:

1. The user presses a button for reviewing his/her design.
2. The feedback agent receives the data of the model that is kept by the main program.
3. The feedback agent evaluates the data and sends the evaluation back to the main program in the form of a feedback message. In addition, a progress value for the progress bar is also sent.
4. The main program sends the feedback message to the text balloon of the avatar and updates the progress bar.
5. The user reads the feedback message and possibly improves the design.

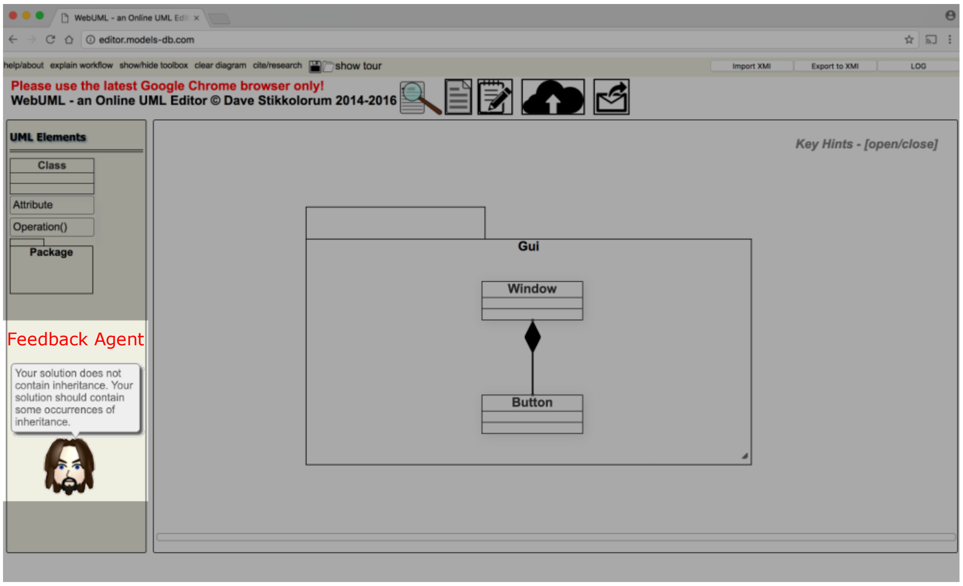
Figure 10.3 shows a screenshot of WebUML with the integrated feedback agent enabled.

The feedback is triggered 'on-demand'. This is done because of Nelson-Le Gall's second stage: decide to seek help. We wanted to examine the decision of asking for help rather than the reaction on a offer for help. In addition, letting the user decide is less intrusive than forcing help during modelling. When a student is in the need for help he/she has to press the 'help-button'. After pressing for help. the feedback agent is activated. The agent compares the student's solution with the 'ideal' solution that lectures prepare and store in a database file. This solution file also contains a set of synonyms. The synonyms make the comparison more flexible. It supports a reasonable flexible interpretation of the assignment text and textual representation of the student solution.

We investigated how to equip the feedback agent with the appropriate feedback messages. As feedback is a powerful tool, it can also have negative effect when not properly applied [49]. The feedback agent was designed to give feedback in sequential stages, not providing too much detail at once, in order to prevent extraneous load.

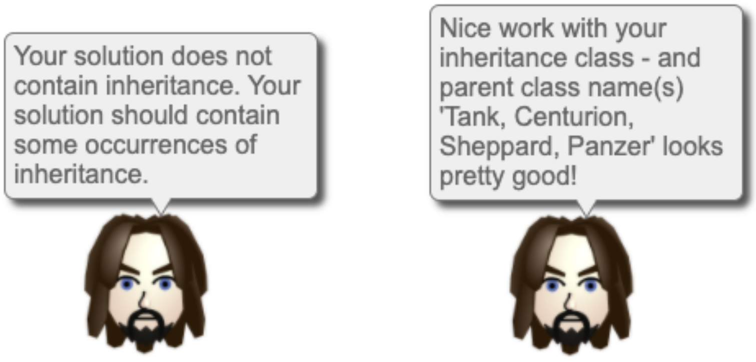
As mentioned in Section 10.2 Hattie et al. explain four levels of feedback [49]: Task Level, Process Level, Self-regulation Level and Self Level.

The feedback agent implements three types of feedback and for these we focus on the levels that are the closest to the actual design task, rather than on the general self-level. Figure 10.4 shows two moments of the avatar's feedback during a student's assignment process. Table 10.1 shows examples of feedback messages that are currently implemented in the feedback agent.


WebUML - an Online UML Editor
editor.models-db.com
help/about explain workflow show/hide toolbox clear diagram cite/research show tour
Import XMI
Export to XMI
LOG
Please use the latest Google Chrome browser only!
WebUML - an Online UML Editor © Dave Stikkolorum 2014-2016
UML Elements
Class
Attribute
Operation()
Package
Feedback Agent
Key Hints - [open/close]
Gui
Window
Button

Your solution does not contain inheritance. Your solution should contain some occurrences of inheritance.

Figure 10.3: *WebUML including the feedback (highlighted) agent*



Your solution does not contain inheritance. Your solution should contain some occurrences of inheritance.

Nice work with your inheritance class - and parent class name(s) 'Tank, Centurion, Sheppard, Panzer' looks pretty good!

Figure 10.4: *left: feedback on missing elements, right: feedback on good progress*

Table 10.1: *Examples of feedback messages used by the feedback agent*

Feedback Message	Feedback Level
Your classes Tank, Centurion look good but you are still missing some important classes.	Task
Read the task carefully and see if you can find the remaining classes by identifying nouns, preferably complete singular nouns.	Processing
Class name(s) Tank looks good.	Regulatory

10.3.4 Implementation and evaluation

In this section we describe the evaluation steps we performed. We evaluated the tool by asking students to perform a class diagram task. We collected observations on their behaviour while designing and performed post-task interviews (Table 10.2). Based on the observations and interviews, we adjusted and extended the feedback agent. In total 1 first year and 4 third years students participated in the evaluation of our agent. The students were studying at a bachelors IT program. They had a basic understanding of UML and software design.

Table 10.2: *Interview questions during research design feedback agent*

What kind of feedback do you think would have been great to see now?
What do you like about the implemented features?
What do you like about the tool as a whole?
What do you think about the progress bar?
How do you think the tool could be improved?
Which part of the feedback helped you solve the problem?

10.3.5 Data analysis and Results

We collected data as part of the design research process. This data was obtained by observing and interviewing the users who trialled the system. This qualitative data was collected during subsequent iterations of the design process and was transcribed and coded.

10.4 Results

In this section we present our findings on the design of the feedback agent.

10.4.1 Design Improvements per Iteration

We started with an initial design and iteratively improved the feedback agent as discussed in Section 10.3. By observing the students during their task, analysing their solutions and listening to their feedback we improved the following:

- Iteration 1 – In the initial design the feedback was mainly focused on task level related feedback. This type of feedback seemed to induce students to use a trial-and-error strategy. Some students solutions lacked important classes, thus we extended the feedback agent with feedback about missing elements, for example missing classes, attributes or operations.
- Iteration 2 – We added feedback about deleting or editing of elements. Students are informed that classes, attributes or operations should be omitted because they should not be present in the design. Elements can be irrelevant to the assignment case and therefore should be omitted. Another example could be that elements should not appear as class in the design, but as attribute. Students can receive feedback about renaming elements when for example a mandatory parent class has wrongly named child classes.
- Iteration 3 – The last addition for the feedback agent was the suggestion to look for hints in the assignment text.

10.4.2 Students' Interviews

The coding and labelling process of the answers to the students' interview questions led to 3 dominant themes: i) the pedagogical agent, ii) the usability of the tool and iii) the user interface.

- Pedagogical Agent – Two main matters were reflected in the interview answers on the theme of *Pedagogical Agent*:
  1. Agent's Guiding Role is Appreciated – All participants appreciated the confirmation of classes, attributes or operations that we constructed in the

right way. Also, they valued the possibility of the feedback agent to advise them about the next steps in a task.

"The feedback is good, I like that it shows correct classes. The part where it says, classes Tank, Centurion and the others look good. I know that these classes are correct. Also, the part that says find classes by identifying nouns in the text."

2. Need for More Help – 3 participants explicitly mention the need for more hints during the task. They suggest to give more information about elements such as missing and irrelevant classes. It could be due to the increasing feedback approach (Section 10.3) that students have this need. Also, when their design skills are low they ask for more information:

   "Someone like me who's not good at UML, I need more hints"

- Tool Usability – The students find the tool easy to use. This was mostly confirmed by observation during the tasks, but was also explicitly mentioned by a student. Another student appreciated the ability to move back and forth between the assignment text and the editor. During the experimental design of the research in Chapter 6 we decided about having the possibility to integrate the reading of the assignment text in the WebUML environment in order to let the student stay focused on the task.

  "It's quite easy to use"
  "I like the ability to move back and forth from the assignment text to editing mode"

  2 students mentioned the lack of the possibility of changing the size of a class, for example, when having long attribute names. 2 students mentioned that they want to be able to select more than one class for moving around the whole diagram in stead of moving one class at a time.

- User Interface – 3 students did not notice the progress bar. It should be improved or reconsidered. It could be due to a bad choice of colour that students mistake the progress bar for a scroll bar or just don't observe it at all.

  "The progress bar is very vague, colour-wise. I thought it was a scroll bar. Maybe add percent or something?"

  1 student seems to appreciate the progress bar and seems to use it in the way we design it:

  "It's helpful you know, if I were to do this without the progress bar, I would still have something missing e.g. attributes, or methods."

- Observations – Although the students recognised the benefits of using a feedback agent, in some cases they ignored the suggestions of the feedback agent and left the errors unchanged.

  Interesting to observe is that the students that had poor UML skills were more willing to change their design according to the suggestions of the feedback agent than the students that had good UML skills. The students with prior UML design experience maintained their design although the feedback agent did not identify it as a good solution. They were confident about their solution to be also good.

10.5 Discussion - Strengths, Possibilities, Limitations and Future Directions

In this section we discuss the study of the feedback agent by discussing strengths, possibilities, limitations and future directions.

The participating students react very positive to the feedback agent. The most important strength of the tool is that the students indicate that they are helped by the agent to move forward in their task by receiving the help hints. In this sense the agent fulfils the role of a guide. Students mention that they appreciate the possibility to get confirmation about their progress from the feedback agent. This makes them more confident about the direction that the are heading. This is in line with our findings in Chapters 3, 7 and 8 in which students mention this and teaching assistants observe the same about confirmation of the progress of the task.

We observed that students that have poor UML knowledge and skills are more willing to use the feedback agent than more experienced students. In general it is logical that students don't need the agent when they are confident about their skills. To make the agent attractive for these students we could expand the feedback with messages that supports more in depth information that challenges students to achieve higher design quality. However, the challenge to trigger them asking for help in the first place will remain.

The progress bar was perceived as a helpful addition but its design needs to be modified. At the moment it can be mistaken with for example a scroll bar.

A limitation is the fact that some students ignore the feedback of the agent. In our case, the students that neglected the hints of the feedback agent were more experienced than the ones that seemed to follow the suggestion more closely. Also letting students do the task under observation could have influenced their behaviour. Maybe they did not want to look 'dumb' in front of the observer by following the suggestions. Also, because the detection of alternative solutions is a challenge, we have to accept the ignoring of the hints by the students.

In addition, in the future the introduction of disconfirmation hints could be explored. Hatie et al. [49] point out there is evidence that this can be more potent on the self level.

At the same time they show that only the self level is not effective. A combination of the self level and the task level is preferable. At the task level disconfirmation should be accompanied by corrective feedback to be powerful. Combined this could lead to feedback messages such as "you are not on the right track, you are missing important classes. Did you find all class candidates in the assignment text or did you maybe select them to be an attribute?".

We did not take into account irrelevant elements, such as an irrelevant class that is present in the student's class design. The missing of hints about irrelevant classes could have encouraged the trial-and-error behaviour of the participants. Hinting about irrelevant elements could have motivated them to analyse the text again or try to look at the problem from another angle instead of trying to 'find' the classes by asking the feedback agent for confirmation. The introduction of irrelevancy could stay a limitation in general.

It is known that design problems can have multiple solutions and thus could have classes that are being recognised as irrelevant by a feedback agent that makes this distinction. Having a large pool of synonyms related to the words in the assignment could be a good start. Maybe natural language processing should be involved. Further investigation is needed for this direction.

The feedback agent compares a student solution with an ideal solution, therefore it cannot handle multiple solutions in a broad sense. Although the introduction of synonyms helps to allow flexibility in the naming, this still leaves challenges when variations of design structures are made by students. Future research could focus on the introduction of design principles in the agents' diagnosis as measure for the quality of the design. The judgement of the design would then be quality driven in stead of a 'perfect' fit driven approach.

10.6 Conclusion

In this chapter we described our approach in designing a pedagogical agent for giving feedback to students while learning to make UML class designs. We explored the possibilities for addressing the challenges of guiding students in using the right semantics, satisfying the requirements of assignments and help them with design decisions.

We presented our feedback agent, a software module that extends WebUML. We used a design research approach to iteratively develop a pedagogical agent that uses 3 task related feedback levels. We aimed to overcome the above mentioned challenge by using the method of letting the feedback agent compare the student's work with an ideal solution that is prepared by a lecturer.

Early findings from observing and interviewing a small group of students that tested the prototype of the feedback agent gives us preliminary insights into the benefits and limitations of this approach. The students appreciate the guiding role of the feedback agent: the way it confirms steps that were done in the right way and suggestions for next steps. Students mention they would like to see more help during their task. We observed that some students ignore the hints and leave errors unchanged.

Based on the comments of the students, it seems the agent can be well used in the role of a guide that supports students towards the fulfilment of their task. More experienced students sometimes did not follow the advice offered by the agent even though this could improve their diagram. Future research should investigate how this type of students can be motivated to use the feedback agent.

Although still immature, we see a future for the use of the feedback agent in our future improvements of WebUML and/or other interactive learning environments.

Part V

Student Engagement

Chapter 11

A Workshop for Agile Modelling

Students have various difficulties with software modelling, the software development process and with positioning modelling as a means to support their software development. The Agile methodology Scrum has gained popularity in industry and also amongst students. Unfortunately agile projects often lack adequate documentation. Modelling and the agile process could complement each other. The combination of modelling and agile development is not often used in education. Based on our positive experience with the interactive LEGO4SCRUM workshop we use in our programs, we propose an approach based on this workshop that integrates UML modelling into the Scrum process. The workshop lets students experience a whole development cycle from a modelling perspective. Besides this new approach we also categorised comments students wrote down based on their discussions with their peers. We evaluated the workshop with a questionnaire. The students react positive on the approach and indicate they have gained new insights. This chapter explains the workshop set-up, presents its evaluation and discusses the results.

This chapter is based on the following publication: Dave R. Stikkolorum and Michel R.V. Chaudron. A workshop for integrating UML modelling and agile development in the classroom. In *Proceedings of the Computer Science Education Research Conference 2016*, pages 4–11. ACM, 2016

11.1 Introduction

As educators we often see students have various difficulties with software modelling and organising a software development process. Besides learning a syntax of a language such as UML, students have to deal with translating problems into models (analysis) and suggest a solution (design). The abstraction skills that have to be trained for this kind of task often are isolated from another student challenge: the software development process cycle.

Although methodologies such as RUP (Rational Unified Process) and MDD (Model Driven Development) are intertwined with modelling activities, students have difficulties in giving the models a role in the process and treat them isolated, as necessary evil, not as useful tool that supports their development process. Mussbacher et al. [94] show in their work that modelling is not that widely spread (yet) as we (educators and scientists) had hoped for. In the past decade agile methodologies, in particular Scrum, have won popularity in industry and in the classroom. Scrum is designed to do 'just enough' and not to overdo on documentation and does not emphasise modelling. Students seem to appreciate Scrum but, as also found by Stettina et al. [129] in an industry survey, the documentation they produce is inadequate and often lacks models of the design.

Only providing students a methodology that they like cannot do 'the trick'. Students still have to cope with the difficulties of the analysis and design skills and they need to be trained. Educators have to better understand what kind of difficulties students have and how they can guide them to improve their modelling skills.

In our own software engineering programs we use the UML standard for modelling domain and solution with a focus on class diagrams and sequence diagrams. Although the syntax is relatively easy to learn, we notice students seem to have difficulties in placing the use of these diagrams in the process of software development: they do not see the relationship between the diagrams and cannot see the supporting role it has during development.

Consider the following example: in an initial design some related classes play a role in a certain scenario. A very useful technique is to use sequence diagrams to update the design through simulation of certain scenarios of use cases. Our experience is that students often only see the separate steps and do not update their models.

Another problem we noticed students struggle with is how to move from analysis to design. They hesitate what should be the first step.

In industry most software development is done in project settings and students have to learn how to efficiently and effectively work together. After having experience using RUP in our programs we now use Scrum as a software development methodology. We believe the best way for learning Scrum is to let students apply it in a project course. Because we want to avoid to let them just 'use Scrum', we introduce them to the subject of Scrum in a series of lectures and then let them apply this in a project. Often this did not turn out to work that great. We wanted the students to have more experience in Scrum before they start, therefore we adopted a well known workshop that introduces Scrum in a few interactive hours to the students: 'LEGO4SCRUM' [76](mentioned in Section 11.2). The advantage in using the workshop is that students see a whole development cycle in a short time frame. We believe, because of the short time frame, they better relate the development steps.

In previous work [137, 136] we tried to find what typical difficulties students have during a software design task in order to collect information and what kind of feedback we could provide students to improve their modelling skills. We found out that students have difficulties in summarising and explaining their problems clearly.

Based on our positive experience with the LEGO4SCRUM workshop, we propose in this chapter an interactive 2 lecture-hours workshop. In the workshop students are activated to use various types of modelling and have to come, with the help of the discussion in their group, to a detailed design for sprint 1 of the Scrum process (explained in Section 11.3).

In this chapter we answer the following research questions:

- RQ1: Can we integrate (UML) modelling and agile development into an educational workshop?
  and the following sub-questions of RQ1:
  - RQ1.1: Do students learn about the role the (UML) diagrams have in a development process?
  - RQ1.2: Does the workshop help students to improve their modelling skills?
- RQ2: Which insights for modelling education do we gain from the students' group discussions during the workshop?

With this research we aim to contribute: i) a workshop approach for combining modelling and agile development in the classroom and ii) collecting insights into the problems students face during software modelling activities in order to develop future educational programs.

When we talk about modelling in this chapter we mean expressing the problem domain or designing the software solution with the help of UML diagrams.

This chapter is organised as follows: in Section 11.2 we discuss related work. In Section 11.3 we explore our approach and show the set-up of the workshop in more detail. We show the results in Section 11.4 and discuss them in Section 11.5. Section 11.6 discusses threats to validity. The last Section (11.7) is used for conclusions and future work.

11.2 Related Work

To put our work into context, we discuss the related work in this section. First we address work that discusses the problem of understanding the different modelling steps and models. Secondly we address research that combines (elements of) agile with model based methodologies. Lastly, we show interactive approaches of software education workshops and games.

Kaindl [62] discusses the difficulties in transitioning from analysis to design by arguing that the analysis model represents other things than the design model. He explains that analysis objects reflect the problem domain and therefore cannot be the same thing as a design object that reflects a part of the solution. Shin explains that this transition difficulty is caused by semantic distance [122]. He argues that design objects are closer to code than analysis objects are to the problem domain. This creates a semantic distance between the analysis and design.

Since more than a decade researchers suggest to combine modelling and agile from different point of views. Scott Ambler discusses the use of models in an agile way [5] in order to avoid ‘over documenting’ and proposes to do ‘good enough’. Rumpe [116] suggests to use an executable version of UML and compares the use of that with the agile focus on creating rapidly executable models. He applies agile modelling with UML in his textbook [117].

Zhang et al. [162] describe a combination of MDD and agile. They use pair modelling, continuous modelling and UML code generation.

Schroeder et al. [119] use Scrum and a network based card game case for their educational activities. They encourage students to use UML to support their steps and documentation(not per se model based). They mention LEGO4SCRUM [76] as a good team building exercise.

Krivitsky [71] describes the LEGO4SCRUM workshop, a workshop in which students experience a Scrum development cycle including multiple sprints within a couple of hours. The workshop is divided into parts that represent typical Scrum phases. A timer constrains the students’ actitivities. Paasivaara et al. present a simulation game for teaching Scrum to students with LEGO blocks [98]. The game was first developed as a training tool for a security company that adopted agile development in their processes.

Other game based approaches are described by Fernandes et al. They use a card game [36] for university level students to learn Scrum. Wangenheim et al. show a paper and pencil game to teach their students the application of Scrum [158].

Our workshop combines the idea of using models within an agile process with the time-driven structure of the LEGO4SCRUM workshop. We cover a Scrum development cycle and let students experience to move from analysis to design. We do not use the LEGO blocks.

To our knowledge, in practice agile and modelling is not combined often in education.

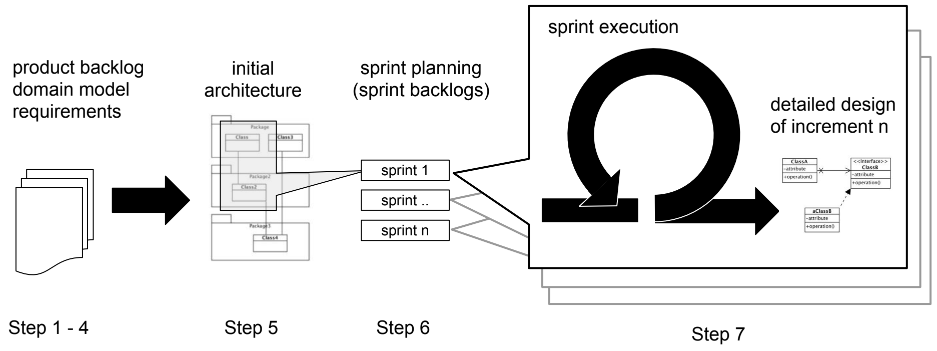


Figure 11.1: *workshop setup*

11.3 Approach

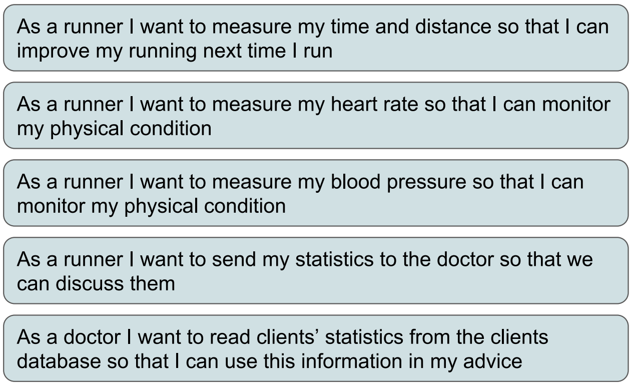
In this section we describe the approach of the workshop. We describe the participants, the case that was used during the workshop, show the tools that were used and explore the workshop structure. We conclude with the questionnaire that was conducted at the end of the workshop.

11.3.1 Participants

The participants of the workshop were first year Bsc students that followed the second semester of a software engineering program. In advance the students followed a Java programming course and followed a project course in which they used Scrum. The students had a basic understanding of class diagrams and sequence diagrams from the

UML. The workshop took place during a regular lecture of the 'analysis and design' course. Of the 88 registered students 50-60 participated in the lecture. In parallel students worked on a project. We used the project groups in the workshop. Students of 13 groups participated. The groups in the workshop consisted of 3-5 students.

11.3.2 Case



As a runner I want to measure my time and distance so that I can improve my running next time I run

As a runner I want to measure my heart rate so that I can monitor my physical condition

As a runner I want to measure my blood pressure so that I can monitor my physical condition

As a runner I want to send my statistics to the doctor so that we can discuss them

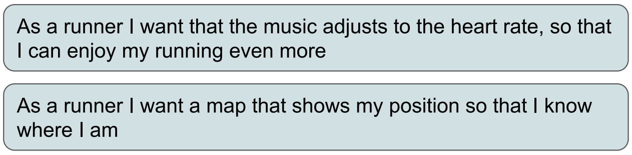
As a doctor I want to read clients' statistics from the clients database so that I can use this information in my advice

Figure 11.2: *Initial User Stories*

The case that was provided was about a 'running app' for a mobile phone that is capable of handling different sensors, such as a watch (measures heart rate) and a blood pressure belt.

11.3.3 Materials and Tools

The students needed a pen and paper to write down proposals they had to hand-in during the workshop. As a modelling tool most of the students used Visual Paradigm [157]. Pen and paper was also allowed. The lecturer needed a beamer and laptop to present slides and to present a timer that was visible during the workshop.



As a runner I want that the music adjusts to the heart rate, so that I can enjoy my running even more

As a runner I want a map that shows my position so that I know where I am

Figure 11.3: *Adding User Stories*

11.3.4 Workshop Procedure

The main idea of the workshop, like the LEGO4SCRUM version, is to simulate a whole Scrum development life-cycle with the emphasis on modelling. The pushing factor is time. The time constraints per phase force the students to discuss, make decisions and model. The product a sprint delivers in our workshop is not a LEGO building or for example (compiled) code. The students produce partial designs that represent an increment of the software solution. The diagrams students use from the UML are use case diagrams, sequence diagrams and class diagrams. The workshop procedure described below follows a typical Scrum work flow. We integrated the modelling activities with the 'standard' Scrum phases. We see this as an extension, because Scrum does not prescribe modelling activities. We believe the iterative character of Scrum, especially having multiple sprints, enables developers to elaborate on their (initial) design.

Figure 11.1 shows the workshop process. The workshop procedure was as follows:

1. **Initial set-up**: The workshop starts by presenting a first domain model that was based on initial user stories (see Figure 11.2). Also the user stories are shown on the screen.

2. **Adding new user stories** (5 minutes): The first activity is to come up with suitable use stories to add. The students have to think as the client and deliver two properly described user stories in time. After the deadline the lecturer discusses the proposed user stories and explains why some of them are not properly defined. Two additional user stories are chosen. This is done by acclamation (for the extra user stories see Figure 11.3).

3. **Update the domain model** (5 + 5 minutes): the students have to update the domain model based on the new user stories. The domain models are peer reviewed. When the deadline has passed the teams have to send two students to criticize the models of two other teams. They grade the models on a 1-5 Likert scale and hand-in the grades.

4. **Adding new requirements** (5 minutes): the next step is to add two new requirements. Five requirements are already shown on the screen (see Figure 11.4 for the initial requirements). The students may use internet and other resources to come up with suitable requirements for the 'running app' case. They hand-in their proposals on paper. Two additional requirements are chosen. This is done by acclamation again.

Requirements:

- The measurement is done by bluetooth devices
- The application should run on Mac OSX+, Windows 8+ and Ubuntu 14+
- There should be a graphical interface and sound
- The distance is displayed in km or miles
  - there is one digit used after the '.' e.g. 10.3 km

Figure 11.4: *Initial requirements*

Extra Requirements:

- The application should adjust the type of music to the pace of the runner.
- The application should use the Google Maps API.

Figure 11.5: *Extra requirements*

5. **Propose the initial architecture** (10 minutes): to be able to plan your sprints most Scrum implementations mention an initial architecture or initial high-level design. This design shows developers an overview of the work that has to be done and therefore supports planning. In our case the students are given a 4 layered architecture framework (UI layer, Application Layer, Domain Layer, Library Layer). The domain model together with requirements that were introduced in the previous steps push the design of the students in a certain direction. The students have to use a UML class diagram to model the architecture. The initial designs are peer reviewed again by different students from other groups.

6. **Sprint planning** (5 minutes): the students choose what particular part of their initial design can fulfil one or more user stories and defines that as a sprint. Sprints take 15 minutes. Students have to choose the part from which they estimate it is feasible to design (refine) it in 15 minutes.

7. **Sprint 1 .. n** (15 minutes per sprint): An agile project consists of multiple sprints that deliver increments of the product. In the workshop the increment of the product is represented as a detailed design part. The iterative character of Scrum, having sprints, is the typical link with what we expect from using modelling languages, elaborating and refining the (design) models. In the sprint the students will use two different diagram techniques from the UML: class diagrams and sequence diagrams. The students will update (refine) the design based on the sequence they explore with the sequence diagram, or test their updated classes and relations by simulating possible scenarios that link to user stories. Again the results are being reviewed by peer reviewers.

8. **Workshop feedback** (5 minutes, not shown in Figure 11.1): the lecturer mentions what stood out while he/she was observing and ends the workshop.

11.3.5 Questionnaire

After the workshop the students were asked to fill out an online questionnaire about their experience with the workshop approach. The questionnaire consisted of 8 questions that were answered on a likert scale (-2,1,0,1,2) and 4 free text questions. The questions targeted the students' experience with this different way of learning and how well they think they have learned from the workshop.

11.4 Results

In this section we discuss the questionnaire that was used to collect the data for evaluating the workshop. First we show the results of the 8 Likert scale questions. Subsequently we show the results of the 4 free text questions. 30 students filled in the questionnaire.

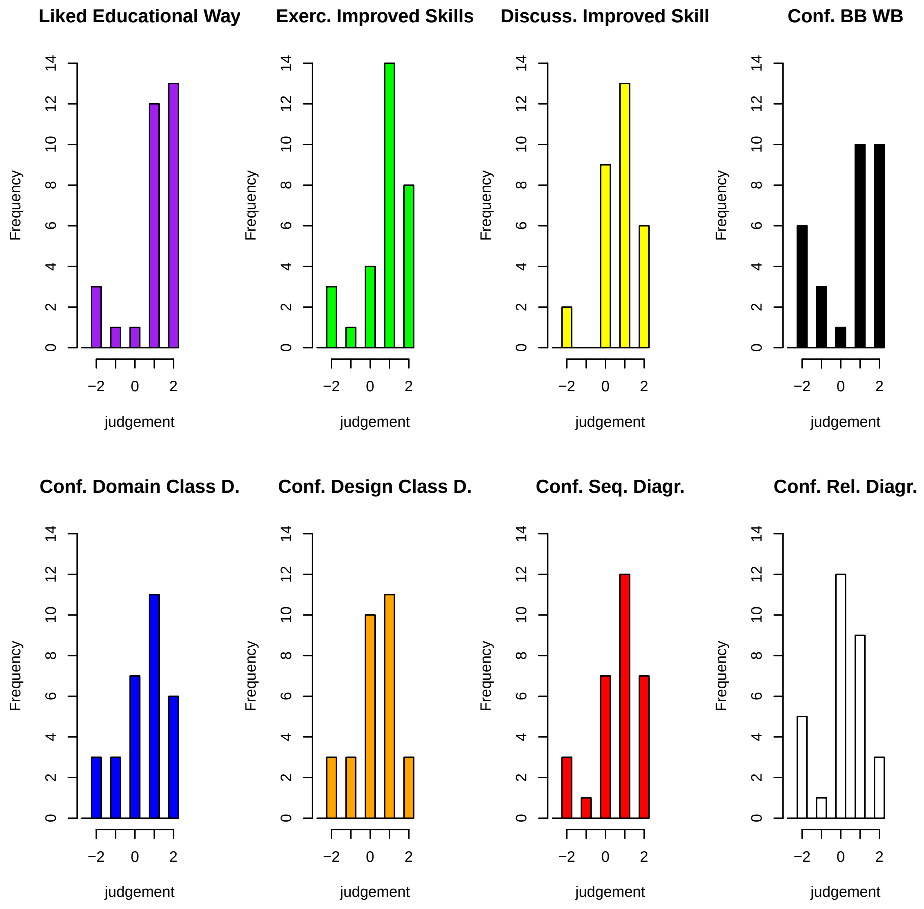


Figure 11.6: *Student Survey Responses (N=30)*

11.4.1 Scaled Questions

Figure 11.6 shows the histograms of the scaled questions of the questionnaire. To explain to which question an abbreviation in the histogram relates we list the questions with the abbreviation between the parentheses. We added statistical information: the mean and standard deviation. For every question we show a percentage. It is the percentage of the total respondents that judge neutral (0) or positive (+1,+2) which is notated as follows: j⩾0: 50%

- Did you like the way we used the exercise as a way of learning? (Liked Educational Way: M=1.03, SD=1.25) j⩾0: 87%
- Did the exercise help you to improve your modelling skills? (Exerc. Improved Skills: M=0.77, SD=1.19) j⩾0: 87%
- Did the discussion in the group help you to improve your modelling skills? (Discuss. Improved Skills: M=0.70, SD=1.02) j⩾0: 93%
- How confident are you that you understand the difference between black box perspective and white box perspective? (Conf. BB WB: M=0.5, SD=1.55) j⩾0: 70%
- How confident are you that you understand what a domain class diagram is? (Conf. Domain Class D.: M=0.47, SD=1.22) j⩾0: 80%
- How confident are you that you understand what a design class diagram is?(Conf Design Class D.: M=0.27, SD=1.11) j⩾0: 80%
- How confident are you that you understand what a sequence diagram is? (Conf. Seq. Diagr.: M=0.63, SD=1.19) j⩾0: 87%
- How confident are you that you understand the relation between the different diagrams? (Conf. Rel. Diagr. M=0.13, SD=1.20) j⩾0: 80%

The results show a positive judgement on all the scale questions. On all questions 70% - 93% of the subjects answer neutral or positive (+1 or +2). If one only considers the very positive judgements (+2) there is more variation.

Considering the mean (M=1.03) and the high amount of positive judgements (25 out of 30) of ‘Liked Educational Way’, students judge most positive about the educational approach, followed by the positive judgements about the improvement of their modelling skills.

What typical difficulties/challenges were discussed in your group while your were doing the modelling tasks?
“How detailed or simplified data modelling should be under different context”
“Which class belongs in which package. What functions and attributes to give the classes.”
At the moment what is the most difficult topic in modelling for you?
“ ... but I will continue my exercise to understand more about the relationships between the different diagrams”
“Constraining and limiting the modelling. You can include everything if you want. The most difficult is knowing where to draw the line. What’s included in this system? What’s included in this label etc.”
What did you learn from the exercise that you did not learn from the lectures?
“I learned more about refining the domain model as well as grasping a better understanding of the initial design concept.”
“gained a better understanding of the whole modelling process since we went from the start to almost end.”
Remarks about the lecture
“Really good exercise lecture and I wish we could have more lectures like that. I believe the presence of the TAs could have helped a bit more as they could be going around and help with questions.”
“Interesting. We should have more exercises like this.”
“A lot of fun. I love the competition part of it!!! But I think that whole groups are a bit too big for such a short time exercise. Its hard for everyone to contribute equally.”

Table 11.1: *General comments on the exercise*

11.4.2 Free Text Questions

Table 11.1 shows some of the comments the students had on the exercise. Here we list the questions we asked:

- Please write down (short sentences) what typical difficulties/challenges were discussed in your group while your were doing the modelling tasks.
- At the moment what is the most difficult topic in modelling for you?
- What did you learn from the exercise that you did not learn from the lectures?
- Any remarks about the lecture? Please write them down.

The answers on the free text questions were analysed and categorised.

Group discussion and difficulties The students mentioned the following matters as part of the group discussion or as a subject that they find difficult:

Translation of a case - students commented on the difficulties they have 'translating' a case-text into a conceptual model. The main step here is an analysis step: the discovering of key concepts in the domain. Students struggle when they have to decide whether something is a concept or not. They explain they have difficulties in linking new concepts to the concepts they identified earlier in the analysis of the case: do responsibilities belong together in one concept or do they need to be split across separate ones? Is a responsibility big enough to be its own class or can it be an operation of some class?

Organising - another matter that was pointed out is having difficulties in organising a design into layers. Students indicate they discussed about how to assign attributes and operations to the classes and how classes should be assigned to packages.

Detail - a point of discussion that emerges is how much detail should be used in a diagram? One student commented about where to draw the line: 'You can include everything if you want'. The following questions were asked: should attributes and operations be added? If so, how many? Should parameters be included in operations? Do we have to put multiplicities and labels on associations?

Layout - students have the need to discuss the layout of the diagram. They mention they sometimes discussed more about the aesthetics than the logic.

Different diagram types - students discuss about when to use which diagram type. Also, they are puzzled about the relations between the diagrams. We believe this relates to the consistency rules that apply to the diagrams, such as: if a method appears in a sequence diagram, what does this mean for the class diagram of the design of the system?

Learned from the exercise Students indicate they have learned several matters from the exercise. They relate to the topics found in Section *Group discussion and difficulties*. Students gained a better understanding in how to build diagrams (detail, organising, translation of a case). They comment they learned about refining the domain model by analysing the case, identifying key concepts and adding detail (detail, translation of a case). After attending the workshop they better understand the initial design concept (diagram types). Students mention 'how to easily update from the diagram' (detail) and learned about layering (layout, organising).

Related to the development process students comment they have learned about the different steps and roles of the diagrams in software modelling (diagram types) by seeing the whole development process from the beginning until the end.

Other remarks One student wrote that he/she learned 'about the importance in consensus for naming'. Another mentioned reflected 'our group thought about more domain aspects than other groups'. Some students suggested to work in smaller groups in order to let everybody participate equally.

11.5 Discussion

In this section we discuss the research questions that were presented in Section 11.1. Based on the results presented in Section 11.4 we aim to answer them.

11.5.1 RQ1: Can we integrate (UML)modelling and agile development into an educational workshop?

The workshop based approach enabled us to combine UML modelling and agile development. In a short time frame students were able to experience UML modelling as support during a Scrum based development process. We believe, by compressing real-world time into minutes, students benefit of seeing the whole development cycle in one lecture. They are able to quickly experience difficulties that arise because of decisions made just a couple of minutes earlier. Besides the usefulness students also mention it was 'fun' to do. Also the statistics (fig 11.6: Liked Educational Way) show this.

RQ1.1: Do students learn about the role the (UML) diagrams have in a development process? In the free text questions the students comment, because of the exercise they were able to see the 'whole picture' of the development process. They explained that they were able to see the relations between the different steps and the different models that they can use. The statistics show 40% of the students is neutral and another 40% is positive about seeing the relationship between the different UML diagrams. Only 20% has a negative feeling about this. Although, considering the mean of 'Conf. Rel. Diagr.' (0.13), students seem somewhat cautious to judge their understanding very positive, we believe the workshop helped for the larger part of the students.

RQ1.2: Does the workshop help students to improve their modelling skills? Students believe they improved their skills by doing the exercise. 73% has a positive judgement about this. 63% thinks the discussion in their groups helped improving their modelling skills. We expect that the improvement students judge about are: understanding the relationship between the different diagrams and development phases and refinement steps (mentioned in Table 11.1). The workshop actively emphasises these topics through practice. Of course additional long term practice is required for significant improvement in these skills and the variety of additional modelling skills.

11.5.2 RQ2: Which insights for modelling education do we gain from the students' group discussions during the workshop?

The statistics suggest us to use this kind of exercise more often. The 'Exerc. Improved skills' and 'Discuss Improved Skills' data show a positive judgement from the students. Also free text comments saying 'Really good exercise' or 'it was very interesting' indicate a, for students, motivating educational approach.

The workshop enables the students to discuss the difficulties they face during the modelling tasks. We believe that modelling in groups enabled the discussion between the students and causes them to argue about the choices they make. This is comparable with the pair-programming technique from agile development, where coders review each other during coding. The students were able to show us there is a variety of challenges they have to face. From the response on the free text questions, we were able to identify 5 topics students find difficult as shown in Section *Group discussion and difficulties*.

Maybe dividing the group into pairs, as suggested by some of the students could achieve even better results.

11.6 Limitations and Threats to Validity

At the moment the workshop does not reflect on the quality of the student software designs from the lecturer's perspective. It is limited by the judgements of the students' peers. This was done because of the focus on the role of the diagrams in the agile process rather than to have an implementable design.

Out of the 50-60 students, 30 participated in the questionnaire. We are aware that this could introduce a bias in our results.

We are also aware that we have analysed one particular case and have to be careful to generalise our findings.

11.7 Conclusion and Future Work

In this chapter we proposed a workshop that integrates UML modelling into agile development in order to: i) have an integrated software development approach that helps student to understand the role of (UML) modelling in a software development cycle and ii) collect information about students′ difficulties that arise during discussing the modelling task in a group. The results indicate that the students in general have a positive judgement about the workshop. The statistics also show that most students judge they have improved their modelling skills and that the workshop gave insight into the relationship between modelling and the software development process. We were able to categorise their comments. In future work we will add the results of the comments to our already collected student feedback on similar analysis and design assignments. We will use the data in order to expand our own educational modelling tools and educational approaches with proper feedback to guide students during their modelling tasks. As future work we like to explore a variation of this workshop in which we will include actual coding and exercise maintenance of the code.

Part VI

Reflection

Chapter 12

Conclusion and Future Work

In this chapter we summarise the conclusions and highlight the contributions of the research of this dissertation. We also relate the findings across chapters. In addition we make recommendations for approaches, interventions, and tooling for teaching software design.

12.1 Reading Guide

This chapter starts with a summary of the research objectives that are explored in this dissertation (Section 12.2). Subsequently it discusses the three main research questions (Sections 12.3, 12.4 and 12.5) with the following structure:

- main objectives,
- research approach, and
- recommendations for teaching software design

In addition, this chapter will continue with explaining the supporting role of our own developed research and educational tooling, as it affects all three research questions (Section 12.6). The chapter closes with future directions research could take (Section 12.7).

12.2 Research Objectives

The research in this thesis supports the following main objective:

How can we improve the ways of teaching software design?

We decided on the following scoping of our research:

- We focus on online learning environments, such as MOOCS. This implies that the tool for doing modelling, the tool for its assessment, and the tools for generating feedback during modelling have to fit into an online learning environment.
- We put an emphasis on the learning of designing. We support the opinion that the student challenge is about the comprehension of the design – an abstract and difficult topic – not on the specific UML notations.

In order to explore the main research objective, we refined it by the following three research questions:

RQ1 Can we assess how good students are at designing software?

RQ2 What guidance do students need to improve their understanding of designing software?

RQ3 How can we increase the motivation and engagement of students for learning software design?

The relationship between the research questions and research topics is shown in Figure 12.1). In the figure is seen that a research question relates to a specific topic area (dashed, rounded rectangles). Each topic area is divided into subtopics that are explored in the different chapters of the dissertation. Some chapters support the research of other chapters (light arrow).

Software design education can be explored in many different ways. In our research we mainly focused on the following topics:

- Assessing students' skills of software design: we wanted to assess students' comprehension of software design (Chapter 2). We explored measuring students' comprehension with a test that is based on general software design principles.
- The role of guidance and feedback in the process of students' learning of software design: we want to understand the way students reason and what strategies they

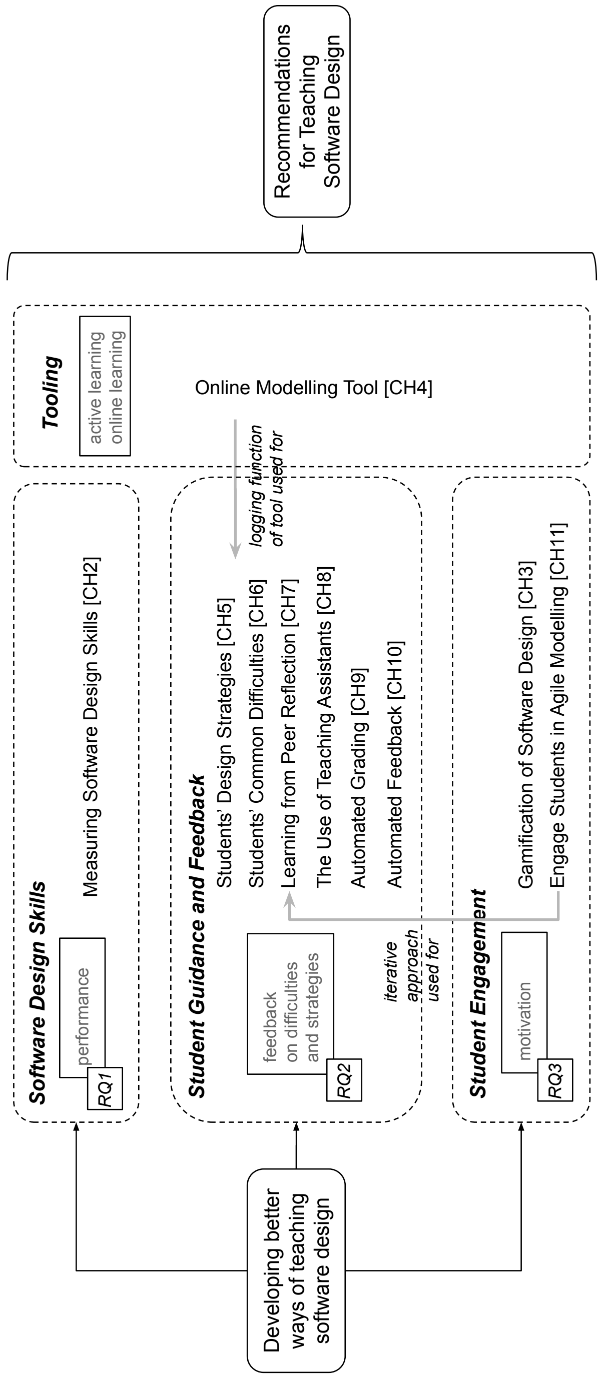

Developing better ways of teaching software design
Software Design Skills
performance
RQ1
Measuring Software Design Skills [CH2]
Student Guidance and Feedback
feedback on difficulties and strategies
RQ2
Students' Design Strategies [CH5]
Students' Common Difficulties [CH6]
Learning from Peer Reflection [CH7]
The Use of Teaching Assistants [CH8]
Automated Grading [CH9]
Automated Feedback [CH10]
iterative approach used for
Student Engagement
motivation
RQ3
Gamification of Software Design [CH3]
Engage Students in Agile Modelling [CH11]
Tooling
active learning
online learning
Online Modelling Tool [CH4]
logging function of tool used for
Recommendations for Teaching Software Design

Figure 12.1: *Overview of the research themes and their corresponding chapters.*

use when making decisions while making a software design (Chapter 5). We aimed to reveal students' common problems while making a software design (Chapter 6). Next to revealing the above mentioned strategies and difficulties, we analysed the use of peer-feedback by fellow students (Chapter 7) and guidance and feedback of teaching assistants (Chapter 8) in order to collect best practices.

The knowledge of the common problems and strategies helped to develop tools and educational approaches where feedback and guidance play a role. We explored automated grading with the use of machine learning (Chapter 9) and automated feedback by extending our tools with a feedback agent (Chapter 10)

- Approaches for improving the engagement of students for learning software design: we studied the use of gamification for engaging students into software design. We developed a puzzle game for the learning of software design with UML class diagrams (Chapter 3). From that game we used the concept of an avatar giving feedback and comparing student solutions against example solutions in our online UML editor.

  Another approach for engaging students in software design was the exploration of the use of an interactive workshop on agile software design with UML (Chapter 11).

- Research tooling for supporting the aforementioned topics: In particular, we desired to develop an assessment tool for measuring software design comprehension and a tool for software designing that could monitor design activity (Chapters 2, 4).

In the following sections we discuss *RQ1*, *RQ2* and *RQ3* and relate the findings and contributions from different chapters. For the various angles of our research questions, we provide closing recommendations for teaching software design. For an overview of all the recommendations that emerged from the research in this dissertation, along with the problems they address we refer to Table A.1 in Appendix A.

12.3 *RQ1* – Can We Asses How Good Students Are at Designing Software?

We start by summarising the aspect of assessing students' software design skills. Then, we explain the approach we followed in our research. Subsequently we explain what effects the findings had on the research direction of this dissertation. We conclude this section with recommendations for teaching software design.

12.3.1 Main Objectives

The main objectives for exploring RQ1 (*Can We Asses How Good Students Are at Designing Software?*) were the following:

- To have an objective and repeatable instrument that could be used to provide insight into the software design skills of students. A unique aspect of our approach is that we focused on the understanding of software design principles.
- To be able to compare skills between students and groups of students. This enables us to explore different educational interventions. One practical constraint on the assessment instrument was that it should be applicable to a large group of students. This led us into the direction of an online test.
- To have insight into 'learning yield' of educational interventions (by looking at the difference between the after and before level of the assessment).
- To understand the relation between design skills and generic skills. Therefore, we studied the relationship between the abstract reasoning skills and language skills and software design comprehension.

Next, we move on explaining our research approach.

12.3.2 Research Approach

Our initial attempt towards an instrument was to construct an online multiple choice test for measuring the comprehension of software design. The test consisted of questions based on design principles (Chapter 2). Each question presented alternative solutions for designs. These alternatives differed in the way they satisfied a particular software design principle. In this test students had to choose the best solution out of four alternatives. We applied the test to several groups of students in software design courses. We conducted a pre-test at the beginning of a software analysis and design course and a post-test at the end of that course. By running these experiments we found that our instrument was suitable for measuring learning yield. Moreover, the results of the tests were consistent with the course examinations. We consider this as a validation of our instrument. Also, the results demonstrate the scalability (number of students) of the test. Further more, we found that knowledge of the UML notation did not relate to the performance of the students on software design comprehension. This result is in line with our intentions: we aimed to design a test that measures software design skills but at the same time required little knowledge of any particular software design notation.

We explored whether abstract reasoning and language skills influence the performance on the software design skills test. In order to measure this we conducted three tests: one on visual reasoning (Raven test), one on verbal reasoning (verbal analogies) and one that tested general language proficiency (C-test). We found that visual- and verbal reasoning correlate with the performance on the design skills test. An implication of this is to find complementary teaching methods for students that have a lower talent for abstract reasoning. The language proficiency level did not correlate with the performance on the design skills test. We assume this can be explained by the high level of knowledge of the English language in the sample groups. The test results may not generalise to student populations that have a poorer understanding of the English language.

12.3.3 Effect On Research Directions

After having gained experience with performing the design skill test, we found some limitations. In its current shape, the online test does not call on any synthetic capabilities/skills for creating software designs. Instead the test seems to address the ability to recognise good and bad designs - which also involves some analysis skills. The test was usable for diagnosing which design principles students did not understand. Using this test, we are able to guide students into the direction of explanations about the topics on which they achieved low scores.

However, even if students master the design principles, we suspected that *creating* a design involves additional types of reasoning steps (such as making design decisions) that cause difficulties. The design skills test does not cover such skills. These limitations were due to the multiple-choice character of the test. In the subsequent steps in our research, we tried to remedy these limitations. We created an online interactive design environment, WebUML, which enabled us to monitor the synthetic process of constructing software designs. WebUML enabled us to explore higher levels of Bloom's taxonomy (from *apply* and up) than we could with the design skills test. In particular, we started exploring methods for identifying the difficulties students encounter during the design process. More about WebUML in Section 12.6

By using our tools (the online test and WebUML), additional questions arose. These questions relate to the research questions that are discussed in the next sections:

- We want to understand why students have difficulties with specific topics in the area of software design and what specific design decisions they make that lead to their solution. If we understand this, we could improve our guidance and/or feedback towards students (RQ2).

- We want to gain insight into what kind of learning interventions increase the engagement in the learning of software design topics (RQ3).

12.3.4 Recommendations for Teaching Software Design

This section often refers to recommendations for education that can be found in Table A.1

In this section we summarise recommendations for teaching software design distilled from the research of RQ1.

Based on the experience we have with assessing students with our online test and WebUML, we recommend teachers to use design principles as a basis for measuring software design understanding (Table A.1 Recommendations 18, 19 and 22). Although UML syntax should be discussed in courses, we recommend teachers to focus on design principles rather than syntax (Table A.1 Recommendation 20). The understanding of design principles lays the foundation for understanding complex software design challenges.

The test covers the following topics about software design principles: Dependencies, Cohesion, Dependencies and Changes, Maintainability, Dependencies and Reuse, Responsibility, Extensibility, Reuse Information Hiding, and Coupling. Students can use the online design skills test for self-reflection on their software design skills. Such a self-test produces reports that present the number of correct and incorrect answers per topic. Through these reports students can uncover their own deficiencies.

12.4 *RQ2* – What guidance do students need to improve their understanding of designing software?

The second aspect we describe is our approach for understanding what guidance students need when learning software design. First we mention our main objective. Then we describe our research approach in which we used our own tooling. Subsequently we show that the research revealed different strategies that students use for designing software and discuss the common difficulties that they face. We close this section with the recommendations we have for education based on our findings.

12.4.1 Main Objectives

The main objectives for exploring RQ2 (*What guidance do students need to improve their understanding of designing software?*) were the following:

- To have a set of guides that can help students when learning software design.
- To explore if the guidance approach could be implemented in an online environment.
- To be able to recognise students approaches to guide them to a proper solution.

We explored the above-mentioned objective in online settings with large number of students. This exploration identified the need for students to perform self-assessments. For such self-assessment we studied the automation of guiding and grading. Thus, we extended our objective:

- To offer automated didactic guidance for students during their software design tasks.
- To explore the possibility of an automated grading system in an online software design environment.

12.4.2 Research Approach

In researching which guidance helps students best, we first needed to understand what difficulties are common for students and what strategies they use when performing a design task. Knowing students' difficulties and strategies gives us leads to possible guidance that can be used in educational interventions.

For exploring students' strategies, we started with equipping our WebUML tool with logging functionality that logged user actions, such as creating of UML elements, deleting elements, moving elements and modifying elements (see Chapter 5). During a students' modelling session, all items were logged with their corresponding time stamp. We were able to analyse the logs with our own developed tool *LogViz* that visualises the student's modelling session.

The analysis revealed two dominant strategies that students use when designing class diagrams:

- Breadth-first: an approach where students first make an overall high level class design, including associations and add details in the form of attributes and operations later.
- Depth-first: an approach where students first create a class with much detail (attributes and operations) and then relate it to another class (associate) with much detail.

We compared the grades of the students that used the breadth- and depth-first strategies. We could not find a significant difference that shows one strategy is better that the other (see Chapter 6). We did however find that the Depth First strategy used fewer re-orderings of the layout of the diagram. Hence, if we would consider 'Movement'/Re-ordering of classes as a measure of efficiency, then this would make Depth First more efficient than Breadth First.

Next to the strategies, analysis shows that students spent a lot of time on adjusting layout and understanding the task. Both adjusting layout and understanding the task also show up in other research, described in Chapters 7 and 8. In our research (see Chapter 6) a relation between the layout and the grade was shown. It seems that students that organise their diagrams better also create a design that is better in terms of design principles. Therefore, next to the focus on design principles, lecturers should guide their students to care about the layout of their diagrams.

For understanding the common difficulties students have, we added a form to Web-UML. With this form, we were able to collect difficulties students have when performing modelling tasks. During several practical sessions, we asked students to fill in and submit the form with questions and problems that arose during the performance of their tasks. We were able to categorise the common ones, based on questions students have (see Chapter 6). These categories are:

- task comprehension,
- tool usage,
- tool feedback,
- UML/OO comprehension, and
- UML syntax/notation.

Surprisingly, a low amount of questions was in the 'UML/OO comprehension' category. Knowing these common problems helps to create future interventions for learning software design.

We found that a limitation of our exploration into students' strategies and difficulties was that they did not reveal the thinking process of the students that they performed while creating software designs. Students tend to summarise and abstract their own thoughts or are unable to formulate what they don't understand (unconscious incompetence from the stages of competence [22]). In this way we could not obtain a deeper understanding of students' reasoning.

Because of the aforementioned limitation, we designed a study in which we used peer reflection to reveal more detail about the problems that arise with students while constructing a software design. During a software analysis and design task that was part of a running course, we asked students to be the peer-reflector of their fellow student (Chapter 7). Students worked in pairs and were asked to discuss their intermediate results on the basis of questions that were focused on analysis- and design-comprehension and modelling-skills. This approach triggered the students to actively discuss and critically reflect their task. A lecturer is guiding the process, but can also actively be approached by students for questions. The lecturers that were involved during the peer-reflection sessions were positive about this approach and noticed students were more involved and active than usual. In addition, this approach can be used by students as a self-guiding approach outside of the classroom. By analysing the student peer-reflections, we found 'points of concerns' that can be used when creating courses or for subsequent research.

Still, a limitation of this research was the low amount of detail students provide when documenting their answers. We can say that we found more detail than the research with the online form in WebUML (Chapter 6). Eventually this led to an extended collection of common difficulties, but not yet an in-depth insight into students' thinking strategies.

Another limitation of peer-reflection we see is that students discuss matters based on their prior knowledge and assumptions. This means that such a discussion can only lead to a certain level of learning yield. Because of the small age gap between novice and senior students, we came up with the idea to explore the role of teaching assistants (TAs). Teaching assistants are selected because of their (high) competence in the subject of the course. Using this competence, we could enable similar discussion as in the peer-reflection approach, but with a deeper level of discussion. Additional motivations for conducting this research were:

- to extend our knowledge on common difficulties that students have.
- to categorise difficulties, and
- to have a categorisation of support approaches used by the TAs.

In a case study, we analysed the involvement of the aforementioned TAs as support of students' learning during a bachelor course on software analysis and design. The TAs were involved in:

- supervision sessions, in which the students worked in groups on a practical assignment.
- weekly meetings with the lecturers running the course, and
- giving feedback to the students on the work they handed in.

The data that was analysed came from transcribed interviews with the TAs, student answers on questionnaires about the TA support and written feedback of the TAs on the hand-ins.

We found that TAs are mostly consulted during abstract reasoning tasks, such as electing conceptual elements in a diagram (creating classes based on some description). In this research tool support was less of an issue. Although we suspect TAs were involved explaining the tool in a more natural way. The explanation of tools by TAs was not initiated by problems that occurred. We observed that students have high expectations of TAs. The TAs are 'forced' into a lecturer role. Just like with lecturers, TAs are expected to give confirmation ("Is this the right way?") and should give clear directions.

We observed that typically for SE assignments, student have difficulties with TAs (just as lecturers) giving inconsistent answers (from the students point of view). We suggest to use our categorisation of common software design difficulties and the categorisation of guiding approaches to create profiles and/or programs to prepare future teaching assistants.

The exploration of student reasoning in online settings led us to the idea of exploring the use of automated grading and guidance in order to enable the online tooling for self-assessment.

We explored the application of machine learning for grading student solutions (see Chapter 9). We approached grading as a prediction problem and trained models for the prediction of the grades of software design assignments, with a regression model as well as classification models (numeric versus symbolic outcomes, at different scales). We tested two different sets of features. One that used a model that was built with a feature set that focused on generic (solution elements that dominate the design construct, i.e. inheritance or aggregation) modelling elements and second experiment that used a feature set that consisted of specific elements (solution elements that carried specific names, i.e. 'car'). We conclude that grading student models with machine learning with a 10 point scale is not accurate. Although, using a 3 point scale in the second experiment resulted in a better accuracy (69.42%) than the 5 point scale of the first experiment (65.24%), this is still not reliable to use for automated grading. For a rough indication of design quality it could be integrated in online tools. This led us to the idea of having a guide that could do intermediate judgements during the task and provide the student of feedback.

We explored the application of a didactic software component, a pedagogical agent (see Chapter 10). The pedagogical agent guides the students through their software design task. The agent can be asked for guiding feedback on demand during software design tasks. The agent compares a model solution with the (intermediate) submission of a student. In this way the agent guides the student towards the solution of an assignment. Early findings of a tested prototype show that students appreciate the provided guiding feedback as confirmation of being on the right track.

12.4.3 Recommendations for Teaching Software Design

This section often refers to recommendations for education that can be found in Table A.1

In this section we summarise recommendations for teaching software design distilled from the research of RQ2.

Although more research is needed to reveal more detailed information about strategies students use when solving software design problems, we advise lecturers to be aware of the difficulties students have in understanding and constructing software designs. We suggest to use activating teaching forms for gaining insight into students' struggles with learning software design.

We recommend using approaches were students discuss intermediate results in peer-reflection settings (Table A.1 Recommendations 22 and 26). By discussing intermediate results students are more willing to reflect on and discuss their progress. This discussion not only increases the confidence of the student, it also activates the student to formulate questions and approach TAs or lecturers more often.

Besides the learning of the syntax of modelling languages and the comprehension of software design principles, lectures should focus on the layout and the organisational aspect of the diagrams used for modelling (Table A.1 Recommendation 29). A clear layout supports a better understanding of a diagram, even when constructing a diagram.

Use the logging of students' design tasks in order to enable teachers to analyse students' design approaches and reflect on the patterns they observe. This can be applied during lectures or other feedback moments. By using a logging system or peer review approaches, students are enabled to reflect on each other and also on themselves. Lectures can use the reflections to focus on those matters in follow-up lectures. We see feedback as a key factor for successfully understand the topic of software design (Table A.1 Recommendations 2, 7, 9, 13 and 19).

By knowing a student's strategies, lecturers might have to choose another approach when supporting a student. It is advisable to collect and discuss the problems that arise in student-groups.

When employing TAs during a course, we suggest to plan weekly meetings with the TAs. On the one hand, these meetings should discuss the possible solutions for the assignments (so that the TAs understand the range of possibilities). On the other hand they serve as a bridge between the lecturer and the students that follow the course (Table A.1 Recommendations 7 and 13). In addition, we recommend to prepare TAs before a course starts. Our categorisations of common challenges students face where TA support can help, can be used to create profiles for the selection of TAs. Also our findings can be used for creating programs for training future teaching assistants.

12.5 *RQ3* – How can we increase the motivation and engagement of students for learning software design?

The third aspect we studied are approaches for engaging students in activities for learning software design. First we summarise our main objective. Second we discuss two different research approaches in which we explored the application of gamification and we explored an interactive workshop as educational tool. We close with recommendations for education based on this research.

12.5.1 Main Objectives

For studying engagement, we explore the use of gamification. A main objective then becomes the design of a gamification approach that satisfies the following goals:

- It has a positive influence on the engagement of students during software design activities.
- It can be used to breakdown the difficult topic of software design into smaller parts that could be exercised separately and repeatedly.

As an alternative instrument for student engagement, we studied the use of an interactive workshop. Key elements of this workshop are i) the simulation of a 'real life' agile development process, and ii) stimulation of peer-modelling, peer-discussion and peer-review between students while making a UML software design.

For this workshop, we studied:

- How the workshop affects the student's motivation for the learning of software modelling.
- Whether the workshop benefits the understanding of the relationships between the UML models used in different phases of software development.

12.5.2 Research Approach

From our experience with several UML editors, we believe that an integration of a learning platform with a UML editor could be a good approach. This led us to the construction of a game that was based on the idea of a UML editor: The Art of Software Design.

We developed an educational game called *The Art of Software Design* (AoSD). AoSD uses a gamified UML editor as the basis of the game (Chapter 3). The game supports the "learning by doing" approach. The aim of the game is to solve a series of puzzles, where puzzles correspond to the completion or creation of a design or a design-fragment.

By using actual software design concepts, such as class or association, as game elements we aim to motivate the students for learning design principles such as cohesion, coupling and modularity. AoSD was designed in such a way that it offers a breakdown of topics in increasing level of difficulty. Students are able to choose the order in which they want to work on topics - while staying within logical paths of learning demonstrating prerequisite knowledge before moving on to topics that build on this more basic knowledge. The ability to choose your path (implementing a need for autonomy [106]) in a video game supports the engagement of the player.

The AoSD game includes a scoring mechanism (which implements feedback on competence and achievement [106]). The scoring mechanism works interactively: it shows the updates to the score while a students is making changes to his/her solution. The feedback on the score was provided through a combination of visual and audio effects.

A preliminary study with AoSD showed that students valued the freedom of choice in learning paths. When a student got stuck, it was possible to choose another puzzle and come back to the difficult one later. With this approach, the student remained engaged in the game. In the same study, we also found that textual instruction to a puzzle should be kept short. Otherwise, students are more likely to skip such instructions. We found that subjects that were not used to UML terminology were using UML terminology after playing the game.

A limitation of AoSD is that AoSD is not equipped with the possibility to monitor the activities of students which can be used to determine their strategies. As mentioned earlier, monitoring strategies or modelling steps could give us insight into students reasoning.

The use of WebUML in our own classroom and experiences in sister courses on agile software development projects led us to the idea of integrating software design and agile practices in a workshop. The aim of this combination was to explore if an integrated approach could contribute to the understanding of software design and the role it plays in the software development process and provide a motivating approach for learning software design. In Chapter 11, we presented a workshop approach to address the difficulties students have with understanding the relationship between the different phases of software development and software modelling. By performing short sprints, inspired by the LEGO scrum workshop [76], students delivered iterations of their software designs. Because the sprints were time-boxed, the students were confronted with their design decisions almost instantly. By using a follow up questionnaire we used the students' self reflection as a measure of their understanding of modelling in a software development process. The students indicated they are better at relating the different UML diagrams with the different phases of software development. By choosing the form of a short workshop we aimed to let students discover the overall overview of a software project and the (UML) models that can support the different stages of software development. The students judged that they learned from the workshop and suggested to use this form of exercising more often.

12.5.3 Recommendations for Teaching Software Design

This section often refers to recommendations for education that can be found in Table A.1

In this section we summarise recommendations for teaching software design distilled from the research of RQ3.

We believe that the learning of software design requires practising of software design (Table A.1 Recommendation 5). This should be facilitated by a tool that supports the learning by doing approach. As explored with the AoSD game, we recommend to breakdown the difficult topic of software design into smaller bits that are linked to common software design principles, such as coupling, cohesion or modularity (Table

A.1 Recommendations 18 and 19).

Although practising with isolated assignments can be helpful for addressing the learning of the syntax of a modelling language, we advice to also integrate the modelling of software designs into the overall software development process (Table A.1 Recommendation 34). Especially with the use of UML and the multiple ways a diagram can be used, it is important that students understand the purpose of the diagram in the software development process. In this way they are able to see the 'big picture'.

We experienced students are more engaged in software design activities when gamification is applied (Table A.1 Recommendation 27). Game mechanics that give students freedom of choice, challenges, competition and feedback are engaging students in the software development process. As seen with AoSD, UML modelling can be integrated well into a game-based learning approach.

Use modelling tools in a simulation (workshop) of the software development cycle. Students can judge about their own understanding of modelling during the development process they experience (Table A.1 Recommendations 7, 13 and 19). They will be able to discover their understanding of the difference between analysis and design, and the purpose of the different diagrams that can be used during the different phases of software development.

12.6 Supportive Tooling

In this research we developed various types of online tools support for conducting our research. In this section we explain how the two most important tools support the different research questions of this dissertation. In addition we close this section with recommendations for teaching software design that are distilled from the experience of working with these tools.

12.6.1 Design Skills Test

In our research we were in the need for a tool that was easy to scale up. If we wanted to be able to perform statistical analysis on our data, we needed to be equipped with a tool that could easily handle larger groups of students. We used Limesurvey [1] as a basis. Limesurvey is a web based survey tool. We needed a tool that met the following requirements:

[1] https://www.limesurvey.org

- time restriction (we added this to Limesurvey)
- statistical data (e.g. right / wrong answers, overall time spend, time spend per question)
- combine question text with images
- anonymous answers - this was provided by a unique survey token
- export of data for further statistical analysis (e.g. in R or SPSS)

The design skills test supports RQ1 and was used as a diagnostic tool in several of the classroom settings were research was performed for this dissertation.

12.6.2 WebUML

WebUML is an online UML editor that enables us to monitor students while constructing their software design (Chapter 4). It was especially developed for educational purposes. *WebUML* was developed for better fulfilling students' needs. In different researches students mention the frustration of using modelling tools. Installation and configuration of tools takes a lot of the students' time and does not always yield a working set-up. By using an easy to install application in combination with a simplified subset of UML we aim to contribute to the students' motivation for learning.

Using this instrument we could analyse several modelling - and design activities, such as creation, deletion and editing of UML elements. These modelling activities are recorded by a logging system that is part of the WebUML editor. In addition to our gained insights about the comprehension level of software design, our aim with *WebUML* was to collect data about the modelling/design process of students. With this data we were able to analyse strategies students use to create their designs.

We equipped *WebUML* with the capability of logging students activities and recording students' questions. With these features we aim to monitor students strategies and try to identify difficulties. Lecturers can give their feedback to the students, based on the recorded data. By supporting the student's learning process we aim to keep students motivated. By choosing an online platform *WebUML* fits well in modern tools and online platforms, such as MOOCs.

WebUML had to fulfil the following requirements:

- handling of large scale groups

- monitoring of student software design activities, such as creation of UML elements and edits of previous actions, and storing log files
- tool interaction
- extensible for future modules that interpret student solutions, such as the feedback agent (Chapter 10)

WebUML supports the research of Chapters 5, 6, 7, 9 and 10.

While using WebUML during our experiments we noticed that the tool was easy to use after a small set of instructions for both students and lecturers. In experiments, we were able to scale up to larger groups of students.

While the online test was focused on testing skills based on knowledge or comprehension, WebUML facilitated the judgement of the students' design process.

Recommendations for Teaching Software Design *This section often refers to recommendations for education that can be found in Table A.1*

In this section we summarise recommendations for teaching software design learned from the use of different tools.

Teachers should offer students an easy to use tool that uses a simple subset of UML. WebUML enables students to practice software design assignments in a simple environment (Table A.1 Recommendation 5). In addition, we recommend a feedback mechanism, such as our pedagogical agent for providing feedback during practising designing (Table A.1 Recommendations 7 and 9).

Students give up on using modelling tools because of complex installation and configuration problems. Industrial tools more often frustrate students because of their steep learning curve. Courses on Software Design should use tools that are easy to install and easy to use (Table A.1 Recommendation 33). We advise to use a small subset of the UML notation when teaching novices. In this way we stay away from specific object-oriented dependent principles and focus on general software design principles.

12.7 Future Work

For future research we propose to refine our online test so that the regression model can be used by lecturers to test their students and to choose appropriate interventions.

One of the directions that could be interesting to pursue is to use the test as a diagnostic instrument for students.

In addition, for further research in assessing students' software design capabilities, an extended version of our online test could be explored. This can be done by integrating WebUML and the online test. With this extension also the synthetic skills for creating a software design can be assessed. We propose to set up a similar experiment as described in Chapter 2 with a new set of questions that target synthetic skills.

To integrate WebUML into other systems, it should be extended to better fit users needs. We want to test our tools as part of other educational suites, i.e. e-learning or MOOC-like environments. For example as suggested by Vesin et al. ([155]) In general more research should be conducted on the use of modelling tools in education. WebUML could be enriched with the outcomes of this research.

Further research is needed to demonstrate the learning effect of our game based approach. We suggest to conduct a study that uses 'The Art of Software Design' to gain more insights into the validity of the scoring metrics that were used and the assumed learning effect.

To let students learn from the experience to go through a complete software development process, the agile UML modelling workshop could be extended to also include code generation or coding steps. In this way, students also reflect on the final product. Moreover, even maintenance steps could be integrated.

Future research should continue to explore students' strategies and difficulties in more detail, with the aim to develop educational programs in the field of software design that fit students' profiles better. Approaches have to be found that reveal more of the detailed reasoning and decision making steps that students take during their design-effort. For example, a combination of peer-reflection sessions, performing think-out-loud and follow-up interviews about the students' decision making could lead to new insights.

We see a huge opportunity for automated guiding and grading feedback systems in online learning environments. The explored machine learning and pedagogical agent approaches could be developed to a mature didactic tool which improves the quality of self assessment as part of self-study material and/or environments.

Bibliography

[1] Ritu Agarwal and AP Sinha. Object-oriented modeling with UML: a study of developers' perceptions. *Communications of the ACM*, 46(9):248–256, 2003. (cited on page 79).

[2] Seiko Akayama, Marion Brandsteidl, Birgit Demuth, Kenji Hisazumi, Timothy C Lethbridge, Perdita Stevens, and Dave R. Stikkolorum. Tool use in software modelling education: state of the art and research directions. In *the Educators' Symposium co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, 2013. (cited on pages 8, 63, 65, 104, 117 and 145).

[3] Vincent Aleven and Kenneth R Koedinger. Limitations of student control: Do students know when they need help? In *International Conference on Intelligent Tutoring Systems*, pages 292–303. Springer, 2000. (cited on page 170).

[4] Vincent Aleven, Elmar Stahl, Silke Schworm, Frank Fischer, and Raven Wallace. Help seeking and help design in interactive learning environments. *Review of educational research*, 73(3):277–320, 2003. (cited on page 170).

[5] Scott W. Ambler. Agile model driven development is good enough. *IEEE Software*, 20(5):71–72, 2003. (cited on pages 110 and 190).

[6] Helen Anckar. Providing automated feedback on software design for novice designers. BSc thesis, Goteborg University, 2015. (cited on page 153).

[7] John R Anderson, Frederick G Conrad, and Albert T Corbett. Skill acquisition and the lisp tutor. *Cognitive Science*, 13(4):467–505, 1989. (cited on page 170).

[8] Cynthia Andres and Kent Beck. Extreme programming explained: Embrace change. *Reading: Addison-Wesley Professional*, 2004. (cited on page 110).

[9] Nikolaos Avouris, Vassilis Komis, Georgios Fiotakis, Meletis Margaritis, and Eleni Voyiatzaki. Logging of fingertip actions is not enough for analysis of learning activities. In *12th Int. Conf. AI Edu. (AIED 05) Workshop 1: Usage analysis in learning systems*, pages 1–8, 2005. (cited on page 79).

[10] Nilufar Baghaei and Antonija Mitrovic. COLLECT-UML : Supporting Individual and Collaborative Learning of UML Class Diagrams in a Constraint-Based Intelligent Tutoring System. *n R. Khosla, R. Hewlett & L. Jain (Eds.) Proc. KES*, pages 458–464, 2005. (cited on page 169).

[11] Nilufar Baghaei, Antonija Mitrovic, and Warwick Irwin. Supporting collaborative learning and problem-solving in a constraint-based CSCL environment for UML class diagrams. *International Journal of Computer-Supported Collaborative Learning*, 2(2-3):159–190, 2007. (cited on page 66).

[12] Lut Baten. The c-test revisited a freeware package for placing efl students of business english in the new bachelor-master structure. *BELL: Belgian Journal of English Language and Literatures*, 2:301–315, 2004. (cited on page 30).

[13] Mordechai Ben-Ari. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–74, 2001. (cited on page 108).

[14] Jens Bennedssen and Michael E. Caspersen. Abstraction ability as an indicator of success for learning computing science? In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 15–26, New York, NY, USA, 2008. ACM. (cited on page 27).

[15] Marc Berges, Andreas Mühling, and Peter Hubwieser. The gap between knowledge and ability. In *Proceedings of the 12th Koli Calling international conference on computing education research*, pages 126–134. ACM, 2012. (cited on page 44).

[16] Gustav Bergström. Using machine learning to evaluate the layout quality of uml class diagrams. M.sc. thesis, Chalmers and Göteborg University, Sweden, April 2021. supervisor: M.R.V. Chaudron. (cited on page 167).

[17] Weiyi Bian, Omar Alam, and Jörg Kienzle. Automated grading of class diagrams. In *Proceedings of the Educators' Symposium at the 22nd International Conference on Model Driven Engineering Languages and Systems*, 2019. (cited on page 153).

[18] Paulo Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proc. 1st int. conf. learning analytics and knowledge*, pages 110–116. ACM, 2011. (cited on page 79).

[19] Narasimha Bolloju and Felix S.K. Leung. Assisting novice analysts in developing quality conceptual models with UML. *Communications of the ACM*, 49(7):108–112, 2006. (cited on page 110).

[20] Grady Booch. Object-oriented design. *ACM SIGAda Ada Letters*, 1(3):64–76, 1982. (cited on pages 4 and 7).

[21] Nadia Boudewijn. Automated grading of java assignments. Master's thesis, Utrecht University, The Netherlands, 2016. (cited on page 154).

[22] Martin M Broadwell. Teaching for learning (xvi). *The Gospel Guardian*, 20(41):1–3, 1969. (cited on page 214).

[23] PJ Burton and RE Bruhn. Using UML to facilitate the teaching of object-oriented systems analysis and design. *Journal of Computing Sciences in Colleges*, 19(3):278–290, 2004. (cited on page 41).

[24] Weiqin Chen, Roger Heggernes Pedersen, and Øystein Pettersen. CoLeMo: A collaborative learning environment for UML modelling. *Interactive Learning Environments*, 14(May 2014):233–249, 2006. (cited on pages 66 and 169).

[25] S R Chidamber and C F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. (cited on pages 53 and 167).

[26] Jan Claes, Irene Vanderfeesten, Jakob Pinggera, HajoA. Reijers, Barbara Weber, and Geert Poels. A visual analysis of the process of process modeling. *Information Systems and e-Business Management*, 13(1):147–190, 2015. (cited on page 93).

[27] Jan Claes, Irene Vanderfeesten, Hajo a Reijers, Jakob Pinggera, Matthias Weidlich, Stefan Zugal, Dirk Fahland, Barbara Weber, and Jan Mendling. Tying Process Model Quality to the Modeling Process : The Impact of Structuring , Movement , and Speed Background on the Process of Process Modeling. *10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, pages 1–16, 2012. (cited on page 79).

[28] J. Cohen. *Statistical power analysis for the behavioral sciences*. Erlbaum, 1988. (cited on page 32).

[29] Eric Crahen, Carl Alphonce, and Phil Ventura. QuickUML: a beginner's UML tool. In *OOPSLA '02 Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '02, pages 62–63, NY, USA, 2002. ACM New York. (cited on page 66).

[30] Holger Danielsiek, Jan Vahrenhold, Peter Hubwieser, Johannes Krugel, Johannes Magenheim, Laura Ohrndorf, Daniel Ossenschmidt, and Niclas Schaper. Undergraduate teaching assistants in computer science: Teaching-related beliefs, tasks, and competences. In *IEEE Global Engineering Education Conference, EDUCON*, pages 718–725, 2017. (cited on pages 128 and 148).

[31] Oswald de Bruin. The art of software design, creating an educational game teaching software design. Master's thesis, Leiden University, 2012. (cited on page 34).

[32] Birgit Demuth and Dave R. Stikkolorum, editors. *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 29, 2014*, volume 1346 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014. No citations.

[33] John Dewey. *Experience and education*. Simon and Schuster, 2007. (cited on page 44).

[34] Serkan Dinçer and Ahmet Doğanay. The effects of multiple-pedagogical agents on learners' academic success, motivation, and cognitive load. *Computers and Education*, 111:74–100, 2017. (cited on page 172).

[35] Brian Dobing and Jeffrey Parsons. Dimensions of uml diagram use: a survey of practitioners. *Journal of Database Management (JDM)*, 19(1):1–18, 2008. (cited on page 6).

[36] João M Fernandes and Sónia M Sousa. Playscrum-a card game to learn the scrum agile method. In *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2010 Second International Conference on*, pages 52–59. IEEE, 2010. (cited on page 191).

[37] Kiko Fernandez-Reyes, Dave Clarke, and Janina Hornbach. The impact of opt-in gamification on students' grades in a software design course. *21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS-Companion 2018*, 3:90–97, 2018. (cited on page 44).

[38] L Dee Fink. *Creating significant learning experiences: An integrated approach to designing college courses*. John Wiley & Sons, 2013. (cited on page 10).

[39] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23):8410–8415, 2014. (cited on page 146).

[40] Fabrice Gibert-Darras, Elisabeth Delozanne, Françoise Le Calvez, Agathe Merceron, Jean-Marc Labat, and Fabrice Vandebrouck. Towards a design pattern language to track students' problem-solving abilities. In *12th Int. Conf. AI Edu. (AIED 05) Workshop 1: Usage analysis in learning systems*, page 33, 2005. (cited on page 79).

[41] Jos Groenewegen, Stijn Hoppenbrouwers, and Erik Proper. Playing archimate models. In *Enterprise, Business-Process and Information Systems Modeling*, volume 50 of *Lecture Notes in Business Information Processing*, pages 182–194. Springer Berlin Heidelberg, 2010. (cited on page 43).

[42] Agneta Gulz and Magnus Haake. Design of animated pedagogical agents - A look at their look. *International Journal of Human Computer Studies*, 64(4):322–339, 2006. (cited on page 172).

[43] Magnus Haake and Agneta Gulz. A look at the roles of look & roles in embodied pedagogical agents - A user preference perspectivenet. *International Journal of Artificial Intelligence in Education*, 19(1):39–71, 2009. (cited on page 172).

[44] Patricia L. Hardré. Instructional design as a professional development tool-of-choice for graduate teaching assistants. *Innovative Higher Education*, 30(3):163–175, 2005. (cited on page 128).

[45] Patricia L. Hardré and Alicia O. Burris. What contributes to teaching assistant development: Differential responses to key design features. *Instructional Science*, 40(1):93–118, 2012. (cited on page 128).

[46] R Harrison, S J Counsell, and R V Nithi. An investigation into the applicability and validity of object-oriented design metrics. *Empirical Software Engineering*, 3(3):255–273, 1998. (cited on pages 60 and 167).

[47] Robert W Hasker. UMLGrader: an automated class diagram grader. *Journal of Computing Sciences in Colleges*, pages 47–54, 2011. (cited on page 153).

[48] Robert W Hasker and Mike Rowe. UMLint: Identifying defects in uml diagrams. In *American Society for Engineering Education*. American Society for Engineering Education, 2011. (cited on page 153).

[49] J. Hattie and H. Timperley. The Power of Feedback. *Review of Educational Research*, 77(1):81–112, mar 2007. (cited on pages 59, 146, 171, 176 and 181).

[50] Orit Hazzan and Jeff Kramer. Assessing abstraction skills. *Communications of the ACM*, 59(12):43–45, 2016. (cited on pages 12 and 108).

[51] Peter B. Henderson. Mathematical reasoning in software engineering education. *Commun. ACM*, 46(9):45–50, September 2003. (cited on page 27).

[52] Peter B. Henderson. Math counts: Mathematical reasoning in computing education. *ACM Inroads*, 1(3):22–23, September 2011. (cited on page 27).

[53] Peter B. Henderson. Mathematical reasoning in computing education ii. *ACM Inroads*, 2(1):23–24, February 2011. (cited on page 27).

[54] Alan Hevner and Samir Chatterjee. *Design Science Research in Information Systems*, pages 9–22. Springer US, Boston, MA, 2010. (cited on page 173).

[55] Shuhei Hiya, Kenji Hisazumi, Akira Fukuda, and Tsuneo Nakanishi. clooca: Web based tool for domain specific modeling. In *Demos/Posters/StudentResearch@ MoDELS*, pages 31–35, 2013. (cited on page 67).

[56] Ting-Chia Hsu, Shao-Chen Chang, and Yu-Ting Hung. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126:296–310, 2018. (cited on page 12).

[57] Jeff Irvine. A framework for comparing theories related to motivation in education. *Research in Higher Education Journal*, 35, 2018. (cited on page 10).

[58] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993. (cited on page 4).

[59] W Lewis Johnson and Jeff W Rickel. Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *International Journal of Artificial Intelligence in Education*, 11:47–78, 2000. (cited on page 66).

[60] Rodi Jolak, Boban Vesin, Marcus Isaksson, and Michel R.V. Chaudron. Towards a new generation of software design environments: Supporting the use of informal and formal notations with octouml. In *Second International Workshop on Human Factors in Modeling (HuFaMo 2016). CEUR-WS*, pages 3–10, 2016. (cited on page 67).

[61] Goda Jusaite, Pim Sanders, Damani Lawson, Koen van Polanen, Hani Al-Ers, and Dave R. Stikkolorum. Improving the quality of online collaborative learning for software engineering students. In *International Academic Conference on Teaching, Learning and E-learning*, pages 8–15, 2020. No citations.

[62] Hermann Kaindl. Difficulties in the transition from OO analysis to design. *IEEE Software*, 16(5):94–102, 1999. (cited on pages 12 and 190).

[63] Slava Kalyuga. Enhancing instructional efficiency of interactive e-learning environments: A cognitive load perspective. *Educational Psychology Review*, 19(3):387–399, 2007. (cited on page 171).

[64] Bilal Karasneh, Dave R. Stikkolorum, Enrique Larios, and Michel R.V. Chaudron. Quality assessment of UML class diagrams. In *Proc. Educators' Symp at MoDELS*, 2015. No citations.

[65] Barbara a. Kitchenham, Shari L Pfleeger, Lesley M Pickard, Peter W Jones, David C Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002. (cited on page 14).

[66] Andrew J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, 32(12):971–987, 2006. (cited on page 79).

[67] David A Kolb. *Experiential learning: Experience as the source of learning and development*. FT Press, 2014. (cited on pages 78 and 92).

[68] Michael Kölling. Using bluej to introduce programming. In *Reflections on the Teaching of Programming*, pages 98–115. Springer, 2008. (cited on page 44).

[69] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, April 2007. (cited on pages 4, 12, 26, 40, 92, 108 and 166).

[70] David R. Krathwohl. A revision of bloom's taxonomy: An overview. *Theory Into Practice*, 41(4):212–218, 2002. (cited on pages 10 and 44).

[71] A. Krivitsky. A multi-team, full-cycle, product-oriented scrum simulation with lego bricks – the small & medium business edition v2.0., jun October 2011. (cited on page 190).

[72] Han Lai and Wenjuan Xin. An experimental research of the pair programming in java programming course. *Proceeding of the International Conference on eEducation Entertainment and eManagement*, pages 257–260, 2011. (cited on pages 108, 110 and 117).

[73] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Pearson Education India, 2012. (cited on page 7).

[74] Michael J. Lee and Andrew J. Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research - ICER '11*, page 109, New York, New York, USA, 2011. ACM Press. (cited on page 43).

[75] Michael J. Lee and Andrew J. Ko. A demonstration of gidget, a debugging game for computing education. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pages 211–212, 2014. (cited on pages 43 and 59).

[76] Lego4scrum. `https://www.lego4scrum.com`. [Online; accessed 20-March-2016]. (cited on pages 189, 190 and 219).

[77] T.C. Lethbridge. What knowledge is important to a software professional? *Computer*, 33(5):44–50, 2000. (cited on page 27).

[78] Timothy C Lethbridge. Teaching modeling using Umple: Principles for the development of an effective tool. *Software Engineering Education and Training (CSEE&T), 2014 IEEE 27th Conference on*, pages 23–28, 2014. (cited on page 67).

[79] F. Leung and N. Bolloju. Analyzing the Quality of Domain Models Developed by Novice Systems Analysts. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 00(C):188b–188b, 2005. (cited on pages 108, 110 and 127).

[80] Felix Leung and Narasimha Bolloju. Analyzing the quality of domain models developed by novice systems analysts. In *System Sciences, 2005. HICSS'05. Proc. of the 38th Annual Hawaii Int. Conf. on*, pages 188b–188b. IEEE, 2005. (cited on pages 40, 92, 93 and 166).

[81] C. Lewis. Using the "thinking-aloud" method in cognitive interface design. Technical report, IBM T.J. Watson Research Center, 1982. (cited on pages 15 and 45).

[82] Clayton Lewis and John Rieman. Task-centered user interface design. *A Practical Introductio*, 1993. (cited on page 94).

[83] Grischa Liebel, Rogardt Heldal, and Jan Philipp Steghofer. Impact of the use of industrial modelling tools on modelling education. *Proceedings - 2016 IEEE 29th Conference on Software Engineering Education and Training, CSEEandT 2016*, pages 18–27, 2016. (cited on pages 8, 118 and 145).

[84] Zhiyi Ma, Chih-yi Yeh, Huihong He, and Hongjie Chen. A Web Based UML Modeling Tool with Touch Screens. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 835–838, 2014. (cited on page 67).

[85] Ntima Mabanza and Lizette de Wet. Determining the usability effect of pedagogical interface agents on adult computer literacy training. In *E-Learning Paradigms and Applications*, pages 145–183. Springer, 2014. (cited on page 71).

[86] Philip Machanick. A social construction approach to computer science education. *Computer Science Education*, 17(1):1–20, 2007. (cited on page 11).

[87] Maíra R Marques, Alcides Quispe, and Sergio F Ochoa. A systematic mapping study on practical approaches to teaching software engineering. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–8. IEEE, 2014. (cited on pages 11 and 40).

[88] Robert C Martin. Design Principles and Design Patterns. *Object Mentor*, pages 1–34, 2000. (cited on pages 7, 29 and 44).

[89] Ian McChesney. Three Years of Student Pair Programming – Action Research Insights and Outcomes. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE '16)*, pages 84–89, 2016. (cited on pages 108, 110 and 117).

[90] Sabine Moisan and Jean Paul Rigault. Teaching object-oriented modeling and UML to various audiences. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6002 LNCS:40–54, 2010. (cited on page 108).

[91] Anders I. Mørch, Silje Jondahl, and Jan A. Dolonen. Supporting conceptual awareness with pedagogical agents. *Information Systems Frontiers*, 7(1):39–53, 2005. (cited on page 172).

[92] Robert Moser. A fantasy adventure game as a learning environment. why learning to program is so difficult and what can be done about it. *ITiCSE 97 Proceedings of the 2nd conference on Integrating technology into computer science education*, 29(3):114–116, 1997. (cited on page 52).

[93] Madlen Müller-Wuttke and Nicholas H Müller. Cognitive load levels while learning with or without a pedagogical agent. In *International Conference on Human-Computer Interaction*, pages 266–276. Springer, 2019. (cited on page 173).

[94] Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty HC Cheng, Philippe Collet, Benoit Combemale, Robert B France, Rogardt Heldal, James Hill, et al. The relevance of model-driven engineering thirty years from now. In *Model-Driven Engineering Languages and Systems*, pages 183–200. Springer, 2014. (cited on pages 3, 13 and 188).

[95] Valbona Muzaka. The niche of graduate teaching assistants (GTAs): Perceptions and reflections. *Teaching in Higher Education*, 14(1):1–12, 2009. (cited on page 126).

[96] Sharon Nelson-Le Gall. Help-seeking: An understudied problem-solving skill in children. *Developmental review*, 1(3):224–246, 1981. (cited on page 170).

[97] Hafeez Osman, Arjan van Zadelhoff, Dave R. Stikkolorum, and Michel R.V. Chaudron. UML class diagram simplification: what is in the developer's mind? In *Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling*, page 5. ACM, 2012. No citations.

[98] Maria Paasivaara, Ville Heikkilä, Casper Lassenius, and Towo Toivola. Teaching students scrum using lego blocks. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 382–391. ACM, 2014. (cited on page 190).

[99] Chris Park. The graduate teaching assistant (GTA): lessons from North American experience. *Teaching in Higher Education*, 9(3):349–361, 2004. (cited on page 127).

[100] Óscar Pastor, Sergio España, and Jose Ignacio Panach. Learning Pros and Cons of Model-Driven Development in a Practical Teaching Experience. In *International Conference on Conceptual Modeling*, pages 218–227. Springer, 2016. (cited on page 110).

[101] Elizabeth Ann Patitsas. A case study of the development of CS teaching assistants and their experiences with team teaching. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research - Koli Calling '13*, pages 115–124, 2013. (cited on page 128).

[102] Dewayne E Perry, Adam a Porter, and Lawrence G Votta. Empirical Studies of Software Engineering : A Roadmap. *Proceedings of the conference on The future of Software engineering ICSE 00*, pages 345–355, 2000. (cited on page 14).

[103] Jakob Pinggera, Pnina Soffer, Stefan Zugal, Barbara Weber, Matthias Weidlich, Dirk Fahland, Hajo a. Reijers, and Jan Mendling. Modeling styles in business process modeling. *Lecture Notes in Business Information Processing*, 113 LNBIP:151–166, 2012. (cited on page 79).

[104] M Prensky. *Digital game-based learning*. McGraw-Hill & Paragon House, New York, 2001. (cited on page 42).

[105] Marc Prensky. Digital Game-Based Learning. *Computers in Entertainment (CIE)*, 1(1):1–4, 2003. (cited on page 42).

[106] Andrew K Przybylski, C Scott Rigby, and Richard M Ryan. A motivational model of video game engagement. *Review of General Psychology*, 14(2):154–166, 2010. (cited on pages 52, 59 and 218).

[107] Truong Ho Quang. LogViz - visual log analyzer. `https://gitlab.com/truonghoquang/LogVisualizer`, 2015. [Online; accessed 8-March-2015]. (cited on page 71).

[108] Ebrahim Rahimi and Dave R. Stikkolorum, editors. *CSERC '19: Proceedings of the 8th Computer Science Education Research Conference*, New York, NY, USA, 2019. Association for Computing Machinery. No citations.

[109] Ervin Ramollari and Dimitris Dranidis. StudentUML: An educational tool supporting object-oriented analysis and design. *... of the 11th Panhellenic Conference on ...*, pages 363–373, 2007. (cited on page 66).

[110] John Raven. The raven's progressive matrices: change and stability over culture and time. *Cognitive psychology*, 41(1):1–48, 2000. (cited on page 31).

[111] Maryam Razavian, Antony Tang, Rafael Capilla, and Patricia Lago. In two minds: how reflections influence software design thinking. *Journal of Software: Evolution and Process*, 28(6):394–426, 2016. (cited on page 110).

[112] Arthur J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1996. (cited on page 29).

[113] Patricia Roberts. Abstract thinking: a predictor of modelling ability? In *Educators Symposium of the ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems*, pages 753–754. Springer, 2009. (cited on pages 27, 40, 92 and 166).

[114] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172, 2003. (cited on page 44).

[115] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William E. Lorensen, et al. *Object-oriented modeling and design*, volume 199. Prentice-hall Englewood Cliffs, NJ, 1991. (cited on page 4).

[116] Bernhard Rumpe. Agile Modeling with the UML 1 Portfolio of Software Engineering Techniques. *Radical Innovations of Software and Systems Engineering in the Future. 9th International Workshop*, pages 297–309, 2004. (cited on pages 110 and 190).

[117] Bernhard Rumpe. *Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring*. Springer-Verlag, 2012. (cited on page 190).

[118] Adrian Rusu, Robert Russell, Remo Cocco, and Spence DiNicolantonio. Introducing object oriented design patterns through a puzzle-based serious computer game. In *2011 Frontiers in Education Conference (FIE)*, pages F1H–1. IEEE, 2011. (cited on page 43).

[119] Andreas Schroeder, Annabelle Klarl, Philip Mayer, and Christian Kroiss. Teaching agile software development through lab courses. *IEEE Global Engineering Education Conference, EDUCON*, pages 1177–1186, 2012. (cited on page 190).

[120] Noah L Schroeder. The influence of a pedagogical agent on learners' cognitive load. *Journal of Educational Technology & Society*, 20(4):138–147, 2017. (cited on pages 172 and 173).

[121] Swapneel Sheth, Jonathan Bell, and Gail Kaiser. A competitive-collaborative approach for introducing software engineering in a cs2 class. In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 41–50. IEEE, 2013. (cited on page 42).

[122] Shin-shing Shin. A Study on the Difficulties of Learning Phase Transition in Object-Oriented Analysis and Design From the Viewpoint of Semantic Distance. *IEEE*, 58(2):1–7, 2014. (cited on pages 12, 108, 121 and 190).

[123] Ricardo Azambuja Silveira, Eduardo Rodrigues Gomes, and Rosa Maria Viccari. Intelligent learning objects: An agent approach to create reusable intelligent learning environments with learning objects. *Advances in Artificial Intelligence - Iberamia-Sbia 2006, Proceedings*, pages pp. 17–26, 2006. (cited on page 71).

[124] J. Soler, I. Boada, F. Prados, J. Poch, and R. Fabregat. A web-based e-learning tool for UML class diagrams. *2010 IEEE Education Engineering Conference, EDUCON 2010*, pages 973–979, 2010. (cited on page 67).

[125] Minseok Song and W.M.P. van der Aalst. Supporting process mining by showing events at a glance. *In Proceedings of 17th Annual Workshop on Information Technologies and Systems*, pages 139–147, 2007. (cited on page 93).

[126] Mauricio Ronny De Almeida Souza, Lucas Furtini Veado, Renata Teles Moreira, Eduardo Magno Lages Figueiredo, and Heitor Augustus Xavier Costa. Games for learning: bridging game-related education methods to software engineering knowledge areas. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, pages 170–179. IEEE, 2017. (cited on pages 42 and 44).

[127] Shashank Srikant and Varun Aggarwal. A system to grade computer programming skills using machine learning. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, pages 1887–1896, 2014. (cited on page 153).

[128] Thomas Staubitz, Hauke Klement, Jan Renz, Ralf Teusner, and Christoph Meinel. Towards practical programming exercises and automated assessment in massive open online courses. In *Teaching, Assessment, and Learning for Engineering (TALE), 2015 IEEE International Conference on*, pages 23–30. IEEE, 2015. (cited on page 44).

[129] Christoph Johann Stettina and Werner Heijstek. Necessary and neglected?: An empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM International Conference on Design of Communication*, SIGDOC '11, pages 159–166, New York, NY, USA, 2011. ACM. (cited on page 188).

[130] Dave R. Stikkolorum. Online Experiment with WebUML : recording difficulties and strategies of students. Technical report, Leiden University, LIACS, 07 2015. (cited on page 94).

[131] Dave R. Stikkolorum and Michel R.V. Chaudron. A workshop for integrating UML modelling and agile development in the classroom. In *Proceedings of the*

Computer Science Education Research Conference 2016, pages 4–11. ACM, 2016. (cited on pages 109, 110 and 116).

[132] 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. (cited on pages 127, 145 and 147).

[133] Dave R. Stikkolorum, Michel R.V. Chaudron, and Oswald de Bruin. The art of software design, a video game for learning software design principles. In *Gamification Contest MODELS'12 Innsbruck*, 2012. (cited on page 34).

[134] Dave R. Stikkolorum, F. Gomes de Oliveira Neto, and Michel R.V. Chaudron. Evaluating didactic approaches used by teaching assistants for software analysis and design using uml. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 122–131, New York, NY, USA, 2018. ACM. No citations.

[135] Dave R. Stikkolorum, Birgit Demuth, Vadim Zaytsev, Frédéric Boulanger, and Jeff Gray. The MOOC hype: Can we ignore it? In *Reflections on the current use of massive open online courses in software modeling education. MODELS Educators Symposium, EduSymp*, 2014. No citations.

[136] Dave R. Stikkolorum, Truong Ho-Quang, and Michel R.V. Chaudron. Revealing students' UML class diagram modelling strategies with WebUML and LogViz. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*, pages 275–279. IEEE, 2015. (cited on pages 71, 97, 153, 155 and 189).

[137] Dave R. Stikkolorum, Truong Ho-Quang, Bilal Karashneh, and Michel R.V. Chaudron. Uncovering students' common difficulties and strategies during a class diagram design process: an online experiment. In *Educators Symposium 2015, co-located with the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, 2015. (cited on pages 71, 108, 109, 174 and 189).

[138] Dave R. Stikkolorum and Ebrahim Rahimi, editors. *CSERC '20: Proceedings of the 9th Computer Science Education Research Conference*, New York, NY, USA, 2020. Association for Computing Machinery. No citations.

[139] Dave R. Stikkolorum and Ebrahim Rahimi, editors. *CSERC '21: Proceedings of the 10th Computer Science Education Research Conference*, New York, NY, USA, 2021. Association for Computing Machinery. No citations.

[140] Dave R. Stikkolorum and Ebrahim Rahimi, editors. *CSERC '22: Proceedings of the 11th Computer Science Education Research Conference*, New York, NY, USA, 2022. Association for Computing Machinery. No citations.

[141] Dave R. Stikkolorum, Claire Stevenson, and Michel R.V. Chaudron. Assessing software design skills and their relation with reasoning skills. In *EduSymp 2013. CEUR, vol. 1134, paper 5*, 2013. (cited on pages 40, 92, 108 and 166).

[142] Dave R. Stikkolorum, Claire E. Stevenson, and Michel R.V. Chaudron. Technical report 2013-02. `https://research.drstikko.nl/files/technical_report_2013-02.pdf`, 2013. (cited on page 31).

[143] Dave R. Stikkolorum, Peter van der Putten, Caroline Sperandio, and Michel R.V. Chaudron. Towards automated grading of uml class diagrams with machine learning. In *BNAIC/BENELEARN*, 2019. No citations.

[144] Harald Störrle. On the impact of layout quality to understanding UML diagrams: Diagram type and expertise. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pages 49–56, 2012. (cited on page 94).

[145] Harald Störrle. On the impact of layout quality to understanding UML diagrams: Size matters. In *Model-Driven Engineering Languages and Systems*, pages 518–534. Springer, 2014. (cited on page 94).

[146] Michael Striewe and Michael Goedicke. Automated checks on uml diagrams. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 38–42, New York, NY, USA, 2011. ACM. (cited on page 153).

[147] Michael Striewe and Michael Goedicke. Automated assessment of uml activity diagrams. In *ITiCSE*, page 336, 2014. (cited on page 153).

[148] Antony Tang. Software Designers , Are You Biased ? *Program*, pages 1–8, 2011. (cited on page 110).

[149] Jennifer Tenzer and Perdita Stevens. GUIDE: Games with UML for interactive design exploration. *Knowledge-Based Systems*, 20(7):652–670, 2007. (cited on page 44).

[150] Damiano Torre, Yvan Labiche, Marcela Genero, and Maged Elaasar. A systematic identification of consistency rules for uml diagrams. *Journal of Systems and Software*, 144:121–142, 2018. (cited on page 167).

[151] Kalliopi Tourtoglou and Maria Virvou. User modelling in a collaborative learning environment for uml. In *Fifth International Conference on Information Technology: New Generations (itng 2008)*, pages 1257–1258. IEEE, 2008. (cited on page 169).

[152] Ian Utting, Stephen Cooper, Michael Kölling, John Maloney, and Mitchel Resnick. Alice, greenfoot, and scratch–a discussion. *ACM Transactions on Computing Education (TOCE)*, 10(4):17, 2010. (cited on page 43).

[153] Marcella Veldthuis, Matthijs Koning, and Dave R. Stikkolorum. A quest to engage computer science students:using dungeons & dragons for developing soft skills. In *Proceedings of the Computer Science Education Research Conference 2021*. ACM, 2021. No citations.

[154] JDHM Vermunt. *Leerstijlen en sturen van leerprocessen in het hoger onderwijs - Naar procesgerichte instructie in zelfstandig denken*. Amsterdam: Swets & Zeitlinger, 1992. (cited on pages 78 and 92).

[155] Boban Vesin, Aleksandra Klašnja-Milićević, Katerina Mangaroska, Mirjana Ivanović, Rodi Jolak, Dave Stikkolorum, and Michel Chaudron. Web-based educational ecosystem for automatization of teaching process and assessment of students. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, pages 1–9, 2018. (cited on page 223).

[156] Arto Vihavainen, Matti Luukkainen, and Jaakko Kurhila. Multi-faceted support for mooc in programming. In *Proceedings of the 13th annual conference on Information technology education*, pages 171–176. ACM, 2012. (cited on page 44).

[157] Visual paradigm. `https://www.visual-paradigm.com`. [Online; accessed 20-November-2022]. (cited on page 192).

[158] Christiane Gresse von Wangenheim, Rafael Savi, and Adriano Ferreti Borgatto. Scrumia—an educational game for teaching scrum in computing courses. *Journal of Systems and Software*, 86(10):2675–2687, 2013. (cited on page 191).

[159] Lorin W Anderson, David R Krathwohl, Peter W Airasian, Kathleen A Cruikshank, Richard E Mayer, Paul R Pintrich, James Raths, and Merlin C Wittrock. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged Edition*. Allyn & Bacon, 2000. (cited on page 30).

[160] Wim Westera, Rob Nadolski, and Hans Hummel. Serious Gaming Analytics : What Students Log Files Tell Us about Gaming and Learning. *International Journal of Serious Games*, 1(2):35–50, 2014. (cited on page 79).

[161] Grant P Wiggins, Grant Wiggins, and Jay McTighe. *Understanding by design*. Ascd, 2005. (cited on page 10).

[162] Yuefeng Zhang. Agile Development in Practice. *IEEE Software - IEEE Computer Society*, 2011. (cited on pages 110 and 190).

[163] Chen Zhuoyi, Li Na, and Zhang Hongjie. Exploration of teaching model of the database course based on constructivism learning theory. In *Consumer Electronics, Comm. and Networks (CECNet), 2nd Int. Conf. on*, pages 1808–1811. IEEE, 2012. (cited on page 110).

Acknowledgements

Over the past years many people and organisations supported my research. I would like to take the opportunity to thank them. Without their support I could not have finished this journey.

To the following organisations: Leiden University (LIACS and Dual PhD Centre), The Hague University of Applied Sciences. Thank you for the opportunity and for trusting me to complete this research.

To the following collaborators: I would like to thank the students and lecturers from Chalmers University of Technology and Göteborg University in Sweden and Utrecht University in The Netherlands for their participation in the study of Chapter 2. I would like to thank the lecturers and students of the Software Engineering program of Makerere University in Uganda for their collaboration in the experiment of Chapter 6. I would like to thank the students and lecturers from The Hague University of Applied Sciences in The Netherlands for their participation and helpful discussions in the research of Chapter 7. I would like to thank the students of Chalmers University of Technology and Göteborg University, for evaluating the feedback agent of Chapter 9. I would like to thank all the students from Chalmers University of Technology and Göteborg University that participated in this experiment for their enthusiasm and constructive contribution to Chapter 11. I would like to thank the following students (they were students at the time of conducting the research): Claire Stevenson for collaborating with me in the research of Chapter 2, Oswald de Bruin for the collaboration in Chapter 3, Caroline Sperandio for the collaboration in Chapter 9 and Helen Anckar for the collaboration in Chapter 10.

To my critical friends: Grischa Liebel, Jan-Philipp Steghöfer, Marcella Veldthuis, Felienne Hermans, Tony Andrioli, Tim Cocx and Vreda Pieterse. Thank you for your constructive critics on my research approaches and the writing of this dissertation.

To my colleagues from LIACS: Werner Heijstek, Ana Fernandez, Hafeez Osman, Bilal Karasneh, Christoph Stettina, Ali Mirsoleimani, Ben Ruijl, Ariadi Nugroho and Bob Zadok. Thank you for all the nice talks and discussions, good advises and sharing of insights of your own research.

To my colleagues from Chalmers University of Technology and Göteborg University: Håkan Burden, Rogardt Heldal, Truong Ho Quang, Rodi Jolak, Pariya Kashfi, Regina Hebig, Hiva Alahyari, Boban Vesin and Francisco Gomes. Thank you for sharing your insights, the treats during *fika* and the nice after work moments.

To my colleagues from The Hague University: I would like to thank you all (the group is too large to name everyone individually) for supporting me all these years in combining my teaching and managing role with my PhD research.

To my family: I would like to thank and express my love to all my family members and close friends, but in particular my wife *Maria José* and my kids *Jay* and *Julian* for their belief and patience. I could not have done this without you.

Summary

Software design is a challenging task for students. Due to the trend of increasingly complex systems, teaching students software design becomes also challenging. It is known that students struggle with the (abstract) task of software design, therefore new teaching approaches are necessary.

This dissertation explores different ways of improving and extending our current approaches in software design education. The main objective of this research is:

Main Objective *How can we improve the ways of teaching software design?*

We studied different aspects of the main objective: measuring software design skills; guidance and feedback on design tasks; and, student motivation and engagement in software design activities.

We explored interventions through the use of tools we developed for assessing design skills, modelling, guiding and grading. We centered our approach around design principles. Design principles form the foundations of the complex balancing of multiple conflicting objectives that good software designers should eventually master.

Next, we describe our different interventions and developed tools.

We designed an on-line multiple-choice test as an instrument for measuring software design skills and abstract reasoning skills of students. Our research revealed the relationship between abstract reasoning and the ability of solving software design problems. We see opportunities in exploring educational interventions for specific (abstract) reasoning tasks and/or alternative teaching methods.

Lecturers can use our test to diagnose students and choose the appropriate interven-

tions.

For motivating students for learning software design and for keeping them engaged, we explored an educational puzzle game.

By practising making design decisions, the players learn about balancing between the different software design objectives. An important instrument in the game is the visual feedback system that guides students towards their solution. We see the game as an opportunity to support students' self-assessment possibilities by using the built-in feedback mechanism. The interactive game keeps them engaged.

Because of the need for a UML tool that could be used as an educational tool as well as a research tool we developed WebUML. The web-based architecture combined with a logging system enabled larger scale classroom studies that could be analysed with statistical analyses. We were able to collect data about student activities during class diagram analysis and design. Our tools open up new ways of studying the process of software design. Having the possibility to analyse students' modelling behaviour enables us to develop better educational programs and tools. Recognising certain strategies of making designs can help us to give better feedback during a design-task rather than after completion of the task.

By analysing the logs of students using the WebUML tool when making designs, we found typical strategies students use to make class designs: Depth Less, Depth First, Breadth First, Ad Hoc. The strategies differ in whether a student first focuses on high-level class structure or focuses on individual class detail.

As another teaching-intervention, we analysed peer-reflections of students during an analysis and design assignment. The peer-reflections revealed a collection of challenges the students face. We identified common problems and difficulties and presented 10 concrete points of concern for teaching software design. 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 that they had as well as the progress and quality aspects of their models. With the presented points of concern we expand the knowledge about common difficulties in students' learning of software design.

For another approach of distilling the struggles that students face while making a software design we observed the interaction between teaching assistants (TAs) and students during an analysis and design course. We were able to distil and categorise the common challenges students have recorded where TAs supported them. Further, we have an overview of the typical interventions TAs use when helping students during their assignment tasks.

We explored the application of automated feedback and grading of UML class diagrams in order to enable students to practice more often and reflect on their own progress by self-assessment. In addition to the evaluation function of such mechanisms the feedback enables the guidance of students towards their solution. The feedback is about: using the right semantics, such as the appropriate translation of a relationship between concepts and satisfying the requirements of the assignment, such as identifying the relevant concepts within a given context.

We collected the data of submitted software design assignments of bachelor students. On this data we applied machine learning for trying to grade assignments. Based on our results we conclude that using machine learning for a classification that uses the same precision as a 10 point grading[2] scale is not accurate. However, the current prediction model does give users a rough indication of the quality of their models.

We presented our pedagogical feedback agent as an extension to WebUML. This agent gives feedback to students while they are learning to make UML class designs.

The students appreciate the guiding role of the feedback agent: the way it confirms steps that were done in the right way and suggestions for next steps.

Next to tooling, we proposed a workshop that integrates UML modelling into agile development. An integrated software development approach helps students to understand the role of (UML) modelling in a software development cycle. In addition we are able to collect information about students' difficulties that arise during discussing the modelling task in a group. Students have a positive judgement towards the workshop. The students judge that they have improved their modelling skills and that the workshop gave insight into the relationship between modelling and the software development process.

With the above-mentioned studies we expanded the repertoire of educational interventions and tools for learning software design. With proper feedback these interventions and tools guide the students during their design tasks. We invite other lecturers to use and build on our approaches in order to extend our insights and enrich future interactive learning environments.

[2] This is the conventional scale for grading exams in The Netherlands.

Samenvatting

Softwareontwerp is een uitdagende taak voor studenten. Door de toename van steeds complexer wordende systemen wordt ook het aanleren van softwareontwerp een uitdaging. Het is algemeen bekend dat studenten worstelen met het aanleren van de (abstracte) vaardigheden van softwareontwerp, daarom zijn nieuwe onderwijsinterventies nodig.

Dit proefschrift onderzoekt verschillende manieren waarop we onze huidige aanpak in het onderwijzen van softwareontwerp kunnen verbeteren en uitbreiden. Het hoofddoel van dit onderzoek is:

Hoofddoel *Hoe kunnen we de manier waarop we softwareontwerp aanleren verbeteren?*

We hebben verschillende aspecten van het hoofddoel bestudeerd: het meten van softwareontwerpvaardigheden; de begeleiding en de feedback op ontwerptaken; en de motivatie van studenten voor - en de betrokkenheid bij softwareontwerpactiviteiten.

We onderzochten interventies door gebruik te maken van tools die we ontwikkeld hebben voor het assessen van ontwerpvaardigheden, modelleren van software en het begeleiden en beoordelen van het ontwerpproces. Centraal in ons onderzoek stonden softwareontwerpprincipes. Deze ontwerpprincipes vormen de basis van de complexe afweging van meerdere tegenstrijdige doelen die goede softwareontwerpers uiteindelijk zouden moeten kunnen maken.

Hieronder beschrijven we onze verschillende interventies en ontwikkelde tools.

We hebben een online meerkeuzetoets ontworpen, een instrument voor het meten van softwareontwerpvaardigheden en het vermogen tot abstract redeneren van studenten. Ons onderzoek toonde de relatie aan tussen abstract redeneren en het vermogen om softwareontwerpproblemen op te lossen. Wij zien hiermee kansen voor onderzoek naar onderwijsinterventies voor specifieke (abstracte) redeneertaken en/of alternatieve werkvormen. Docenten kunnen onze test gebruiken voor het diagnosticeren van studenten en daarmee passende interventies kiezen.

Om studenten te motiveren voor het leren van softwareontwerp, en om ze betrokken te houden, hebben we onderzoek gedaan naar een educatieve puzzel-videogame. Door in-game te oefenen met het nemen van ontwerpbeslissingen leren de spelers hoe ze de balans kunnen vinden tussen de verschillende doelen van een softwareontwerp. Een belangrijk instrument in de game is het visuele feedbacksysteem dat studenten naar hun oplossing toe begeleidt. Met het ingebouwde feedbackmechanisme zien we de game als mogelijk middel voor zelfevaluatie van studenten. De interactieve game houdt ze betrokken.

Vanwege de behoefte aan een UML-tool die zowel als educatieve tool als onderzoekstool kan worden gebruikt, hebben we WebUML ontwikkeld. De web-based architectuur in combinatie met een logsysteem maakt het mogelijk om onderzoek met statistische analyses te doen bij grote groepen. We hebben data verzameld over de ontwerpactiviteiten van studenten tijdens het analyseren en ontwerpen van class diagrams. Onze tools creëren nieuwe manieren om het softwareontwerpproces te bestuderen. De analyse van het modelleergedrag van studenten helpt ons betere educatieve programma's en tools te ontwikkelen. Het herkennen van bepaalde strategieën van studenten bij het maken van ontwerpen kan ons helpen om betere feedback te geven tijdens een ontwerptaak in plaats van ná uitvoering van hun taak. Door de logs van de studenten die de WebUML-tool gebruiken bij het maken van ontwerpen te analyseren, ontdekten we typische strategieën die studenten gebruiken om class designs te maken: Depth Less, Depth First, Breadth First, en Ad Hoc. De strategieën verschillen in waar de student zich eerst op richt, op de high-level class structuur of op de individuele class details.

Een andere onderwijsinterventie die we onderzochten was het doen van peer-reflecties door studenten tijdens een analyse- en ontwerpopdracht. De peer-reflecties legden de problemen waarmee de studenten worden geconfronteerd bloot. We identificeerden veelvoorkomende problemen en presenteerden 10 concrete aandachtspunten voor het aanleren van softwareontwerp. Uit de vastgelegde peer-reflecties concluderen we dat peer-reflectie studenten helpt om hun moeilijkheden bij het modelleren van softwareontwerpen uit te drukken. De studenten bespraken actief de moeilijkheden die ze ondervonden tijdens hun taak. Ze bespraken zowel de voortgang als de kwaliteitsaspecten van hun modellen. Met de gepresenteerde aandachtspunten breiden we de kennis uit over veelvoorkomende problemen bij het aanleren van softwareontwerp door studenten.

Voor een andere benadering om problemen van studenten tijdens het maken van een softwareontwerp bloot te leggen, observeerden we de interactie tussen onderwijsassistenten (OAs) en studenten tijdens een analyse- en ontwerpcursus. We hebben de gemeenschappelijke problemen waarbij studenten de hulp van OAs hebben gevraagd gecategoriseerd. Verder hebben we een overzicht van de typische interventies die OAs gebruiken bij het ondersteunen van studenten tijdens het uitvoeren van hun opdrachten.

We onderzochten de toepassing van geautomatiseerde feedback en beoordeling van UML class diagrams om studenten in staat te stellen vaker te oefenen en na te denken over hun eigen voortgang door het gebruik van zelfevaluatie. Naast de evaluatiefunctie, maakt de feedback het mogelijk studenten te begeleiden naar hun oplossing. De feedback gaat over: het gebruik van de juiste semantiek, zoals de juiste vertaling van een relatie tussen concepten en het voldoen aan requirements, zoals het identificeren van relevante concepten binnen een gegeven context.

We hebben de data van softwareontwerpen van bachelorstudenten verzameld. Op deze data hebben we machine learning toegepast met als doel deze opdrachten automatisch te kunnen beoordelen. Op basis van onze resultaten concluderen we dat het gebruik van machine learning voor een classificatie die dezelfde precisie gebruikt als een 10-punts beoordelingsschaal[3] niet nauwkeurig is. Het huidige voorspellingsmodel geeft gebruikers echter wel een ruwe indicatie van de kwaliteit van hun modellen.

We hebben onze pedagogische feedbackagent gepresenteerd als uitbreiding op WebUML. Deze agent geeft feedback aan studenten terwijl ze leren om UML class designs te maken. De studenten waarderen de begeleidingsrol van de agent: de manier waarop wordt bevestigd dat stappen op de juiste manier zijn uitgevoerd en de suggesties die worden gegeven voor vervolgstappen richting de oplossing.

Naast onze tooling hebben we een workshop ontwikkeld waarin UML-modelleren integreert is in het agile softwareontwikkelingsproces. Een geïntegreerde benadering van softwareontwikkeling helpt studenten de rol van (UML)-modelleren in een softwareontwikkelingscyclus te begrijpen. Daarnaast hebben we tijdens groepsgesprekken over de modelleertaak opvallende problemen van studenten verzameld. Studenten zijn positief over de workshop. De studenten zijn van mening dat ze hun modelleervaardigheden hebben verbeterd en dat de workshop inzicht heeft gegeven in de relatie tussen modelleren en het softwareontwikkelingsproces.

Met de bovengenoemde onderzoeken hebben we het repertoire aan onderwijsinterventies en tools voor het aanleren van softwareontwerp uitgebreid. Met de juiste feedback begeleiden deze interventies en tools de studenten tijdens hun ontwerptaken. We

[3] Dit is de conventionele schaal voor het beoordelen van toetsen in Nederland.

nodigen andere docenten uit om onze aanpak te gebruiken en voort te bouwen op onze inzichten en hiermee toekomstige interactieve leeromgevingen te verrijken.

About the Author

Dave Stikkolorum was born (1976) and raised in the The Hague-region of The Netherlands. After high school he studied electrical engineering with 'technical computer science' as major. After his studies, in 2001, he decided to start as a part-time software developer and as a secondary school teacher in mathematics. He traded the job of software developer for a job as teacher at The Hague University of Applied Sciences (HHS). From 2005 Dave became a full-time member of the 'technical informatics' team at HHS. Because of an interest in the education of software design he started his PhD research at the Leiden Institute of Advanced Computer Science (LIACS, Leiden University). The research was conducted under the supervision of Michel Chaudron and later accompanied by Peter van der Putten. Currently Dave is involved as program director of HBO-ICT at The Hague University of Applied Sciences and teaches in the Game Development and Simulation program.

Appendices

Appendix A

Overview Recommendations for Teaching Software Modelling

Table A.1: *Overview Recommendations for Teaching Software Modelling*

Student Challenges	(Education of) Theory	Tool	Skills		Process Skills
			Abstraction	Bloom levels	
Analysis	**Problem 1**: Students have difficulties with the analysis of text based assignments. Because of incomplete instructions students tend to loose the overall overview. **Recommendation 1**: keep instruction text simple when students start to learn and practice software analysis. **Recommendation 2**: When it is need for practice to have incomplete or poor assignment texts, provide enough feedback moments and/or peer review sessions to overcome students wandering.	**Problem 2**: because UML uses the same notation for Analysis models as well as design models, most tools do not support analysis steps by having a special analysis mode. **Recommendation 3**: emphasise the analysis phase in which a student is modelling and use a simple feature set for making domain models. In best case a teacher can tailor the tool that is used.	**Problem 3**: Students have difficulties to identify the relevant concepts from a problem (domain) in order to abstract them and present them in a (UML) diagram. **Recommendation 4**: use the noun identification technique to train the identification of relevant concepts.	**Problem 4**: Software analysis reaches the higher level bloom tasks: analyse (4) evaluate (5) and create (6). **Recommendation 5**: Students should practice with exercises that are not too complex. Using a domain that students are familiar with can be helpful.	**Problem 5**: Students have difficulties in reflecting to their own analysis models. **Recommendation 6**: students should be trained using an approach where analysis models iteratively evolve. **Recommendation 7**: organise process support were continuously feedback articulated by the lecturer and/or teaching assistants.

Design	**Problem 6**: Students have difficulties with determining how detailed a design model should be. **Recommendation 8**: Offer assignment that make clear which level of detail is expected from the student. **Problem 7**: Because of the open character of most assignments students have difficulties in deciding of their design are "done". **Recommendation 9**: Include feedback moments and design discussions during a lecture. **Recommendation 10**: Offer follow-up assignments that enable students to discover missing elements in their initial design.	**Problem 8**: because UML uses the same notation for analysis models as well as design models, most tools do not support design steps by having a special design mode. **Recommendation 11**: emphasise the design phase in which a student is modelling and use an increasing complex feature set for making domain models. In best case a teacher can tailor the tool that is used.	**Problem 9**: Students have difficulties to include relevant abstractions of the domain concepts (presented in a domain model) in their software design. **Problem 10**: Students have difficulties deciding between classes and attributes (abstraction). *To be explored further.*	idem*	**Problem 11**: Students have difficulties in reflecting to their own design models. **Recommendation 12**: students should be trained using an approach where designs iteratively evolve. **Recommendation 13**: organise process support were continuously feedback articulated by the lecturer and/or teaching assistants.

Analysis vs Design, Analysis and Design		**Problem 12**: there is no mature modelling tool that supports the transition from analysis to design **Recommendation 14**: develop tools that supports the transition from analysis to design **Recommendation 15**: integrate modelling tools with software that supports (agile) software development process tools.	**Problem 13**: Going from analysis to design covers going from a concrete problem domain to an abstraction followed by creating a design from 1) the gathered domain concept abstractions 2) the introduced software concepts from the solution domain. This combining of abstractions of two sources is difficult to grasp for students. **Recommendation 16**: Offer standard abstractions (responsibility driven design, Wirfs-Brock) and standard architectures (layered architecture) as a first start.	idem*	**Problem 14**: Students don't have experience with the development process and therefore find it difficult to see the relation between analysis and design. **Recommendation 17**: students should be trained using an approach where analysis and design are part of iterative cycles (agile UML workshop).

Design decisions	**Problem 15**: students have problems accepting that there are multiple solutions to a problem. **Recommendation 18**: use software design principles as a guide for decision making.	**Problem 16**: there is no mature modelling tool that supports design decision making. Most tools check on syntax level. **Recommendation 19**: develop tools that use feedback based on design principles. **Recommendation 20**: put a low emphasis on syntax.	**Problem 17**: UML offers a very high abstraction for generalisation of (software elements): class. **Recommendation 21**: Make use of stereotypes / roles when designing (responsibility driven design, Wirfs-Brock)	idem*	**Problem 18**: students have problems deciding when a design is better than another. **Recommendation 22**: use software design principles as a guide for decision making. Principles should be used when discussing (pair/collaborated modelling)

Design vs Implementation	**Problem 19**: Students do not see the effect of their design (decisions) on the implementation. There is a difference in time in the development process. **Recommendation 23**: Teach students an agile approach in modelling for having smaller steps in the development process that makes the effect of design decisions clearer.	**Problem 20**: Students don't update their design from the moment they start implementing in code. **Problem 21**: Most tools only offer one way code generation / import or only partial code generation / import. **Recommendation 24**: Develop tools that support complete round-trip code generation - code import (model generation). **Recommendation 25**: Find ways to keep track of code changes and updating students' designs.	**Problem 22**: there is a gap between the completeness of an 'abstract' design and the desired implementation. Students don't feel comfortable with the degree of freedom when they are novice software developers. *To be explored further.*	idem*	**Problem 23**: Students don't update their design from the moment they start implementing in code. **Problem 24**: Students neglect their models when implementing the software. **Recommendation 26**: students should be trained using an approach where design and implementation are part of iterative cycles (agile UML workshop).

Implementation Thinking	**Problem 25**: Because of the direct effect of implementation, students are not motivated to use modelling as a first step during analysis and design. **Recommendation 27**: Use motivating environments such as gaming ("The Art of Software Design") to engage students into modelling.		**Problem 26**: students have difficulties not to think in implementation when making initial designs. It distracts them from using their abstraction skills. *To be explored further.*		**Problem 27**: Students apply implementation knowledge too soon in the development process. *To be explored further.*

Model Complexity	**Problem 28**: Students have difficulties in understanding a design when the design consists of a lot of information. **Recommendation 28**: Consider the layout when preparing assignments. **Recommendation 29**: Teach according to a layout style (Scott Ambler)	**Problem 29**: Tools offer often more information than needed. **Recommendation 30**: When, available use auto layout and change level of detail		**Problem 30**: as model complexity grows, the cognitive load increases on every level. Often models are presented that consist of too many elements (rule of 7 +/- 2). **Recommendation 31**: offer students assignments with increasing complexity of the models. **Recommendation 32**: keep models simple and keep the amount of elements in a diagram low (7+/-2)	

General		**Problem 31**: Students find tools complex. They have a long and steep learning curve and most of the times difficult to install. **Recommendation 33**: Develop easy to use tools that don't require installation, like WebUML			**Problem 32**: Students have difficulties in understanding the role of the different UML diagrams in a development process. **Problem 33**: Students find it difficult to articulate their difficulties during an assignment. (they don't know what they don't know). **Problem 34**: Students do not automatically reconsider their designs. **Recommendation 34**: Train students to use their modelling activities with an agile approach (agile UML workshop), using pair modelling or peer reflection moments, in order to discover diagram relevancy within development cycles and to enable learning discussions. Discussion enables the re-thinking of the delivered design.

* *same kind of problem / solution*

Table A.1 summarises the problems identified in the research of this dissertation. Most of the problems have been investigated and are accompanied by a recommendation.

Appendix B

Coffee Machine Assignment

a Coffee Machine A coffee machine can make different types of coffee (one at a time): black, with sugar, with sugar and milk. The different types of coffee are poured into a cup. The front of the machine contains a pane with buttons. The buttons are used for selecting the desired drink. One can pay with a chip card or coins. The chipcard is read by a chip sensor and the coins by a coin sensor.

Appendix C

Tank Assignment

The Modelling Task – a Tank Game[1] In this task you will design a game with use of the UML class diagram. You do not need to use packages in this assignment. The description of the game is as follows: A player (user) controls a certain tank. This tank is a Panzer Tank, a Centurion Tank or a Sherman Tank. They fire bullets and Tank shells. Bullets can be Metal, Silver or Gold bullets.

A tank moves around a world (level). The aim is to destroy all other tanks in the world. After a world has been completed the tank advances to the next world. A list of all the worlds visited is kept.

An entire game consists of 8 levels. A world contains a maximum of 20 tanks that compete for victory. Each tank remembers which tanks it has destroyed in the past. The score for each level is kept by a scoreboard that gets notified by the individual tanks each time an opponent is shot. The players control their tanks through an interface allowing for steering, driving (reverse / forward), switching ammo and firing.

[1] text: B. Karasneh

Appendix D

Grading Rubric

Table D.1: *Class Diagram Rubric for Grading Design Modelling*

Grade	Judgement, criteria description
1	The student does not succeed to produce a UML diagram related to the task. He/she is not able to identify the important concepts from the problem domain (or only a small number of them) and name them in the solution/diagram. The diagram is poor and not/poorly related to problem description with a lot of errors: high number of wrong uses of UML elements mostly no detail in the form of attributes or operations.
2	The student is not able to capture the majority of the task using the UML notation. Most of the concepts from the problem domain are not identified. The detail, in the form of attributes or operations, linked to the problem domain is low. Some elements of the diagram link to the assignment, but too much errors are made: misplaced operation / attributes non cohesive classes few operation or attributes are used.
3	The student is able to understand the assignment task and to use UML notions to partly solve the problem. The student does not succeed to identify the most important concepts. A number of logical mistakes could have been made. Most of the problem is captured (not completely clear) with some errors: missing labels on associations missing a couple important classes / operations / attributes Logical mistakes that could have been made: wrong use of different types of relationships wrong (non logical) association of classes.
4	The student captures the assignment requirements well and is able to use UML notations in order to solve the problem. Almost all important concepts from the problem are identified. Some (trivial) mistakes have been made: Just one or two important classes / operations / attributes are missing Design could have been somewhat better (e.g. structure, detail) if the richness of the UML (e.g. inheritance) was used.
5	Student efficiently and effectively used the richness of UML to solve the assignment. The problem is clearly captured from the description. concepts from the domain / task are identified and properly named The elements of the problem are represented by cohesive, separate classes (supports modularity) with a single responsibility In the problem domain needed attributes and operations are present Multiplicity is used when appropriate Naming is well done (consistent and according to UML standard) Aggregation / Composition / Inheritance is well used No unnecessary relationships (high coupling) are included.

Appendix E

Class Diagram Evaluation Criteria

- ☐ Syntax – does the diagram follow the formal notation rules of the modelling language? At the time of writing UML 2.4 for example.
- ☐ Layout – Is the diagram readable? Does it not mix styles? For example: is an inheritance relationship drawn from top to bottom across the whole diagram?
- ☐ Consistency – is the diagram consistent with the other diagrams in the model? For example: does a class name corresponds with a lifeline in a sequence diagram?
- ☐ Semantics – Does the diagram represent what it should? For example: a '1 .. *' multiplicity is applied. Which means '1 to many'. Was this meant?
- ☐ Requirements satisfaction – does the diagram support the (functional) requirements? For example: Does the class diagram have a set of classes that cover the responsibilities of the system that is designed?
- ☐ Design principles – are common design principles applied? For example: did the student take coupling and cohesion into account?
- ☐ Design decisions – were the right design decisions made and applied? For example: a pattern is applied to support a quality requirement.