Implementation of Practical Exercises in Software Engineering Education to Improve the Acquirement of Functional and Non-Functional Competences

A field report about project-based learning in software engineering

Alexander Soska, Irmgard Schroll-Decker, Jürgen Mottok
LaS³ - Laboratory for Safe and Secure Systems
OTH Regensburg
Regensburg, Germany
{alexander.soska, irmgard.schroll-decker, juergen.mottok}@oth-regensburg.de

Abstract—**Software Engineering is a very volatile profession that requires a variety of theoretical as well as practical skills. In addition to expertise technical knowledge, graduates have to have a variety of social, methodical and personal competences. The acquirement of these non-functional competences are getting more and more important for a successful software engineer. To fulfill these requirements, it is necessarily important to prepare future professionals already during their college course of education. This paper presents exercises for a software engineering lecture with the goal to strengthen the students' practical experiences and to support the development of their non-functional competences. The developed exercises impart technical knowledge and encourage the students to improve their self-organized and lifelong learning. Thereby they are facing practical issues in all steps of the software engineering process while working on an inter semester project.**

Keywords—software engineering education; project-based learning; software engineering exercises; competences;

I. Introduction

In recent years, the discipline of software engineering has evolved into a wide ranged profession and specialists within are highly demanded from the industry. Due to the various fields of this profession, e.g. testing, quality management or requirement analysis, the graduates in software engineering have to provide a huge quantity of knowledge and skills to fulfill the required abilities. The different fields of expertise claim various technical and non-functional competences. Analyzing related job profiles for software engineering regarding cross-disciplinary competences, the results show some elements that appear in all investigated adverts. The main subject in all job adverts is to have the ability to work in a team. Out of 72 advertisements, 38 listed this element as needed non-technical competence. Besides this, 20 adverts demand communication and coordination skills in a working group from the future software engineer [1]. Based on this result, non-functional competences are inherent parts of requirements for software engineering professionals. Thus, it is necessary to mediate more than just functional competences in academic teaching. First practical experience as well as social, methodical and personal competences are important elements that are crucial for software engineering.

We assume that by integrating project-based learning and group work in software engineering education, the students' comprehension rises and their acquirement of functional and methodical, social and personal competences is supported. Based on the method of project-based learning, we assume that an ideal environment is given that supports the acquirement of competences in this subject. Within using this method, we assume that a regularly change of group size, setting and members as well as goals the students are subconscious demanded to improve their personal, communicative and social competences.

In the following chapter, we describe the consequences on academic education out of the bologna process and the change to competence-oriented learning. In Section III, we give a short introduction in project-based learning and why we believe it is sutable for teaching non-functional competences. In Chapter IV we describe our software engineering curriculum. Following in Chapter V, we present our academic learning objectives. In Chapter VI we describe how we adopt project-based learning and the adjustments we made for our software engineering education. Chapter VII illustrates the timeline and structure of the designed project-based exercises and their theoretical input. Subsequent in Chapter VIII, the designed exercises are presented. Chapter IX summarizes the results of this approach to the students learning followed by a short conclusion in chapter X.

II. Consequences of the Bologna Process

The bologna process and the consequent introduction of bachelor and master degree programs at universities set new challenges for lecturers. Instead of the solitary presentation of technical knowledge, the acquired competences of the students gain more importance in education. This results in a change of perspective in academic teaching that is often known as "The Shift from Teaching to Learning". It is demanded to focus on the learning process and less on pure mediation of content [2]. Subsequent, the learning outcomes change to the gained competences of the students. This competence acquisition is a process that is embedded in subjective contexts and situations and is reflected by personal characteristics. Each individual has to construct a connection between the supplied information on the content side as well as the embedded situational and social environment. According to constructivism, this learning process is a task that should be supported by the teacher, not determined. In order to achieve this change of perspective, the teacher has to develop a new approach of teaching software engineering. He has to design effective learning environments within which the learning process is focused. To maximize the outcome of students' learning, it is essential that the learners are actively integrated in the process of knowledge acquisition and problem-solving activities [3,4]. Thereby it is important that the learners are engaged with the learning targets and are able to successfully achieve the educational objectives. Based on practical examples they gain valuable experiences.

III. PROJECT-BASED LEARNING

Especially in technical fields like software engineering, practical know-how is a significant factor for successful comprehension. Therefore, the students should be early confronted with issues that occur during their later job life. Within, they are usually engaged in software projects and are confronted with problems that arise during the collaborative and inter-social work within the development of a software system. By regarding these occupational activities it is necessary to face the students with realistic software engineering projects. Thus we chose to develop our exercises in a dynamic approach called project-based learning in which students explore real-world problems and challenges. In this setting, the students are involved into a learning environment that tries to mediate learning content by own practical experiences. During the achievement of the projects' goals, the learner experiences new knowledge and can already transfer this to practical elements [3].

With this type of active and engaged learning, students are inspired to obtain a deeper knowledge of the subject matters. In addition this setting gives us the possibilities to carry out the demands of bologna and the constructivism. By defining clear and achievable goals of the project the learner gets engaged within and improves his or hers approaches. Due to his active participation in knowledge acquisition and problem-solving, supported by the lecturer and colleagues in group work, the student reflects and gets instant feedback on his activities [4].

Researchers have identified several components that are crucial to successful project-based learning. The following demands are postulated for designing this method [4,5]:

1. A realistic problem or project
    a. aligns with students' skills and interests
    b. requires learning clearly defined content and skills (e.g. using rubrics, or exemplars from local professionals and students)
2. Structured group work
    a. groups of two to four students, with diverse skill levels and interdependent roles
    b. team rewards
    c. individual accountability, based on student growth
3. Multi-faceted assessment
    a. multiple opportunities for students to receive feedback and revise their work (e.g., benchmarks, reflective activities)
    b. multiple learning outcomes (e.g., problem-solving, content, collaboration)
    c. presentations that encourage participation and signal social value (e.g. exhibitions, portfolios, performances, reports)

By integrating project-based learning into our software engineering lecture, we assume that the students have an ideal environment to focus on their learning process. Hence in exchange with each other, the students develop general and creative methods for problem solving as well as social and communicative competences. Thereby they are demanded to activate their acquirement of functional in connection with their non-functional competences. In addition due to the engagement with the technical content of software engineering they gain important practical experience in the subject matters.

IV. SOFTWARE ENGINEERING CURRICULUM

The presented exercises are part of the software engineering lecture in the fourth semester of the Mechatronics Engineering Bachelor at the Ostbayerische Technische Hochschule (OTH Regensburg). This bachelor is organized over seven semesters. In the first two semesters, an introductory C-programing course is set with a 90 minute lecture each week. A voluntary tutorial is offered parallel in the first semester and a mandatory practical course with 2 ECTS (European credit transfer system) credits is scheduled parallel in the second semester. The course closes in the second semester with an exam and is rewarded with 4 ECTS credits. In the third semester the basic of micro-controllers and assembler programming are presented in a course (5 ECTS credits) with two 90-minute lectures per week. In the fourth semester, this lecture is followed by a practical course (2 ECTS credits) and 90-minute lectures per week in C++ and software engineering. Each of these courses is worth 3 ECTS credits. Parallel to the C++ lecture, a practical course (2 ECTS credits) is scheduled. The curriculum closes in the fifth semester with a practical software engineering course (5 ECTS credits). The fifth semester is the practical semester in which our students are working for 18 weeks in real life engineering environments either in the industry or in the research facilities at our university. The lecture software engineering closes with an exam, while the practical course ends with a course-related certificate of performance.

V. ACADEMIC LEARNING CONTENS

The course software engineering provides an insight into all relevant parts of this profession. Treated contents are, for example, requirements engineering, architecture and design, testing or configuration management as well as their associated methods. According to the module catalogue for this lecture, the following contents are processed within:

• Process Models and Software Development Phases,
• Methodologies of Software Testing and Software Quality,
• Advanced Object-Oriented Programming Techniques,
• Databases and
• Design Patterns.

VI. Adjustment of Project-Based Learning to Software Engineering Education

Referring to chapter II, learning is the acquisition of technical, methodological, social and emotional skills. This process is embedded in a concrete environment and depends on it. Especially in such complex subjects like software engineering, practical experience is a key element for a successful comprehension. Theoretical elements should be strengthened regularly by the use of practical exercises [6]. By integrating realistic problems, the students receive a more practical view of problems and are able to combine software engineering methods and phases with their related use in software projects. Regarding demand one of chapter III, it is necessary that the students understand the sequences and the conjunctions in which the various phases of software engineering connect with each other. Thus we chose to develop an inter-semester project with the goal to develop a working software system in the end. For every phase, we provided the students with specific goals that have to be completed, supported by a lecturer.

Working together in various forms of group work is demand two in chapter III and can be seen as both convenient and efficient to support the development of these skills [7]. These methods increase the incentive for learning, raise motivation, and demand interpersonal exchange from the participants. Thus the learner acquires knowledge more effectively [6]. In addition, group work has its advantages especially in terms of creativity, reflection and quality of problem solving in providing opportunities to change perception of problems and lead to new solution approaches due to different personal knowledge, ideas and views [8]. We chose to change the group size and members from time to time we demand from the students to treat with new group members. Thus they don't fall in a certain routine when working with the same colleagues and have to contribute themselves to the new group members. In addition, the students' accountability for their individual input into the group stays high. Due to the change of size and members, the individual is demanded to impart itself in new situations and faces new communicative and social challenges. To produce successful team results, he or she is demanded to exchange with other positions and thoughts. To adopt actual working methods in software engineering, we integrated e.g. pair programing as setting to face the students with actual practical issues.

The changes in group work also promotes the claims of demand three in chapter III. The change of new members gives the individual multiple opportunities to receive feedback and receive the work. In addition due to the collaborative exchange new insights in problem-solving methods and in treated content are gained.

This project-oriented adoption to software engineering related issues and methods presents its technical content with practical elements and encourages students to deal with the topics in group work. To establish a transfer to the consecutive phases, the students are facing exercises that correlate on each other. Poor results in previous exercises affect the treatment with subsequent tasks in the development of the software system [9].

VII. Methodical design of project-based learning exercises

The intention of the exercises is to strengthen the students' knowledge and to give them some experience with subject matters. Thus previous theoretical input is necessary. With the exception of requirements engineering, we designed the integration of the exercises according to the principle "from theory to practice" [8]. The intended content of the exercises is roughly grouped into four main parts of software engineering; requirements engineering, UML (Unified modeling language) diagrams, design patterns and test. Due to the limited time for conducting an exercise (90 minutes) and the huge content of UML we separated this subject into two parts, UML I and UML II. Table 1 illustrates the timeline for the designed software engineering lecture.

Altogether there are 15 weeks per semester whereat the last week is reserved for exams. Software engineering lecture is determined with a 90 minutes lecture per week. Thus the students have a weekly change of theoretical input and practical execution in between they can prepare themselves with the content. If needed, additional material is provided at least one week before the exercises through our moodle platform. These material mainly consists of literature or software which is needed to conduct the exercises.

TABLE I. Timeline for the conduction of the Software Engineering Lecture

Week	Content	Method
1	**Introductory course**	Lecture
2	**Process models**	Lecture
3	**Requirements Engineering**	Exercise
4	**Requirements Engineering**	Lecture
5	**UML I**	Lecture
6	**UML I**	Exercise
7	**UML II**	Lecture
8	**UML II**	Exercise
9	**Design Pattern**	Lecture
10	**Design Pattern**	Exercise
11	**Software Test**	Lecture
12	**Software Test**	Exercise
13	**Refresher Course**	Lecture
14	**Exam Preparation**	Lecture
15	**Exam**	Exam

As mentioned in previous chapters, practical experience is a key element in software engineering. At the end of the exercises, the student's goal is to develop a software system. For this, at the end of every exercise, the students have to deliver interim results which are reviewed by the lecturer and used for the next phase. If wrong, the students get feedback and can correct their results before moving to the next steps in the software engineering process. If the results of a previous exercise are not revised and are still incomplete or incorrect, the execution of the consecutive phase is demanding. In consequence, the students are responsible for themselves if the development of the software system lacks or stagnates. Thus, they distinguish how various decisions and methods affect the phases in a software engineering process and have a direct reflection on their work.

VIII. CONTENT OF PROJECT-BASED LEARNING EXERCISES

In the exercises, the students have to develop a software system for a coffee dispenser. The associated simulation interface can be seen in Figure 1. At the beginning of the introductory course, the students are presented the timeline of the lecture and a working simulation of the software system. Thereby, they are already informed at the beginning of their software engineering course and can imagine how the software system should work.

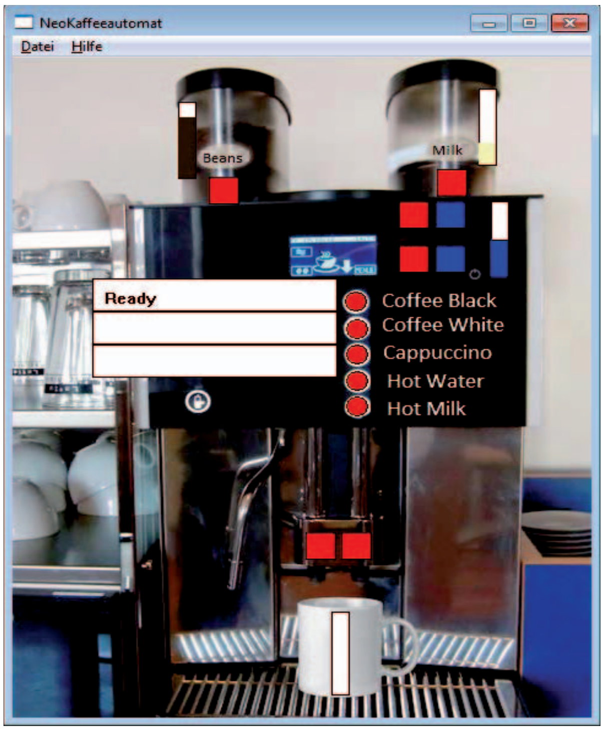


Fig. 1. User-Interface of the coffee dispenser simulation

As element of project-based learning, the students need to perform their exercises in group work. Thus, they are embedded in a learning network and share their activities within. Due to the diverse skill levels and independent roles, they need to collaborate to produce successful outcomes. By switching or decreasing the group size (e.g. by performing pair programing), the students get involved with each other. This promotes their social and communicative competences and their competences in conflict management because they have to deliver group results which demand output oriented collaborative group work.

In addition, it is ideal if the students have first contacts with job-related software. Due to the installation of Enterprise Architect on our university CIP (Computer Investitions Program)-Pools, we use this software as often as possible.

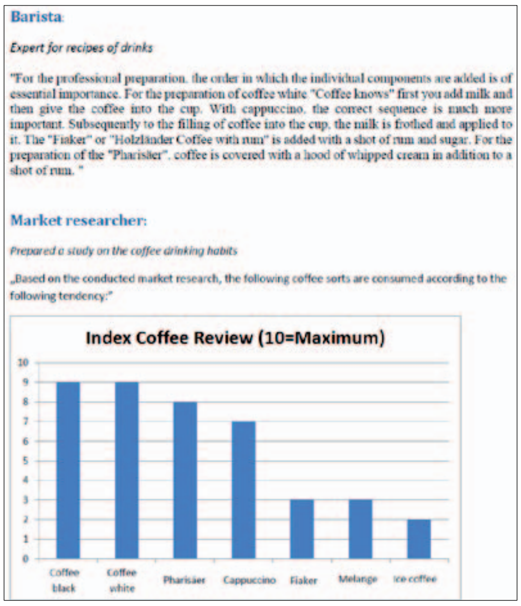
In the following chapter, we explain the exercises. The task is that the students are software engineering employees and have to establish a software system for a coffee dispenser based on the V-Model as process model.

IX. PROJECT-BASED SOFTWARE ENGINEERING EXERCISES

A. Requirements Engineering

In this exercise, the students should adopt the difficulties in performing Requirements Engineering, especially in analyzing the requested specifications of the customer. In consequence, they gather the important information for their role as software engineer and differ between functional and non-functional requirements. At the end of the exercise, a requirement document is developed which lists the specifications for the desired coffee dispenser system according to professional principles.

In contrast to the reality where the requirements are gathered during a discussion with the costumer, the statements are already collected in the form of fictive written interviews (see Figure 2). During the exercise, the various groups have to analyze the interviews and identify the requirements for the software. Due to the large quantity of participants, usually 20 - 40 students, the provided time for the exercise would be exceeded if performed in face-to-face communications. Thus, we prepared interviews of altogether eight different persons, e.g. the quality manager or the company director, to illustrate different point of views and desires for the software system.


Barista:

Expert for recipes of drinks

"For the professional preparation, the order in which the individual components are added is of essential importance. For the preparation of coffee white "Coffee knows" first you add milk and then give the coffee into the cup. With cappuccino, the correct sequence is much more important. Subsequently to the filling of coffee into the cup, the milk is frothed and applied to it. The "Fiaker" or "Holzländer Coffee with rum" is added with a shot of rum and sugar. For the preparation of the "Pharisäer", coffee is covered with a hood of whipped cream in addition to a shot of rum. "

Market researcher:

Prepared a study on the coffee drinking habits

„Based on the conducted market research, the following coffee sorts are consumed according to the following tendency:"

Fig. 2. Exemplary interviews for requirements engineering exercise

To differentiate between relevant and useless requirements, some additional statements in the interviews are included but not relevant. In addition, there are also some fragmentary information available out of the interviews that need additional material. The intention is to stimulate the students to ask for the lack of information. Thereby, the lecturer appears as mediator to the costumer and adds missing information on inquiry. Furthermore, after the half of the processed time, the students are given new requirements. This causes them to practice change management already in the act of requirements engineering.

As result, the students deliver a document that includes all relevant requirements. Within they need to distinguish between functional and non-functional requirements and define a nomenclature for the purpose of traceability and uniqueness. To help them how to define the specification, we provided them with a requirements template.

B. *UML I*

In this exercise, mainly UML-diagrams are treated which describe the behavior of a system. These describe the dynamic aspects of a system.

1) Use-Case-Diagram

Regarding its complexity, the use-case-diagram is one of the easier models compared to other UML-diagrams. During this exercise, the students should establish a use-case-diagram based on their previous results in requirements engineering. Thus, they need to understand that this diagram only shows the possible user interactions with the system.

The designed exercise for this UML-diagram is adopted on the method capture cards. After separated in groups from 4 – 6 persons, each student is given several moderation cards. During the following 5 – 10 minutes, each student writes suggestions for possible user-interactions with the system on the cards. Following, in group work the individual suggestions are collected and discussed. Together, the students cluster their cards and place them on a pin board. To finish this exercise, they have to complete the use-case-diagram by creating the system borders as well as the notation elements (see Figure 3).

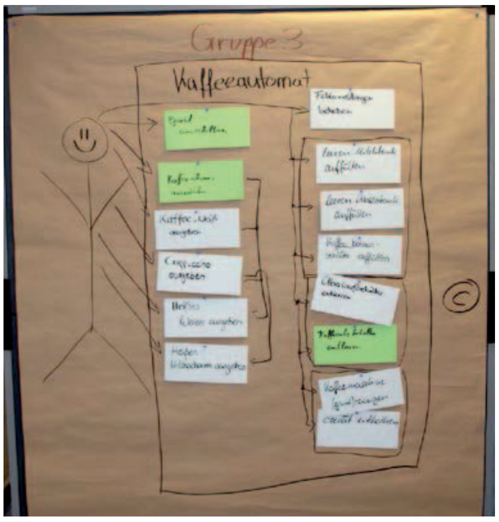


Fig. 3. Creating use-case-diagrams on pin boards

Due to this setting, all students are actively participating and discussing their suggestions. By comparing their assumptions, the students gain instant feedback on their thoughts and reflect their activities. Mostly the students argue about the boundaries of the use-case-diagram. They have difficulties in the differences between the user interactions with the system and the tasks that run based on user activities.

2) Finite-State-Machine

In this exercise, the students should gain first experiences with finite-state-machines. State machines refer to the states of a system based on certain activities. According to this diagram, the system is described by the states which an object has during its lifetime. Therefore, it is necessary to know how to identify and distinguish between state, event and action and learn how to establish correlations between these elements. Thus, it is often a tough learning task to understand the principles of states when regarding a software system; we did not want the students to start from the beginning.

When creating diagrams of finite-state-machines, the students should know how to sketch the model and to know about the correct nomenclature. To accomplish an introduction into this UML-Model, the idea is to give the students an incomplete finite-state-machine in which they should insert given texts for events and actions (see Figure 4). Thereby, each proposed text fits into the gaps only once. Thus, they can recognize the way how this diagram is build and have examples for the following task.

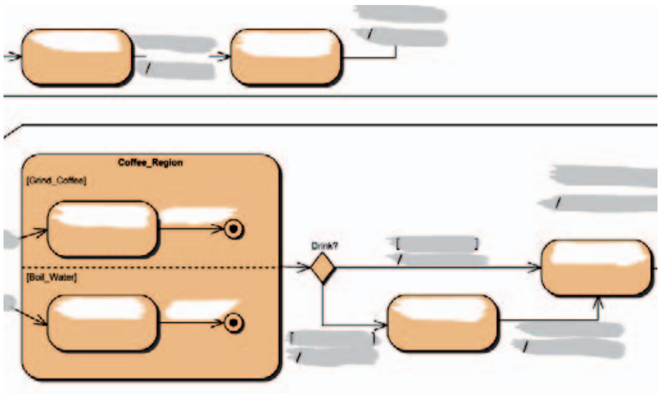


Fig. 4. Excerpt of the work sheet for the exercise finite-state-machine

To complete the diagram, the students are working together with one colleague on finishing the fragmentary finite-state-machine. Together they can exchange themselves and discuss their ideas for the insertion of states, events or actions. This promotes the social interaction of a student due to the communicative exchange between the students. Thereby they gain insights into other views and ideas but have to finish the task on their own. At the end of the exercise, every student should have completed the fragmentary diagram and possess an own solution of this exercise.

C. *UML II*

In this exercise, mainly UML-diagrams are treated which describe the structure of a system.

1) *Class Diagram*

A class diagram illustrates each used class of a software system and their relations between each other. A class diagram is one of the most frequent used diagrams when describing a software system.

For the following exercise, we expect the students to have already little experiences in defining and describing classes in object-oriented programing due to their first contacts with C++ in a parallel lecture. For the exercise, the students are given various classes and associations related to the coffee dispenser printed on paper pieces. The goal is to establish a class diagram in groups a 4 – 6 persons for the coffee dispenser based on the previously specified requirements and the use-case diagram as well as the finite-state-machine. We provided the students with all necessary types of coffee dispenser classes (e.g. abstract, virtual) and all possible connections. The pieces include all necessary information like attributes and methods that are needed to understand the use and dependencies. Figure 5 shows one solution made by a group of students.

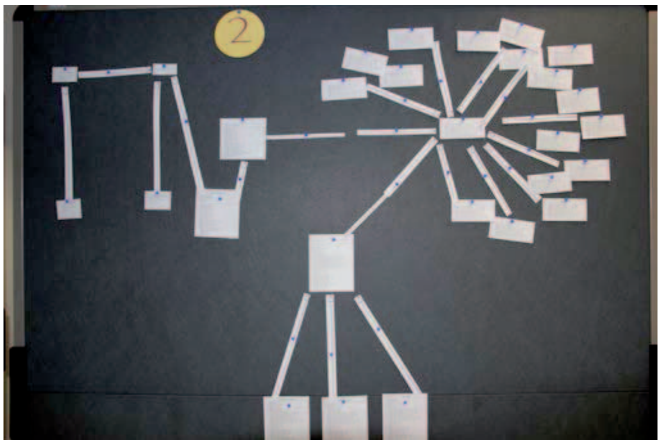


Fig. 5. Overview of students solution for a class diagram

Due to this approach, the students repeat their basic knowledge about the topic and the repetition of definitions or nomenclatures. Based on different personal experience, the students exchange themselves in the group when trying to develop a solution.

2) *Sequence Diagram*

This diagram displays the exchange of messages within a system. Thereby the focus is on the chronological sequence of messages that are send from one object to another. Similar to the example for the finite state machine, the students in groups of two persons are given the frame of the coffee dispenser sequence diagram that they fulfill based on given texts. Again, every text just fits once.

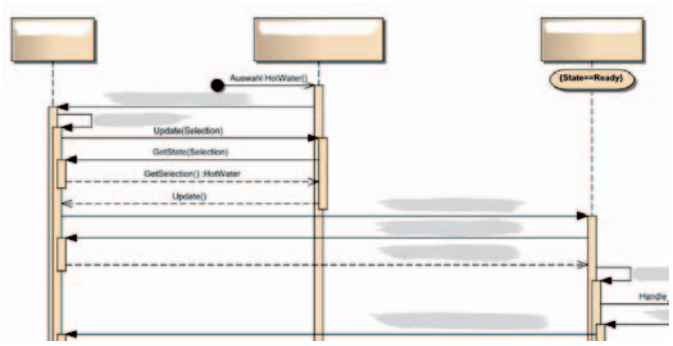


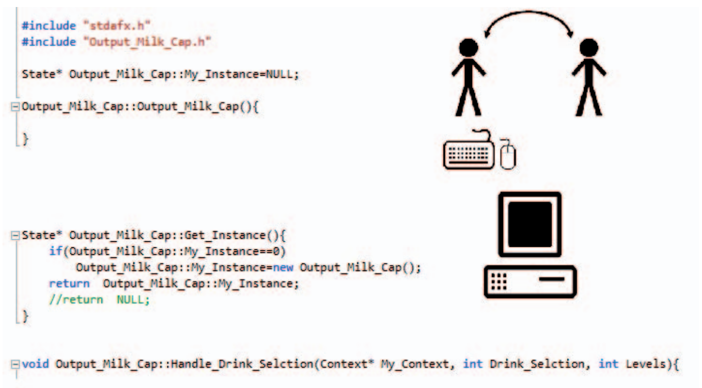
Fig. 6. Extract of the work sheet for the sequence diagram

After the students have finished the insertion of the texts, the fulfilled sheets are passed to a group that checks the sheet on their opinions and insights. Thereby they make notes that are discussed in the following phase of this exercise. Here, the two groups are set together and have to discuss about their suggestions and critics on the solutions. Out of this following discussion, the two groups develop together a short presentation for the solution of this exercise. At the end of the exercise, each representative of this cooperative group work presents his or her solution for this exercise.

D. *Design Pattern*

Pattern has been defined as „an idea that has been useful in one practical context and will probably be useful in others [10]". Design Patterns are basic knowledge for every software engineer and are used as in the development of a software system. To understand design patterns, it is essential to have basic knowledge in object-oriented programing and in designing the architecture of software systems. In literature, the patterns are mostly described very abstract and have no specific coding examples that help to transfer pattern to actual problems. This said, to use design pattern in developing software systems experience is needed.

For this exercise, we provided the students with a framework for the coffee dispenser software. Within this fragmentary code are specific classes that have to be implemented by the students. For the coding, the students use their previous solution for the class diagram and sequence diagram. To give a brief introduction in how to use design patterns, at the beginning of the exercise a short presentation by the lecturer is conducted. Afterwards, the task is to implement particular classes that are designed according to certain design patterns in pair programming. This setting was chosen due to its rising use in industrial software engineering context. Thereby, two students work on the implementation of code while their roles change regularly between programmer and viewer. The goal for the students is to complete the software framework on the basis in the previous diagrams as well as the presentation and to establish a running simulation of the coffee dispenser. To avoid that students are fixed in roles, we provided them with a rhythm for changing their roles in pair programing.



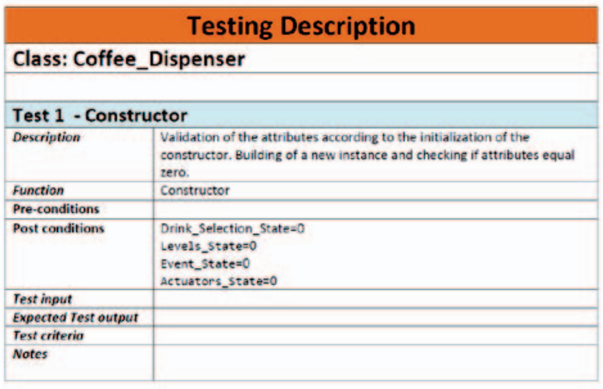
```
#include "stdafx.h"
#include "Output_Milk_Cap.h"

State* Output_Milk_Cap::My_Instance=NULL;

Output_Milk_Cap::Output_Milk_Cap(){

}



State* Output_Milk_Cap::Get_Instance(){
    if(Output_Milk_Cap::My_Instance==0)
        Output_Milk_Cap::My_Instance=new Output_Milk_Cap();
    return  Output_Milk_Cap::My_Instance;
    //return  NULL;
}

void Output_Milk_Cap::Handle_Drink_Selction(Context* My_Context, int Drink_Selction, int Levels){
```

Fig. 7. Exercise for design pattern in pair programing

E. *Testing*

The awareness for software testing is one of the key elements of software engineering. Every phase in a software engineering process has its specific testing methods. Software-test is the process of planning, preparation and measurement of project properties with the goal to distinguish differences between the system and its specifications.

For the selected exercise, the students are again set in pair programming. The students are given a specific class of the already implemented system. Due to the usage of this setting in the previous exercise, the students gain more experience within. Following they have to execute concrete test cases which are prepared by test description. These describe the tested classes, preconditions and post conditions as well as the expected test results (see Figure 8).



Testing Description	
Class: Coffee_Dispenser	
Test 1 - Constructor	
Description	Validation of the attributes according to the initialization of the constructor. Building of a new instance and checking if attributes equal zero.
Function	Constructor
Pre-conditions	

Post conditions	Drink_Selection_State=0
	Levels_State=0
	Event_State=0
	Actuators_State=0

Test input	
Expected Test output	
Test criteria	
Notes	

Fig. 8. Test description for testing exercise

To perform this, we use a testing framework called CppUnit within which the students have to implement their test cases according to the descriptions.

X. Effects of Project-Based Learning to Software Engineering Competences

The students are involved into exemplary and researched learning according to the principles of project-based learning and self-organized knowledge construction by performing group work embedded into a successive practical project. By performing these exercises, various experiences are collected. Thus, a written evaluation about the acquirement of competences will follow in the next semester; the following statements can be made according to observations and talks with the students:

- The technical contents are repeated and deepened.

- Due to the exercises, the comprehension of the technical content raises.

- In order to meet the goals the students need to engage in a self-organized and self-responsible learning-process.

- Due to demanded preparations for the participation to the exercises, the students are improving their active treatment with the technical issues.

- The students increase their interpersonal competences by solving problems in their selected groups.

- Performing Requirements engineering and UML exercises are simple tasks, while teaching design patterns, implementation or testing need additional support by the teacher.

By performing this adoption of project-based learning within software engineering education, the several demands of chapter III are fulfilled. Due to the introduction of the coffee dispenser as realistic problem or rather project the students acquire practical experiences and strengthen their functional competences. Personal selective reflections of the students show that by conducting this project the content of software engineering is well transferred and aligns with the interest of students of frequent practical examples. In addition, the order in which we designed the exercises meets the students skills and their progress during the lecture. Requirements engineering is seen as the easiest phase of software engineering whereat testing is tough to accomplish.

Due to the structured group work, the students are animated to exchange themselves with each other. By changing the size, setting and members, the individual is embedded in new situations in which it meets diverse skill levels and roles. Thereby the student needs to treat with new situations and activate his or hers communicative and inter-social competences. Especially in the demand to hand over personal or team solutions, the student needs to engage him- or herself with the group. Due to the change of setting and membership of groups, new situations are made. How the students meet these settings changes over the semester. Once being used to group work, the engagement in different groups rises and is directly set to the progress in achieving the goal. In addition, the way the students start their problem-solving methods change during the exercises. At the beginning they are waiting for additional input by the lecturer whereat at the end of the exercises they are self-active looking for additional literature and information to complete the exercise.

XI. Conclusion

Altogether the students show interest in new teaching and learning methods and are willing to contribute. Active participating is welcomed and conducted. The designed and integrated exercises gain acceptance among the students and lead to a better understanding as well as an active treatment with software engineering topics. Due to the different difficulties of the software engineering elements, the exercises need various support by the teacher. In addition the experiences with the implementation of the presented exercises in software engineering education show that these are necessary and suited to strengthen students theoretical knowledge and their practical skills within.

Based on observations and talks, the students social competences rise during conduction of the exercises. This said, in the next there will be a written evaluation about the specific acquirement of the competences. Thereby we will perform several tests, before, while and at the end of the exercises, lectures and semester to record the student's competence development.

Acknowledgment

This work is supported by the Federal Republic of Germany, Federal Ministry of Education and Research, BMBF grant EVELIN "Experimentelle Verbesserung des Lernens von Software Engineering", grant identity 01PL12022F, project sponsor DLR. The network project EVELIN is part of the „Bund-Länder-Programm für bessere Studienbedingungen und mehr Qualität in der Lehre". The Universities of Applied Sciences Aschaffenburg, Coburg, Kempten, Landshut, Neu-Ulm and Regensburg are the network partners. More information under http://www.las3.de or http://www.qualitätspakt-lehre.de. Solely the authors are responsible for the content of this publication.

References

[1] N. Stoiber, Erstellung eines Kompetenzprofils von Softwareentwicklern. Regensburg: master thesis, 2013, unpublished.

[2] Schneider, Ralf; Wildt, Johannes: Forschendes Lernen und Kompetenzentwicklung. In: Huber, Ludwig; Julia; Schneider, Friederike (Hg.): Forschendes Lernen im Studium, a.a.O., S. 53-68.

[3] Barron, B., & Darling-Hammond, L. (2008): Teaching for meaningful learning: A review of research on inquiry-based and cooperative learning. Powerful Learning: What We Know About Teaching for Understanding. San Francisco, CA: Jossey-Bass.

[4] Mottok, J., Hagel, G., Utesch, M., Waldherr, F., Konstruktivistische Didaktik- ein Rezept f¨ur eine bessere Software Engineering Ausbildung?, Embedded Software Engineering Congress, Sindelfingen, 2009.

[5] Project-Based Learning Research Review | Edutopia (2014). Online verfügbar unter http://www.edutopia.org/pbl-research-learning-outcomes, last update 22.06.2014.

[6] Erpenbeck, John; Heyse, Volker (2007): Die Kompetenzbiographie. Wege der Kompetenzentwicklung. 2. Aufl. Münster: Waxmann.

[7] Mayo, P., Donnelly, M.B., Nash, P.P., & Schwartz, R.W. (1993) Student perceptions of tutor effectiveness in problem based surgery clerkship. Teaching and Learning in Medicine, 5(4), 227-233.

[8] Stipek, Deborah J. (2002): Motivation to learn. Integrating theory and practice. 4. Aufl. Boston: Allyn and Bacon.

[9] Diaz-Herrera, Jorge L. (1994): Software engineering education. 7th SEI CSEE Conference, San Antonio, Texas, USA, January 5-7, 1994 : proceedings. Berlin, New York: Springer-Verlag (Lecture notes in computer science, 750).

[10] Fowler, Martin (1996-11-27). Analysis Patterns: Reusable Object Models. Addison-Wesley. ISBN 0-201-89542-0. "A pattern is an idea that has been useful in one practical context and will probably be useful in others."

The project EVELIN is funded by the BMBF (Bundesministerium für Bildung und Forschung) with the grant number 01PL12022F, promoters DLR (Deutsches Zentrum für Luft- und Raumfahrt). Further information at www.qualitätspaket-lehre.de