Proof of Competence
UTQ Module 21

Alberto Bacchelli

I. Course description and context

I base this proof of competence on the course Software Engineering Methods (SEM), which I taught as main instructor in the first quarter of 2013 (under the code TI2205). In the next editions, I will continue to be in charge for this course, and I am using this proof of competence as a starting point to improve how I will deliver and assess it.

SEM is a mandatory course of the second year bachelor of Technische Informatica (TI), it accounts for 5 ECTS, and lasts one quarter. In 2013 it involved more than 120 students. The normal scheduling comprises two lectures (two hours each) and one laboratory session (four hours), each week. It belongs to the learning line of "Software Development Fundamentals," together with Objectgeoriënteerd Programmeren (OP) (first year, first quarter), Algoritmen en Datastructuren (first year, third quarter), Softwarekwaliteit en Testen (ST) (first year, fourth quarter) and Concepten van Programmeertalen (second year, third quarter). The only formal prerequisite to SEM is OP and its project (first year, second quarter). To be sure that students see the connection between OP and SEM, I continuously show them, with tangible examples, how programming is only a part of the story when it comes to real-world software development. Software organizations (both industrial and open-source) develop software with sound "software engineering methods." These are the methods that students will learn to recognize, judge, and apply during this course.

SEM is the basis for the Context project (second year, fourth quarter, 10 ECTS), in which students are required to form 5-person groups and implement software solutions for real-world stakeholders, by following appropriate software engineering practices. I am also main co-teacher (together with Prof. Alan Hanjalic) of the Context project, I particularly supervise software engineering related aspects. During SEM lectures I often anticipate how the topics I present to students will be useful in the course of the Context project.

I am the only responsible for the SEM course: I deliver lectures, prepare and grade (with the help of teaching assistants) assignments, and conduct exams. I collaborate with external experts who deliver guest lectures on their topics of expertise. For the edition of 2013, SEM included the following guest lectures: Software Visualization (Prof. Dr. Michele Lanza, University of Lugano, Switzerland), The Power of Scrum (Prof. Dr. Rini van Solingen, TU Delft), Human Computer Interaction (Dr. Willem-Paul Brinkman, TU Delft), and Measuring Software Product Quality (Dr. Eric Bouwers, Software Improvement Group, Amsterdam). These lectures help students connecting to real-world usage of the topics they learn, and give them the occasion to interact with world-experts on the topics.

II. Students' entry levels

Software Engineering Methods is a mandatory course and it requires knowledge about the basics of programming and the object-oriented paradigm. Such a knowledge is assured by the prerequisite course Objectgeoriënteerd Programmeren (OP), taught in the first year, first semester, by Dr. Andy Zaidman. To ensure that (1) all the topics required for SEM are presented in OP and (2) there is minimal overlap with SEM topics, I have several meetings with Dr. Zaidman.

Some students attending SEM might not come from the normal TI curriculum (*e.g.*, they come from different bachelor courses and have to do SEM as a bridge to access a specific master) and thus might lack the programming and object-oriented background necessary for this course. This happened in 2013. The solution I adopted to raise all the students to the same level was to deliver two additional elective lectures on such topics. Only students with lacking background came (less than 25) and we had fruitful interactive lectures, with hands-on sessions, in which students managed to drive the speed of the explanations according to their needs. In the next editions, I plan to still raise all the students on the same level by using special lectures; however, I plan adopt the *flipped classroom* approach, because the topics they have to learn are popular, simple to grasp (at least in the basics), and there is a good deal of superb videos and tutorials covering them. Most of the learning will be done outside class, and the class time will be spent to answer clarification questions. This approach is likely to also interest students with a normal TI background who want to understand these topics better.

III. Learning objectives

A. Before

The learning objectives I set for the "Software Engineering Methods" course in 2013 are the following:

1) The main goal is to give the students the instruments to become Software Professionals.
2) Participants understand the most important software engineering practices needed to build high quality software systems.
3) Participants can apply modern software engineering techniques to create high quality maintainable and evolvable software projects.
4) Participants can reflect about limitations of current software engineering practices, know when and when not to apply them, and are aware of the latest research developments aimed at addressing these limitations.

B. Problems

Thanks the knowledge acquired in the Module 21, I could see several problems with the aforementioned learning objectives.

- *"The goal is to give the students the instruments to become Software Professionals."* This goal is not quantifiable. How can I measure whether they become something? Moreover, what is really a *"Software Professional"*?
- *"Participants understand the most important software engineering practices . . . "*. This is the classical *understand* problem: I cannot directly measure whether students understand. I have to assign them activities whose outcome I can measure and that practically demonstrate their understanding level.
- *"Participants can apply modern software engineering techniques"* This goal is more appropriate than the previous ones, but it is not clear the context in which the students are expected to apply the techniques. Although it is good to use generic terms that go beyond the scope of the course, this is too generic.
- *"Participants can reflect about limitations"*. This goal would be better expressed with a more action-oriented verb: If I want them to say the same things I explain in class I should use something along the lines of ?describe, list, recall?; if I want them to think and use the content explained in class to derive their own knowledge, I should use: ?evaluate, argue, criticize?.
- The third goal packs too many goals together: It would be probably better to split them, so that each of them can be clearly assessed and addressed in class/tests/laboratory.

C. After

In the following, I detail the learning objectives set for the next edition of this course (2014, Q1). The learning objectives roughly follow the Bloom's taxonomy. The higher the goal in the taxonomy, the more weight it will have toward the final grade of each student.

LO1: Students are able to **recall** and **list** the most important software engineering practices designed and used to build maintainable and evolvable software systems.

LO2: Students can **describe** the most common applications of the presented software engineering practices.

LO3: Students can **apply** the presented software engineering practices to a new software project.

LO4: Students can **analyze** the usage of the presented software engineering practices in existing software projects.

LO5: Students can **judge** the real benefits/drawbacks of using each software engineering practice in a given software project.

IV. Constructive alignment

Table I shows the alignment among learning objectives, activities, and assessments. Concepts of constructive alignment influenced my course design in several ways. Starting by clearer learning objectives (instead of mere topics I had to transmit to students) helped me to filter and focus on the parts of the course that are the most important for the students to achieve the objectives. Then, forcing myself to reflect and map objectives to teaching activities helped me in shaping differently the way in which I present the topics, by using techniques that are more engaging to the students. The assessment was the part that received less benefits from this table, because it was natural to me to align teaching and assessment, since I also design laboratory work and weekly assignments. Nevertheless, the table helped me to reflect at a higher perspective on the more effective types of assessments I should use for my objectives.

V. Active teaching and learning

Although I had already included some active teaching and learning elements in my course in 2013, I was only unconsciously aware of their importance and role for the students. Module 21 gave me the right framework and mindset to reflect and develop better and more consistent active teaching/learning moments. Also the feedback from the students helped me realizing the importance of this topic in fostering their interest in the subjects of the course.

The third learning objective is the central point connected to active teaching/learning: "Students can apply the presented software engineering practices to a new software project." This is a very important skill to be achieved by the students, because they will use it to develop software in real-world scenarios, both during the context project and the Bachelor project, and after they will finish their studies. I see active teaching/learning as a great opportunity to achieve this objective (although it is not limited to this).

In the first lecture of SEM, I will ask students to form teams of five people (the same size of teams for the context project) with the colleagues they collaborate better with (groups will be generated by students themselves; I will only help creating groups for students who are "left alone"). Subsequently, at the end of the first weekly assignment, I will present them with the specifications of a famous game/application (*e.g.*,) and ask them, in the first week to develop a working prototype of it with their current programming and software engineering knowledge. This prototype will be the basis of an important exercise of each weekly assignment. The idea is that students will gradually rework their prototype week by week, lecture by lecture, by applying the techniques they learn in class. This will have two main benefits: (1) They will always relate what I am teaching to their actual product, and (2) they will see the tangible differences in the quality of their product, between the first version (created only with the knowledge from OP) and the last (after applying the knowledge they acquired in the course).

This group project will be an active learning activity, because students themselves will have to realize how the knowledge they acquire in class can be applied to their case, what the benefits and drawbacks are, and how to take decisions and justify them within their group of teammates. I will be available to discuss students' decisions within the lab hours, together with the teaching assistants, and we will evaluate their choices weekly by analyzing their source code and practical outcome.

TABLE I
CONSTRUCTIVE ALIGNMENT, BY LEARNING OBJECTIVE

LO	Teaching/Learning Activities	Assessment	Explanation of the alignment
1	Present in class the software engineering practices, using a *problem-based learning*: Ask how some problems can be solved according to their experience, and show that these problems are elegantly solved using certain well established practices.	Individual active writing exercises in the weekly assignments in which the reiterate what said in class. Graded. Final exam will contain related recalling questions.	This is the most basic requirement for the course. The *problem-based learning* is used to show them the relevance of the problem. Their knowledge is assessed through graded exercises in the weekly assignments.
2	Frontal lectures explaining the applications, interleaved with in-class *active learning* (*e.g.*, brain dump in the break), *collaborative learning*, *etc.* The idea is to have many different active learning activities to keep students engaged lecture by lecture.	"How to" exercises in the weekly assignments, in which students describe the steps in which how software engineering methods presented can be used to solve a problem. Final exam will contain related theoretical questions.	The "how to" exercises ensure that the students are able to understand all the steps of the presented practices. To write an appropriate "how to" they are forced to describe it in the scenario of an application of the practice. The teaching activity is aligned, because at the understanding level of the Bloom's taxonomy.
3	*Team-based learning*: Students will form teams to implement a software project by applying the practices they learn in class. This is done out of class, the group work will foster critical thinking and guide application of the concepts.	Team project exercises in the weekly assignments, which will guide students to iteratively construct their software project in team. Each week the exercises are evaluated and the decision of the students discussed with them. Final product will have a very important weight in the final grade.	Having students implement a fully working software system ensures that they are able to apply the presented topics. The frequent (weekly) feedback will both assess the quality of their work and keep them on the right track.
4	*Case Studies* presented in class of how the software engineering practices are used in real world scenarios.	Include exercises in the weekly assignments that will ask students to find real world applications of the presented software engineering practices, and ask to analyze how the practices are used.	Both teaching and assessing involve the analysis of real world examples.
5	*Class discussions* about concrete examples of correct/faulty usage of software engineering methods in real world software, which lead to failure/success. Using *brainstorming* to activate students before the class discussion.	Include exercises in the weekly assignments that will ask students to find real world applications of the presented software engineering practices, and ask for their judgment on their implementation. Graded. This is the highest learning objective, an exercise will be in the final exam asking to judge an example.	Brainstorming followed by class discussion will help critical thinking about the usage of the practices, and the assessments will verify the reasoning of the students.

This will help creating a quick feedback cycle to help both me (by assessing the understanding level of the students) and the students (by verifying the validity of their choices) to improve.

VI. CONNECTION WITH REAL-WORLD

The subjects taught in SOFTWARE ENGINEERING METHODS are deeply rooted in the current practice of real-world software development. Although not all the companies might follow all the presented methods equally, there is a high chance that students will have to make use of most of these methods once they will work for IT or software companies. Even if students will occupy managerial roles, many of the presented methods (*e.g.*, project management and planning) will be relevant to their practice. Although the presented methods are in current wide-spread use, it is very likely that they will change in the future. For this reason, students not only have to *recall*, *list*, and *apply* the taught topics, but they have also to learn how to criticize them and see their benefits and drawbacks.

To demonstrate the actual connection of the presented topics with real-world development, I will rely on three main sources.

First, they will experiment the benefits of the presented methods while developing their our group project: They will be able to see that following sound techniques, they can program better and more maintainable code; moreover they will collaborate and coordinate with more success. Second, I will present in class real-world open-source systems that make use of the subjects taught and discuss in class with the students the consequent advantages and drawbacks; in this vein, weekly assignments will also require students to personally find real-world examples in which software engineering methods are applied and criticize them. Third, I will share in class my own personal industrial experience at CINECA[1] and Microsoft, and I will back it up by referring to software engineering research I conduct; I already employed this in 2013, and I noticed that students are interested and easily engaged when sharing this type of real-life experiences.

VII. Conclusion

To conclude, the main benefit of module 21 is that it gave me the right framework to design an engaging and clear course. I already used a number of techniques to engage students and help their learning, and I also already developed a carefully designed SEM course before starting this module; in terms of the final outcome on students' acquired knowledge I was already on the positive side. Nevertheless, after following module 21, I realized that it was naive to design a course and its content, without taking into account years of valuable research on the teaching topic. It opened my eyes on the importance of clear learning objectives (not only for the students but also for me to design the course) and the practical difficulty of making a correct constructive alignment. Active teaching/learning was already part of my course, but module 21 showed me a huge number of additional ways to activate students and improve their learning: This is a great resource that I will use day-to-day in class to engage students in discussions and critical thinking.

Overall, module 21 gave me the necessary basis to develop better courses in the next years and to even more appreciate the efforts I am putting in teaching my courses, because they are backed up by years of research in this sense.

[1] http://www.cineca.it/en