4th International Conference on Higher Education Advances (HEAd'18)
Universitat Politècnica de València, València, 2018
DOI: http://dx.doi.org/10.4995/HEAd18.2018.8103

Designing Lectures as a Team and Teaching in Pairs

Zehetmeier, Daniela[a,b]; Böttcher, Axel[a] and Brüggemann-Klein, Anne[b]
[a]Department of Computer Science and Mathematics, Munich University of Applied Sciences, Germany, [b]Department of Informatics, Technical University of Munich, Germany

Abstract

A technique that is frequently used in modern software development is the so-called pair programming. The proven idea behind this technique is that innovative work in a highly complex environment can benefit from the synergy between two persons working together with well-defined roles.

The transfer of this technique as a metaphor for teaching has repeatedly been reported as a successful teaching strategy called pair teaching. In this paper, we describe our experiences with designing and teaching a complete lecture on software development as a pair.

Our contribution is the definition of patterns for role-assignments to both persons. These include patterns for the design of the lecture as well as patterns for the teaching in class itself. Our experience shows that there also exists a couple of anti-patterns namely role distributions that should be avoided.

First evaluation results are promising in the sense that the reception of structure and content as well as students' satisfaction increased significantly with the introduction of pair design and pair teaching.

Keywords: *Pair teaching; team teaching; computational thinking; cognitive apprenticeship.*

1. Introduction

The presentation and discussion of topics by two people is common practice in several fields, like educational TV-shows, debates, or in interviews. Furthermore, working in pairs has become a traditional approach in the field of software engineering and is called pair programming. As this practice has proven highly efficient (Williams, 2000) and we are familiar with it, we transferred it to our lecture design and later to our teaching.

In this paper, we present our experience with designing lectures in pairs and later teaching them in pairs. We already did this for several small units over the last year and the results were consistently positive. Thus, we wanted to bring these positive experiences to a larger scale for the winter semester 2017/18. Hence, we designed the whole module "Software Development I" in this kind of setting. We will add a description and discussion of possible definitions of roles that can be used during design as well as throughout the teaching of lectures.

The module "Software Development I" is a compulsory module of our first semester Computer Science Bachelor curriculum. The module has 8 ECTS and consists of two times 1.5 hours lecture and 1.5 hours lab session per week. Students need to hand in and pass several lab exercises in order to be admitted to the final written exam.

2. Related Work

This work is guided by several influencing factors from various disciplines: pair programming, a well-known technique in the practice of software development – computational thinking, an overall goal in higher education – cognitive apprenticeship, which is a teaching method adopted from craftsmen training to cognitive processes and the distinction between cooperative teams and coach.

2.1 Pair Programming

A well-known and established approach in the domain of Software Engineering is pair programming as part of extreme programming (Beck, 2000). Pair programming denotes the cooperation of two people working together at one workstation on the same task and with a well-defined role distribution: the driver and the navigator. The driver is the actively programming person being in control of the keyboard – whereas the navigator monitors the activities with some distance. Several positive aspects of this model of cooperation are reported in (Williams, 2000):

- Pair Pressure: Increase of motivation as you don't want to let your partner down.
- Pair Thinking: Especially in the complex process of developing software, several different solution strategies are possible at any point in time. A pair of developers helps to permanently have them in mind and distill the currently best one of them.

- Pair Relaying: Pairs achieve solutions that a single one often had not reached. The main reason is that different points of view complement each other.
- Pair Reviews: As reviews are typically peer group discussion activities (Gilb & Graham, 1993) they are the natural tool for cooperative work, e.g. in our case during the preparation of lectures.

The approach of teaching in pairs or larger teams has already been described in several publications, like (Huber, 2000), (Burden, Heldal, & Adawi, 2012) and (Liebehenschel & Schäfer, 2017). All authors report positive influence on lecturers and students. Nevertheless, the papers are lacking a discussion of differentiated roles. We expect added value when taking different perspectives that are apparent to the students.

2.2 Computational Thinking

As we educate future computer scientists, we teach them specific professional competences. Computational thinking denotes a class of many skills and abilities, computer scientists need. These competences are independent from a specific implementation in a concrete programming language. Thus, this might be an interesting point where to differentiate the roles in a pair of lecturers.

The term of computational thinking is strongly influenced by Jeannette Wing. A general definition is: "Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science." (Wing, 2006)

However, computational thinking comprises a huge variety of processes and techniques. We focus on some that are relevant for our approach. All of these are based on (Wing, 2006):

- Evaluation of the aesthetics, and the design of a system, rather than just focusing on correctness and efficiency.
- Applying abstraction in order to find an appropriate representation of a problem or modelling all significant aspects.
- Using heuristic reasoning to come up with a solution.

2.3 Cognitive Apprenticeship

Most of the processes defined in computational thinking cannot be taught and done like following instructions in a recipe. Thus, students need to learn the processes experts use to handle complex tasks. In one characteristic, cognitive apprenticeship is defined as a learning-through-guided-experience with focus on cognitive processes (Collins, Brown, & Newman, 1988).

3. Patterns for Pairing of Roles

During the cooperative design and teaching of our class on software development we experimented with various roles and their pairing. The class of patterns we propose as a result of our experiences can be subdivided into two categories. Furthermore, we also found a couple of anti-patterns which we include as a third category. Pair teaching can occur in several constellations, as the lecturers can take different roles. From our perspective, each formation has its own intended use, strategies and thus advantages and disadvantages. It is important that each role is personalized by a lecturer (Andersson & Bendix, 2006).

From our point of view, the fundamental success factors that must be given are that both involved persons see each other as peer and must both be committed to the success of the module. Additionally, they should have similar didactical knowledge and teaching experience.

3.1. Patterns for the design and preparation phase of lectures

Didactic vs. professional. On the one hand side, it is necessary to simplify examples from professional tasks with respect to students' current level of knowledge. On the other hand, it is important not to design teaching examples that are contrary to professional practice. Here both peers need to discuss and find the best way to satisfy these requirements.

Analytically vs. holistically. This setting helps us to deal with the huge content of the curriculum. One peer focuses on the details and in-depth knowledge of a (sub)topic. Whereas the other keeps track of the overall curriculum. In our daily practice, this supported us to stick to our mantra: "everything we teach is correct – but we do not discuss every detail".

Reviewer vs. creative. After several design workshops, we came up with a process of defining requirements and teaching goals, which we build our lecture and lab on. In order to keep this process running, one of the peers constantly reviewed the process and investigated whether we still work on the requirements or not. The other peer was the creative one, who again and again came up with new ideas.

3.2 Implementation/teaching phase

As computer science is built on a solid theoretical basis like many other disciplines, it is necessary to show novice programmers the differences between the scientific concepts and the concrete implementation, as they need to master both later on.

Theorist vs. practitioner (Böttcher, Utesch, & Moore, 2009)**.** This is the role distribution for the classical situation where students learn a theory and apply it later. In this case one lecturer takes the role of the practitioner and the other acts as theorist. The practitioner

works on a real problem that is adjusted to the students' level of knowledge. The theorist teaches the theory and points to the theory students learned during the lecture/module.

Audience oriented vs computer focused. In case of software development and coding live in class it is very helpful, when one peer is coding and thus focused on the computer. Meanwhile, the second peer can keep an eye on the audience. Usually one is able to recognize whether students can follow the coding or not. If not, the peer in the audience oriented role should interrupt the coding in an appropriate moment and ask the peer or the students questions to clarify the decisions before they continue.

When considering computational thinking and cognitive apprenticeship, we came up with several more pairs, which are focusing on education in computer science, but might be transferrable to other disciplines.

Real world vs. software orientation. In their later profession, students need to deal with non-computer scientists as domain experts in order to collect requirements and transfer them to software. In this constellation, the process of translation from real-world abstractions to programming concepts is accompanied by one person focusing on real-word domain and one focusing on the software-based representation of that domain. Equivalent to the case above there exists the role of a practitioner which is in our case a software engineer. The pair is completed by a person that concentrates on the concrete real-world entities the software is intended to represent later.

Computational thinking vs implementation. Another important aspect in teaching computer science is to focus on processes represented in computational thinking and to consider it independent of a concrete implementation. In this case, both roles need to have a background in software development. One focuses on identifying the general concepts like categorization, repetition, or discrimination of different cases in order to solve a real-world problem. During this process, he or she demonstrates the application of key competences like logical or abstract thinking and highlights evaluation criteria. The other transfers the abstract concepts into appropriate constructs of a specific programming language. For both roles, it is important that they make their thinking tangible and highlight decision-points as expressed in the framework of cognitive apprenticeship.

Beside the technical and didactical roles, we unintentionally came up with three more roles.

Breaking point vs flow. In this setting the flow person continues with the new content. In contrast the other peer focusses on predetermined breaking points and intervenes if students did not come up with the predicted questions. Thus, the content flow stops here and the lecturers start a discussion with the students. At this point we often discuss software design decisions.

Checker vs speaker. Is similar to breaking point vs. flow. One peer focusses on the new content. The other one asks students questions about the necessary prerequisites. A good metaphor for this setting is "cruise control". One concentrates on a continuous pace, whereas the other checks for obstacles, like missing knowledge and thus adjusts the speed.

Focused vs overviewing. This last setting proved as a good tool to remind students that they have to perform follow-up course work. While one peer again concentrates on the new content, the other one analyses the questions immediately. If the question has already been discussed in the lecture, we reference to the specific location or just repeated it quickly. Otherwise we can either answer it or put a reminder to the white board.

While teaching it is not always necessary to take contrary or complementary positions. Pair thinking also happens if both peers act on the same level. This could help to identify misconceptions, interpretation of students' questions as they are not also clear. Another advantage is that the students have access to two perspectives and explanations. Furthermore, one can write down questions, tripping points, and much more. Following the lecture, both can reflect on the lecture in order to improve it for upcoming semester, to identify content that needs to be repeated or to plan next steps.

3.3 Beware of Anti-Patterns

Especially during the teaching phase, the pair needs to beware of typical anti-patterns. Often one does not intend to make this happen, but they might occur. A typical example is that one takes the role of a professor or lecturer and the other one acts as a student or even worse an airhead. This intellectual hierarchy lets the second person appear non-professional.

Another anti-pattern that occurred during the design phase, did arise from the job hierarchy. However, from personal experience, we immediately recognized this pattern and planned the lectures in a way that this would not show up. The supervisor or professor should not act with her or his employee e.g. PhD candidate as they are not on the same level.

4. Results of Evaluation

When starting this teaching approach, we had some criteria in mind we would like to improve. On the one hand, we wanted to enhance constructive-alignment (Biggs, 1996) throughout lecture, lab and exam. Furthermore, we wanted to arrange the topics to teach in a sequence where all prerequisites have already been covered at any time. Moreover, we wanted to create demand for the following concepts respectively.

After approximately six weeks we held a lecture where we reflected on the course with our students. Students were asked to think of obstacles that currently reduce their speed of learning and reflect on characteristics why software development is difficult for novices.

What we found interesting was that students were aware of thinking processes, computational thinking and the translation of real world problems into the software context. They were no professionals although they were already able to communicate the necessary steps they need to understand and accomplish in order to become professionals. We observed that the pairing of computational thinking and implementation has established at least the awareness that typical ways of thinking exist in computer science and that the translation of concepts into source code is a process separate from finding the right concept. An example statement is "I have not internalized the way to think yet" or "I'm still suffering with translating text into code".

At the end of each semester there is a university-wide lecture evaluation, which is a solid data basis we can use to evaluate our teaching. The questionnaire consists mainly of items, whose answer format are Likert scales with values ranging from 1 (fully disagree) to 5 (fully agree). We have data from previous courses taught in winter semester 2014. These can be compared with data collected with help of the same questionnaire in winter semester 2017 where we designed and taught a complete lecture in a pair.

The evaluation results we obtained for this make us feel optimistic. A statistically significant improvement can be seen in the questions listed in Table 1. We think that these represent our goal of increasing constructive-alignment and enhancing the topic arrangement.

Table 1. Results of an unpaired t-test, showing the significance level for improvement between lectures in the years 2014/15 (n=40) and 2017/18 (n=29).

Question	Average 2014	Average 2017	Signifi-cance
I can understand the content structure of the course.	3,1	4,2	≥ 99.9%
In my opinion, the two parts of the course complement each other well.	3,2	4,1	≥ 99.9%
I can clearly see what the learning goals of the course are.	3,4	4,1	99.8%
The applicability of the course content is clear to me.	4,1	4,6	99.8%
Complicated issues were explained to me in a comprehensible manner.	3,5	3,9	95%
In my opinion, the practical exercises are well adjusted to the taught content.	3,2	3,9	99.8%

5. Conclusion

From our lecturers' perspective, this approach is great. During the design phase, we could experience pair pressure as we worked much longer and harder than we would have been working on our own. If we try to quantify the additional effort we spent on the lecture, it is about 25% more preparation effort than we invest in our regular preparation. Pair relaying and pair thinking enhanced our work significantly, as we shared and discussed many ideas.

Even during the lecture, we experienced pair thinking and relaying for example when dealing with students' questions. Often students cannot formulate their questions precisely and have underlying misconceptions. We were much more effective in identifying the bottom line and correcting them immediately.

During the review of this semester, the question was raised, whether one can take the different roles on his or her own when disciplined enough. In our opinion it could be possible, but it is necessary to be aware of all the different roles and to take them one by one. Furthermore, we think it is necessary to concern and embed the roles already in the design phase.

Nevertheless, some constellations of roles just occurred during the phases and we did not plan them ahead. Thus, the approach added value to our teaching. Furthermore, effects like pair pressure, relaying and thinking can just occur in pairs. Hence, we recommend to try this approach and experience the great improvement if possible.

References

Andersson, R., & Bendix, L. (2006). Pair Teaching--an eXtreme Teaching Practice. *4: e Pedagogiska Inspirationskonferensen*.

Beck, K. (2000). *Extreme programming explained: embrace change.* addison-wesley professional.

Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher education, 32*, 347-364.

Böttcher, A., Utesch, M. C., & Moore, A. (2009). Erfahrungen mit Pair Teaching für Software Engineering: Kooperation von Hochschule und Industrie. *SEUH*, (pp. 5-15).

Burden, H., Heldal, R., & Adawi, T. (2012). Pair lecturing to model modelling and encourage active learning. *Proceedings of ALE*.

Collins, A., Brown, J. S., & Newman, S. E. (1988). Cognitive apprenticeship. *Thinking: The Journal of Philosophy for Children, 8*, 2-10.

Gilb, T., & Graham, D. (1993). *Software inspection.* (S. Finzi, Ed.) Addison-Wesley Longman Publishing Co., Inc.

Huber, B. (2000). Team-Teaching. Bilanz und Perspektiven. *Eine empirische Untersuchung im Kärntner Volksschulbereich/Integrationsklassen (Schuljahr 1998/99) zur Thematik/Problematik der Zusammenarbeit im Zweierteam. Frankfurt a. M.: Peter Lang*.

Liebehenschel, J., & Schäfer, J. (2017). Teamteaching-ein Fallbeispiel. *SEUH*, (pp. 91-99).

Williams, L. (2000). *The collaborative software process.* Ph.D. dissertation, The University of Utah.

Wing, J. M. (2006, March). Computational Thinking. *Commun. ACM, 49*, 33-35.