Peer-learning and Talents Exchange in Programming: Experiences and Challenges

Corinna Kröhn and Barbara Sabitzer
School Of Education, Johannes Kepler University Linz, Altenbergerstraße 69, Linz, Austria

Keywords: STEM, Programming, Peer-learning, Talents Exchange, COOL, Neurodidactics.

Abstract: "Programming is difficult" society tells us. However, this is only partly due to the subject itself. Very often, the tasks in software development are related to mathematical problems, which themselves are considered as difficult because they have no relation to the student's world of experience. Programming exams contribute to high drop-out rates and often students choose a different subject or even stop their university education. A possible gap could be closed by supporting beginners with the help of the COOL (cooperative open learning) concept that has already found its way into one course in the Business Informatics bachelor program at our university. In addition to this mandatory class, a peer-teaching course has been establishes. Students meet on a weekly basis to work together on the problems of software development based on the COOL concept. Two PhD candidates supervise the students, although they only act as tutors and do not provide any solutions. The approach described above is now being carried out for the third semester in a row and the drop-out rate has been reduced. Of course, further observation must be done to be able to draw concrete conclusions.

1 INTRODUCTION

Whether at school or university, learning to code can be very difficult. High drop-out rates and a major gender-gap verify these experiences. There already exist several different approaches to support beginners like tutoring systems or extra training courses but nonetheless there are still many students that do not pass exams.

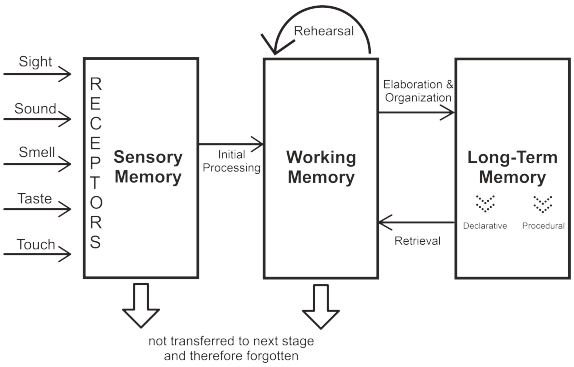


Figure 1: An information processing model (Wolfe, 2010) adapted by the authors.

One possible way to improve the learning outcome is to consider how the brain works and apply teaching methods investigated by Neurodidactics (Sabitzer et al., 2020). Neurodidactics combines Neuroscience and Didactics and presents a relatively young science that is yet not very established. Of course Neurodidactics cannot be transferred to higher education lessons one to one but it can give proposals for effective learning and teaching and can provide information of how the brain works (Sabitzer and Pasterk, 2015).

One of the key findings from Neurodidactics is that knowledge cannot be transferred but has to be newly created in each student's brain. That proposes that the learning process is an active one which also agrees with findings in pedagogy and constructivism. Our memory is a highly complex system (see figure 1) that also includes unconscious processes. The human long-term memory seems to be static but in fact is dynamic which concludes that each recall causes a re-encoding (Sabitzer, 2011).

Teaching methods suggested for effective learning and stimulating the long-term memory are the following (Sabitzer et al., 2013):

Discovery Learning: rests on finding solutions on your own and not getting instructions from teachers.

Social or Observational Learning: is based on so-called mirror neurons that are active when students observe or imitate others.

Kröhn, C. and Sabitzer, B.
Peer-learning and Talents Exchange in Programming: Experiences and Challenges.
DOI: 10.5220/0009472004660471
In *Proceedings of the 12th International Conference on Computer Supported Education (CSEDU 2020) - Volume 1*, pages 466-471
ISBN: 978-989-758-417-6

Learning by Teaching: Students slip into the role of a teacher and explain things to others. This method restarts the memory process and enhances retention.

Learning by Doing: inspires the student to be more active and is more effective than teacher-centered training.

COOL – Cooperative Open Learning: was initiated by a team of Austrian teachers and is described in one of the following sections.

This paper describes the evolution of a tutoring group to a peer-teaching class for programming beginners based on COOL concepts at our university. Sections 2 and 3 describe the underlying theoretical ideas, whereas section 4 concentrates on the detailed explanation of the development process of the different approaches to help students. It also presents experiences of students and teachers and concludes with ideas to further integrate the tested concepts in other university classes.

2 PEER-LEARNING AND ASSESSMENT

2.1 Definitions

Peer-learning is the use of teaching and learning strategies in which students learn with and from each other without any direct intervention of a teacher. Examples of peer-learning are student-led workshops, team projects, study groups or student-to-student learning partnerships. In *reciprocal peer-learning* students act as both teachers and learners (Boud et al., 1999).

In contrast to peer-learning *peer-teaching* also involves advanced students in class who act as tutors. The term "collaborative learning" is used to refer to peer-teaching as well (Boud et al., 1999).

In *Pair programming* two programmers work together at one computer on the same algorithm. One person is the "driver" whereas the other is the "navigator". The driver types at the computer and performs the physical input while the navigator observes and comments the role of the driver. Literature recommends to alternate roles during the software development process. Research has shown that this concept improves design quality, produces higher quality code, and improves communication among team members. But it has to be mentioned that pair programming only works if people are matched with a partner of comparable ability (Han et al., 2010).

2.2 Why Focusing on Peer-learning?

There are pragmatic as well as principled reasons for concentrating on peer-learning at university. One major problem of universities is lack of staff. As peer-learning and peer-teaching both concentrate on students, universities can easily increase the number of students in classes without more input from staff. Of course, it is also being argued that collective forms of learning may better suit women and minority groups than the traditional individualistic teaching methods because they concentrate on cooperation rather than competition (Boud et al., 1999). Considering the gender-gap, more and more of these teaching and learning concepts should be taken into examination.

Another important advantage is to foster students' practice in planning and teamwork because in peer-learning it is necessary to work together. It also engages reflection as well as exploration of ideas when the teacher is not immediately present. Moreover, students achieve more practice in the communication in the subject area compared to traditional teaching methods (Boud et al., 1999).

2.3 Assessment

When considering peer-learning, assessment needs to be taken into account. Assessment works as a form of academic currency and provides compensation for efforts of students. Of course it is also important not to misuse assessment as an instrument to get students to join peer-learning activities (Boud et al., 1999).

Assessing peer-learning has to be done with caution. The teachers will not be able to assess the outcome of peer-learning groups directly as it restricts the benefits of peer-learning itself. But there is a tradition of individual and competitive assessment in most universities which implies competition against others, rather than cooperation and collaboration is seen as cheating (Kohn, 1992).

Boud et al. recommend the following forms of assessment when peer-learning or peer-tutoring is involved (Boud et al., 1999):

Group Assessment: If teamwork and collaborative working is valued then each member of the group should receive the same grade. To prevent freeloading, the group assessment could also be the sum of each member's assessment.

Peer Feedback and Self-assessment: Switching outcomes of given tasks and grading classmates (or oneself) is a common approach

Assessment of Process: shifting the main focus from the learning outcomes towards the learning process

Negotiated Assessment: all parties involved agree on the assessment process

Use of Cumulative Rather than Weighted Assessment: If the weighting of one element of the course is very low, students tend to ignore it or put little effort in it. Peer-learning is often one of those aspects with little weight. But if it is only possible to pass the course by completing all of the tasks, motivation could rise to attain positive results in the smaller tasks as well.

3 COOL CONCEPTS: COOL INFORMATICS AND COOL PROGRAMMING

COOL is an acronym for *Cooperative Open Learning*, *Computer-(science)-supported Open Learning*, as well as *cool* with the meaning of interesting or "in" (Sabitzer, 2014). The COOL concept was developed by Austrian teachers in 1996 and then extended to *COOL Informatics* which was firstly presented in the habilitation thesis of one of the authors (Sabitzer, 2014). The four main principles are *Discovery*, *Cooperation*, *Individuality*, and *Activity* (see figure 2). Each of those concepts is then related to various teaching and learning methods that rest upon Neurodidactics (Sabitzer and Pasterk, 2015).

COOL Programming concentrates on these described neurodidactical concepts and was installed in one of the beginners programming classes in our university's Business Informatics bachelor program. The lessons as well as tasks were designed based upon the following findings:

1. New content is always built on existing knowledge and learning occurs through association (Sousa, 2009) (Spitzer, 2006).
2. Knowledge cannot be transferred but must be constructed in each student's brain (Roth, 2004).
3. Learning occurs through imitation (Sousa, 2009).
4. The brain recognizes and produces patterns, categories, and rules itself (Spitzer, 2006).
5. The instruction method impacts the retention of new information (Sousa, 2009).
6. Double-coded is double-saved (multimedia effect) (Roth, 2004).

Each of the 90-minutes-lessons in the designed course was divided into three parts: *Questioning* (10-15 minutes), *Discovering* (10-15 minutes), and *Pair Programming* (60-70 minutes). More precisely, the students started with an open round of questions that they could ask their professor. The next phase was dedicated to extract structures, rules, and other essential elements from Java Code by using reading corners, sample solutions, puzzles (see figure 3), or step-by-step examples (see figure 4). Afterwards students worked together on various programming tasks, following the rules of pair-programming (Sabitzer et al., 2020).

	Neurodidactical principles	
	Teaching and learning methods	***Neurodidactical basis***
1. Discovery	Solution-based learning Observational learning Step-by-step instructions/tasks Video tutorials Hands-on, Minds-on Learning with all senses	Pattern recognition Mirror neurons Individual learning rhythm modality/multimedia effect
2. Cooperation	Team and group work Peer tutoring and peer teaching Pair programming Cross-curricular learning Project-based learning	Recall = re-storage in long-term memory Integrating individual needs, talents and competences
3. Individuality	Competence-based learning Questioning Self-organized learning with compulsory and optional tasks	Connecting new information to previous knowledge Considering individual interests, needs, tasks, methods and learning rhythm
4. Activity	Hands-on, Minds-on Learning by doing Learning by animation and simulation Learning by playing and designing games (creative learning)	Knowledge must be newly created (constructed) by each learner Learning is an active process

Figure 2: Brain-based teaching concepts (Sabitzer, 2014) adapted by the authors.

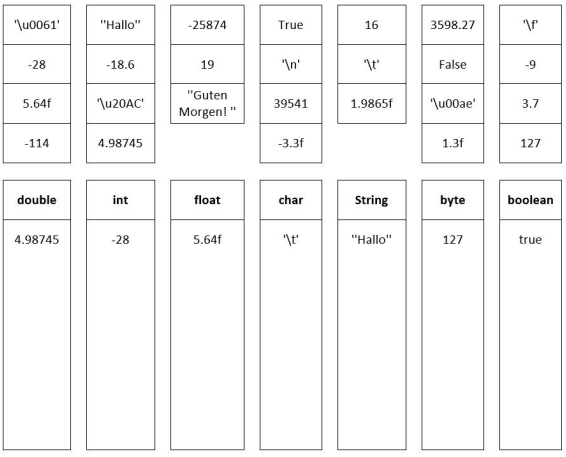
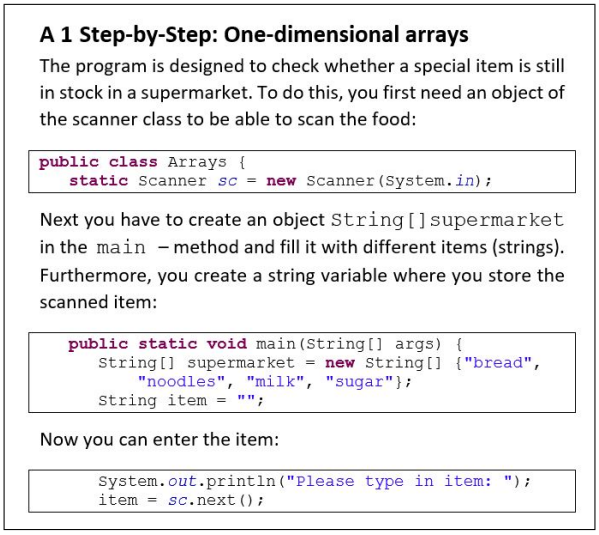


Figure 3: Datatype puzzle (Sabitzer et al., 2019).



A 1 Step-by-Step: One-dimensional arrays

The program is designed to check whether a special item is still in stock in a supermarket. To do this, you first need an object of the scanner class to be able to scan the food:

```
public class Arrays {
   static Scanner sc = new Scanner(System.in);
```

Next you have to create an object `String[]supermarket` in the `main` – method and fill it with different items (strings). Furthermore, you create a string variable where you store the scanned item:

```
   public static void main(String[] args) {
      String[] supermarket = new String[] {"bread",
          "noodles", "milk", "sugar"};
      String item = "";
```

Now you can enter the item:

```
      System.out.println("Please type in item: ");
      item = sc.next();
```

Figure 4: Excerpt from a step-by-step programming example (Sabitzer et al., 2019).

4 DEVELOPMENT OF A PEER-LEARNING/ PEER-TEACHING GROUP

Even though one of the beginners programming classes switched to the concepts of *COOL Programming*, there were many other courses that required support for students who had a hard time starting to code. More and more students created learning and tutoring groups on their own but they lacked locations, a fixed time, and experienced tutors. Therefore, students came to our department and asked if we could help them. Consequently, in winter term of 2018 we created a peer-learning group that met each week in the conference room of our department to discuss the upcoming programming tasks. One student who already worked as a software developer in different companies supported them as tutor. They met on a regular basis but the number of students varied depending on how hard the task was or if the exam was close. Accordingly, some of the students had to alter programming partners and sometimes there was only one student and the tutor present.

In summer term 2019 the first official peer-learning class was installed. Students could take that class voluntarily but still get ECTS for it. In that semester there were only 8 students who enrolled for class. Students were expected to actively participate and attendance was mandatory. The group split up on their own in various smaller groups – most of the time pairs – to work together on the given programming tasks. Two PhD-students acted as tutors but did not solve the coding tasks step-by-step. The problem with this class was that the students attended three different mandatory programming courses at university. One was the beginners class from the Computer Science bachelor program, the second one was the class of Business Informatics described in section 3 that is based on *COOL Informatics*, and the third group consisted of students from an advanced software development class. The problem occurring was obvious: whether the students nor the teachers could answer all the upcoming questions with such a variety of given tasks. But still the survey at the end of the course showed that all students approved and wanted the class to be continued. Fortunately, each one of them passed the final programming exam.

The following winter term the class was restricted to students who were enrolled in either the beginners programming class of the Computer Science or the Business Informatics bachelor program. The system also changed from peer-learning to peer-teaching, as two advanced students were involved in the same group. At the beginning 23 students attended the class, whereas only 14 completed it due to lack of time or interest. The group was split into three smaller study groups: one group consisted only of Computer Science students, the other two of Business Informatics students. Each of the two Business Informatics groups was assisted by one of the advanced students. Due to the lack of one more experienced student the group of students from Computer Science had no designated external tutor. Of course the two PhD candidates who oversaw the class provided help to each of the groups but still the main focus was on the peers. As already described in section 2.3, assessment of such peer-learning or peer-teaching classes is hard. As all students of the course participated actively in class, the teachers decided on assessing them according to their attendance. The end results of all the final exams are not published yet and therefore questioning students about this class has to await the outcome but we already know that at least half of the group passed.

5 EXPERIENCES OF STUDENTS AND TEACHERS

The main motivation for changing the ongoing system of programming education at our university was the close contact to students. At least once a semester the professor (and one of the authors) officially opens her office for first semesters and talks about their problems and wishes. Students as well as teachers expressed their experiences towards the implemented courses:

5.1 Report of a Female Student

"I started the studies of Computer Science with no prior programming experiences. As the introductory software-engineering course started, I soon found out that I was (1) one of the few students with no prior programming experiences and (2) one of the very few girls, which was quite intimidating. Because of the high demands and rapid pace, students dropped out week per week. In my opinion, these are the main reasons for the high drop-out rate:

Collaboration: Students had to complete weekly assignments independently. Collaboration was not allowed and in case of similar codes, the homework of all the students involved did not count. However, especially beginners benefit from collaboration and improve their skills by learning from and talking to each other. It could be argued, that students can indeed collaborate, they only have to develop different algorithms. Nevertheless, for students who are also not familiar with algorithmic thinking, this can be a huge challenge. And in reality, aren't software project usually planned and realized in teams?

Pace and Demands: As a programming beginner, I needed to dedicate the major part of my time to this course to be able to keep up the speed. Besides my job and other courses, this meant very long nights and long weekends. Furthermore, students had to positively complete 80% of the assignments and at the same time, reach a certain amount of points. These demands are very hard to achieve and put a high pressure on the students. Fortunately, this was the first semester, where a weekly peer-learning group was offered for CS students of the teacher training program. In these weekly meetings, we worked on our own algorithms, but in a very supportive environment. In other words, in case we were stuck, there was a tutor who led us to the right direction and gave us very helpful advice. Luckily, in my case it was worth the effort and I passed the course.

However, success very much depends on external support. Without professional support, like we had in the peer-learning group, I would not have been able to complete the course, which probably would also have been the end of my computer science career."

5.2 Report of a Teacher of the Peer-learning Class

"As a teacher of the new peer-tutoring course, I learned software engineering the classical way and was challenged with difficult tasks, an extremely high drop-out rate and hours of searching through the Internet for answers. Teaching this course was a complete new role for me. As a teacher I usually give answers to questions. In this course I had to help the students to find the answers on their own. This limited way to support the students worked better than expected. It showed that a lot of difficulties were the results of theoretical problems like "how does a list work". Furthermore, students started to help each other, since they worked on the same task, the problems were quite similar. The course was offered to students of two different Computer Science classes, the basic one and the advanced. During the course I realised that it is very challenging to stay up to date with all tasks and to switch during the lesson in answering questions to the basic and the advanced tasks. Altogether, the new way in teaching showed me, that students can help their fellow students in a very effective way and they have a lot of patience in helping each other."

6 CONCLUSION AND OUTLOOK

In this paper we have presented the journey from a tutoring group to a peer-teaching course in programming. We had high drop-out rates in our programming classes for beginners at our university. More and more students created learning groups on their own but they lacked experienced tutors. At first attempt, we installed a peer tutoring group that met weekly. In summer term 2019 the first peer-learning class based on *COOL Informatics* and *COOL Programming* was created that could be visited voluntarily. After two semesters, we can already identify improvement in terms of drop-out rates, exam attendance, and homework submissions.

We have identified the need for more collaboration than just working alone on tasks therefore competition has to be decreased. One remaining question is the assessment of the programming tasks. If students work in pairs or in groups on one project they will hand in similar or even equal solutions. How can a tutor distinguish between group work and cheating?

In the upcoming months, we will concentrate on evaluating the survey and the exam grades to further investigate the needs of programming beginners. We also need to collect more data about the background of our students, as their personal factors influenced their learning outcomes. We do not have enough information to answer questions like:

1. How do prior programming knowledge and drop-out likelihood correlate?

2. How does the type of high-school-diploma and drop-out rate correlate?
3. Does peer-learning or peer-teaching reduce the gender-gap?

Summarizing, we want to expand our concepts to advanced programming courses and provide more than one peer-teaching class.

REFERENCES

Boud, D., Cohen, R., and Sampson, J. (1999). Peer learning and assessment. *Assessment and Evaluation in Higher Education*, 24(4):413–426.

Han, K. W., Lee, E. K., and Lee, Y. J. (2010). The Impact of a Peer-Learning Agent Based on Pair Programming in a Programming Course. *IEEE Transactions on Education*, 53(2):318–327.

Kohn, A. (1992). *No Contest: The Case Against Competition*. Houghton Mifflin, Boston.

Roth, G. (2004). Warum sind lehren und lernen so schwierig? *Zeitschrift für Pädagogik*, 50.

Sabitzer, B. (2011). Neurodidactics – a new stimulus in ict.

Sabitzer, B. (2014). *A Neurodidactical Approach to Cooperative and Cross-Curricular Open Learning: COOL Informatics*. Habilitation, Alpen-Adria-University Klagenfurt.

Sabitzer, B., Groher, I., Sametinger, J., and Demarle-Meusel, H. (2020). Cool programming – improving introductory programming education through cooperative open learning. *ICEIT 2020, Oxford, UK*.

Sabitzer, B. and Pasterk, S. (2015). Brain-Based Programming Continued: Effective Teaching in Programming Courses. In *Proceedings – Frontiers in Education Conference, FIE*, volume 2015-February. Institute of Electrical and Electronics Engineers Inc.

Sabitzer, B., Pasterk, S., and Elsenbaumer, S. (2013). Brain-based teaching in computer science: Neurodidactical proposals for effective teaching. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, page 197–198, New York, NY, USA. Association for Computing Machinery.

Sabitzer, B., Spieß-Knafl, S., Pasterk, S., and Kröhn, C. (2019). *Entdecken Sie Java! Programmieren lernen und üben mit Musterlösungen*. Linz, Austria.

Sousa, D. A. (2009). *How The Gifted Brain Learns*. Corwin Press.

Spitzer, M. (2006). *Lernen – Gehirnforschung und die Schule des Lebens*. Springer Spektrum.

Wolfe, P. (2010). *Brain Matters: Translating Research into Classroom Practice*. Assn for supervision & curricu.