Applying Creativity in CS High School Education – Criteria, Teaching Example and Evaluation

Ralf Romeike
University of Potsdam
Department of Computer Science
A.-Bebel-Str. 89
14482 Potsdam, Germany
romeike@cs.uni-potsdam.de

Abstract

This paper describes an innovative method for teaching computer science in general high school education, illustrated with the example of introductory programming. Analyzing the literature in CS education research we found that creativity is rarely regarded, especially in high school education; although a few authors describe promising results from applying creativity. We designed and applied a framework for designing creative CS lessons based on a set of creativity criteria. The conducted teaching unit on introductory programming fulfilled the expectations: the students learned with high motivation and interest, the learning objectives were met and the students' picture of CS improved

Keywords: Programming, creativity, learning, teaching, motivation, high school computer science

1 Introduction

Computer science nowadays has taken an important position in German high school education and is represented as a mandatory or elective subject in almost all secondary schools[1]. The role of the subject is not to educate young computer scientists or programmers, but to provide the students with a positive attitude towards IT systems and a confident, responsible use of IT in the information society, and to allow the students an insight into the science itself. Even though the students arrive being more and more familiar with computers and with a general positive attitude towards them, CS has to deal with problems: low motivation, decreasing interest in the 'core' fields of CS such as programming, low grades, low participation of female students and the transfer of a wrong image of CS in schools. This in continuation has an additional impact on CS studies at university: students often enter with a wrong perception of CS and drop out early (Romeike & Schwill 2006). These problems stand in sharp contrast to some out-of-classroom observations where students and professionals spend a large amount of their free time dealing with programming or other aspects of CS. A key factor for engaging with programming seems to be creativity. In a study about the motivation of open source programmers, creativity-related factors were found to be the most pervasive drivers (Lakhani & Wolf 2005). In an interview with an outstanding motivated student of CS, creativity was also named as the most important factor for engaging in programming (Romeike 2006). The school subject of CS, as we see it, is strongly connected with creativity and can make use of it in manifold ways. Being creative fosters motivation and interest in the field, the subject of CS offers a fertile ground for creativity as the concepts and tools are well understandable and structured, and the omnipresent IT is beneficial for creativity (Romeike 2007c, Shneiderman 2000, Clements 1995, Thomas et al. 2002). One possible way that creativity can be applied in the classroom is described in this paper. After defining creativity and the consideration of it in previous research in CS education, we describe criteria for designing creative CS lessons. Based on these criteria a creativity framework is presented and applied in a lesson example for introductory programming, which was performed and evaluated in a German high school. The evaluation and the results are discussed.

2 Creativity

The term creativity is used with different meanings and is discussed controversially in psychology. Common speech usually defines something as creative when it comes from the arts or is something extraordinary. But not only artists can be creative. Everyday life requires creativity – and so does CS. There is agreement in psychology that something is creative if it is new, original and useful. How can an educator expect new and original achievements from his students? Boden (1990) describes two aspects of creative achievements. Historical creativity (h-creativity) describes ideas that are novel and original in the sense that nobody has had them before. Something that is fundamentally novel to the individual Boden describes as psychologically creative (p-creativity). In an educational context the latter is more interesting and can be aimed for in the classroom. Thus the difference between an exceptionally creative person and a less creative person is not a special ability. It is

[1] In the German education system secondary education starts – depending on the federal state – with the 5th class (age 10) or 7th class (age 12). Students aiming for the Abitur, which is a pre-requisite for higher education, attend the secondary II finishing after 12th or 13th class. CS lessons are offered to students starting from 7th, 9th or 11th class.

based on a larger knowledge in a practical and applied form as well as on the will to acquire and use that knowledge. With that in mind, in this paper we call something creative if it leads to personal new, unique and useful ideas, solutions or insights (cp. Runco & Chand 1995, Kaufman & Sternberg 2007). As summarized by Fasko (2000), in the classroom creativity can enhance learning through improved motivation, alertness, curiosity, concentration and achievement.

3 Creativity in CS Education

Computer science, as computer scientists see it, is a creative field to work in (e.g. Leach 2005, Glass 2006). Hence it is astounding that creativity is rarely reflected in CS education research. Even today, a search of the keyword 'creativity' in the ACM Digital Library returns only a few papers related to education. These papers can generally be assigned to a few groups in the contexts of problem solving, problem finding, motivation, and improving lessons and ICT to support creative practice.

Scragg et al. (1994) argue that CS is a fundamentally creative endeavor. Students need to be encouraged to discover insights in the creative process of problem solving. Hill (1998) describes open-ended problem solving and design processes in technology education as creative processes that engage exploration. She suggests moving away from making models to making prototypes for real-life contexts. In contrast, popular concepts for the school subject of CS are focusing especially on making models and yet leave out their implementation (Hubwieser 2000).

Some authors call attention to the field of problem finding/posing/identifying, which involves creativity and is important in the field of computer science. In the lesson context it is not limited to finding completely new problems, but includes also reformulating given or existing ones (Lewis et al. 1998, Kaasbøll 1998). Sutinen and Tarhio (2001) suggest it is better to speak about problem management than problem solving, as computer experts need skills that include problem recognizing and formulating.

A case study on the use of game programming in CS education was performed by Long (2007). She found that "being able to solve problems on my own" and "to be able to be creative" were the most important factors influencing intrinsic motivation.

Gu and Tong (2004) found, in an empirical study, that in software development courses the students perceived architecture design and programming as creative and that these phases were preferred. For similar reasons some authors employ creativity as a factor for raising motivation and interest in CS lessons. This was done by, amongst others

- changes in the environment and encouraging creative, hands-on learning and exploration into the projects in a data structures and algorithms course (Lewandowski et al. 2005)
- letting students choose and process their own problems (Meisalo et al. 1997)
- allowing programming as personal creative expression (Peppler & Kafai 2005, Resnick 2002)
- presenting programming in an entertaining discovering way (Wilson 2004)

Resnick (2007) sees creative thinking skills as critical for success and satisfaction in today's society. He reports that in computer clubhouses a creative use of the computer and programming is learned by promoting to students a spiral cycle of imagining, creating, playing, sharing, reflecting, and back to imagining. This he describes to be ideally suited to the needs of the 21st century.

Some researchers report achieving a positive effect on students' performance by applying creativity techniques in CS courses (Epstein 2006) or using creative methods for teaching programming (Chaytor and Leung 2003).

Several authors in CS education call for creativity, because

- Graduates in CS are missing creativity and problem-solving skills (Mittermeir 2000)
- Creativity is underrepresented in the curriculum (Sweeney 2003)
- Women drop out because there is no room for individual creativity in CS courses (Guzdial & Soloway 2002)
- Creative abilities are seen as the highest form of literacy, including computer literacy (Van Dyke 1987)

Computers have been found to be a fertile tool for supporting creativity. Many articles address IT support for creative practice, however there are just a few related to computer science education in schools (e.g. Clements 1995).

In summary these works show a broad spectrum of examples where creativity was identified to be beneficial and where creativity was successfully applied for enhancing learning. It is therefore quite surprising that the opportunities offered by creativity are not more frequently applied in general computer science education. It seems promising to us to investigate what the application of creativity can do for high school CS education.

In an analysis of the relevant literature we investigated the application of and the possibility of creativity in published computer science lesson examples (Romeike 2007a). We found that creativity was rarely employed. However, the lessons analyzed offered chances to do so and could be extended to creative lessons when considering creativity factors. Apparently teachers even partly prefer non-creative students, as they are easier to handle in the classroom[2]. Such a teacher attitude encourages students to prefer familiar ways that seem

[2] "Anyhow I am afraid that students let their creativity play too much so that the results of this project would be of limited usefulness." (Janneck 2006).

safe and risk free but do not leave much space for creativity[3].

We consider the role of creativity in CS from two perspectives. First, we believe that creativity is essential to CS. Second, CS makes it easy to be creative. Keeping in mind the relevance of creativity for CS and its value for CS education may help educators to overcome some common problems they experience in the classroom.

Formulated creativity criteria will help teachers in planning lessons and regarding the creativity potential that CS offers.

4 Criteria for Creative CS Lessons[4]

To obtain a foundation for creative CS lessons, we set up a catalogue of criteria based on findings in the literature of psychology and education. These criteria can be used for designing and evaluating computer science lessons. They reflect and combine general pedagogical principles that are essential and beneficial for creative practices in CS education. In addition they consider typical tasks and principles that are common in CS.

4.1 Requirements for the Subject

Relevance. We define the subject of a lesson as the topic that is used for illustrating the teaching matter. As creativity requires personal involvement it needs to be appealing and thus relevant to the students, or needs to be presented that way.

Problem management or creation of a product. Gardner (1993) classified five types of creative activities. Two of those are typical for CS and should be aimed for in creative lesson phases: problem solving and the creation of a product. This includes the implementation of a model, not just the finding of a theoretical solution.

4.2 Requirements for Tasks

Subjective novelty. This important criterion for creativity is often overlooked by teachers who use tasks very similar to those that have been discussed in detail in the lesson. Even if it is unlikely that a student will come up with a general new solution or product, subjectively new (p-creative) ones should be aimed for.

Openness in possible results, approaches and solution methods. Creative processes are characterized by aspects of problem finding and creative problem solving, exploring and discovering. This is possible only if tasks allow several approaches to the problem, diverse solutions, and solutions that can differ in the degree of elaboration.

Application of concept knowledge. A solid foundation of knowledge is essential to creative practice. In a creative lesson phase, concept knowledge needs to be emphasized in contrast to product knowledge or factual knowledge.

Inspiration. A creative achievement is always preceded by a stimulus. Type, content, formulation or circumstances of a task and learning situation can provide such an initiation. For CS lessons this includes revealing to the students, for example, what a piece of software will be used for and which 'broader' problems it is supposed to solve.

4.3 Student-oriented Requirements

Identification. Creative practice may get a person enthused, getting him or her deeply involved with a task, and may trigger a flow-condition. Fundamental for this is that the person can identify himself with the task. For CS lessons this implies that the content needs to be (or can become) meaningful to the student, e.g. by taking over responsibility and/or later presentation.

Originality. Every student is a unique individual with his or her own ideas, visions and preferences. Obeying this criterion means allowing space for a student's originality demands, i.e. letting the student bring in a personal touch.

4.4 Requirements for the Teaching Environment

Experimenting. Being creative means to experiment with ideas, to explore the space of possibilities and to test solution possibilities. A tool used should provide meaningful feedback; for example, the compiler of a programming environment supports experimenting in CS lessons as it gives detailed feedback to the learner.

Freedom in time. Creativity is hard to realize under time pressure, as time is needed to gather, evaluate and realize ideas. Projects in CS lessons support this criterion.

Climate of diversity. Group pressure, early evaluation and expected perfection are known to oppress creativity. Instead the lesson should allow encouragement and inspiration among students. New ideas should be welcome and diverse solutions supported and presented.

Teacher as a coach. The teacher needs to diminish the leading role of transferring knowledge, correcting and assessing. Instead the teacher assists only where a problem cannot be solved by a student himself. He motivates and encourages the students.

5 Introduction to Programming by Applying Creativity Criteria

The question of how programming should be introduced is a central issue of computer science classes in schools over and over again. But universities as well as schools struggle to provide students with a smooth transition into the field of computer science. Often these introductory

[3] Taking risks is difficult for creative students because creativity is not always rewarded with good grades (Sternberg & Lubart 1991). Perhaps this is due to the negative attitudes teachers hold towards creative students, as evidenced by the findings of Westby and Dawson (1995).

[4] A detailed derivation and explanation of the criteria were performed in (Romeike 2007a).

courses and topics are found to be the cause for computer science being seen as hard, mechanistic or even uninteresting or discouraging (Bergin 2005, Curzon 1998, Mamone 1992, Rich et al. 2004, Tharp 1981, Feldgen 2003). We believe it does not need to be this way. The chance to develop software using a programming language can nicely demonstrate that computer systems can be shaped by the student in a motivating way.

For the realization of a lesson example which is motivating and encouraging for the students, and at the same time allows for learning about computer science close to the subject, the criteria for creative computer science lessons were regarded and applied. The lesson example was designed for introducing an 11th class of computer science in a German high school to programming. As a programming language and creativity supporting tool the visual programming language Scratch (Maloney 2004) was used. The application of the creativity criteria results in a creativity framework that was followed in the teaching unit and ensured that all of the criteria could be given due regard. The framework is described as follows, and illustrated by details of the lessons. The teaching unit in detail can be found at (Romeike 2007b). The educational objectives of the teaching unit are summarized in Figure 1.

5.1 The Creativity Framework

5.1.1 Motivation for New Concepts of Programming

Motivation is an essential part of teaching. Receiving students' attention and fostering motivation was supported by showing the use and relevance of the contents to the students and by choosing topics that are meaningful to them, e.g. animating their name or a story of their everyday life or imagination, and the development of games that can be played by them. Often new concepts were brought up by the students themselves after discovering and applying them in their projects before they were formally introduced.

5.1.2 Laying out the Fundamentals

The introduction of new content was done by applying a building block metaphor. Attributes and uses of the programming concepts in the Scratch programming language were discovered or explained. Beneficial for this view is the visual representation of CS concepts in Scratch as blocks that can be snapped together. In this way students learn an appropriate visual representation of the concepts and do not have to deal with syntax errors, as they are not possible. As a teaching method, the concepts were introduced either by the teacher, by work sheets or by student presentations.

5.1.3 Inspiration

It is essential for a creative lesson to provide an inspiration to the students, generally by showing an example program or brainstorming about possibilities. This allows the students to spark their creativity, to balance what they may want to achieve and what they can achieve with the concepts learned so far and what the programming language is capable of.

5.1.4 Challenging the Students

Challenging the students was done with open-ended tasks with variable solution complexity and independent working time for the students. The tasks assigned were basically pointing the students in a direction given a general framework of what to do. Thus the students had to solve a problem they needed to clarify for themselves up front ("What do I want to do?"). There was no single right solution that needed to be achieved (openness) and – as time allowed – the solution could be elaborated as wanted or as possible for the students. Tasks were, for instance, "Design a program that displays your name and animates the letters to interact with the mouse or keyboard!"

This way the students could get familiar with the concepts they had just learned, explore the programming environment, find solutions for their ideas, and implement and test them. The teacher would go around, encourage the students to explore the possibilities, and intervene only if asked or needed. Usually such a working period ended with the end of a lesson. This way those students who wanted to elaborate their work or to extend or modify their programs could continue to do so at home.

5.1.5 Presentation and Reflection

Finally the students uploaded their programs to the Scratch webpage and a few programs were presented to the rest of the class at the beginning of the next lesson. Presentation of the work included presentation and discussion of ideas, problems and strategies. If students discovered and applied new concepts in their program they explained them to the rest of the class.

At the end of the course every student was asked to develop his own game, with the only condition being that all of the learned concepts should be applied. The task resulted in a variety of computer games ranging from

- Basic understanding of programming
- Algorithms:
  - Characteristics (finiteness, clarity, feasibility, general validity)
  - What can be solved algorithmically?
- Basic concepts of programming:
  - Sequence
  - Loops
  - Decisions
  - Variables (local/global)
- Input and output of data
- Arithmetic operations and comparison operators
- Representation of algorithms as Scratch blocks
- Object, message, attributes, methods
- Reading and analyzing programs
- Modifying and extending of programs
- Designing, implementing and testing of programs
- Idea generation and problem management

Figure 1: Educational objectives

pong and memory to sport and shooting games.

5.2 Scratch

The visual programming (mini) language Scratch was originally designed for young students to develop 21st century skills (Maloney 2004). It allows creating animations, games and other programs by 'clicking together' programming constructs represented as building blocks. Nevertheless, due to its intuitive appearance and usability it is used in computer club houses, high schools and even in introductory programming college courses. We chose Scratch because it emphasizes the practical learning of fundamental CS concepts and at the same time supports the idea of fostering creativity in CS classes. Mini languages are said to provide an insight into programming and teach algorithmic thinking for general computer science in an intuitive, simple, but powerful way (Brusilovsky et al. 1997). Thus Scratch meets the needs for the intended purpose.

6 Evaluation

6.1 Method

The attempt to introduce programming was done in parallel in two courses of the school. Parallel to the author conducting a lesson as described above (A), the other course (B) was taught by an experienced teacher following a 'traditional'[5] problem-solving oriented approach. The problem-solving approach was performed by using the tool 'Robot Karol' and followed suggested learning tasks as provided with a schoolbook for CS education (Engelmann 2004).

Course A consisted of 21 students, aged 17, with 38% female students. Course B consisted of 23 students of the same age with 61% female students. The students' prior experience in computer science was comparable.

The accompanying evaluation was following two questions. First, if creativity is explicitly considered, what effect does this have on the students' motivation, interest, and picture of the school subject of CS? Second, what is the impact on the students' task understanding and achievement? As a research instrument for the first question, a questionnaire was used; for the second, the average grades of the students before and after the course, and a test following the course.

The questionnaire was structured the following way:

1. Scale-based responses to statements about computer science lessons in general, e.g. "CS lessons are fun/interesting/creative", "I participate / I am distracted", "I show results at home"
2. Questions about difficulty, amount of content and appropriateness of the last teaching unit
3. Appraisal of teaching techniques, methods and tools
4. Scale-based responses to statements about the topic, e.g. "I could discover new things", "I could concentrate", "I have the feeling I learned something"
5. Questions about the perceived learning outcome/ success of the individual and of the learning group

The questionnaire was answered by the students before and after the 4-week (11-hour) course.

The test following the course contained two sections:

1. Theoretical:
   - Definition and characteristics of algorithms
   - Describing concepts of programming and giving an example
2. Practical:
   - Explaining and optimizing two programs presented on paper
   - Implementing a program to a given problem
   - Implementing a program for a self-chosen task, applying all used concepts

6.2 Results

6.2.1 Motivation, Interest, Picture of CS

The picture students have of the school subject of computer science is forming their understanding of the science in general. Furthermore this is responsible for students' motivation and eventually builds the foundation for the question whether they will consider CS as a subject to study at university. Humbert (2003) investigated students' pictures of CS in his dissertation research. The subject was seen as the science of computers and of how to use computers. The chance of designing and shaping software systems was rarely reflected. This view did not change much after one year of CS lessons.

The creativity-teaching unit changed the students' picture of CS in many ways, as illustrated in Figure 2, which shows the change in responses between the start and end of the course. Fun and interest were raised considerably ("Computer science is fun" (71% → 93% agreement), "I regard the content of computer science as interesting" (29% → 93% agreement)). These factors have a major impact on the motivation of the students and can be greatly used for maintaining students' interest in CS. Programming was very motivating for the students – unlike many experiences in the classroom and in

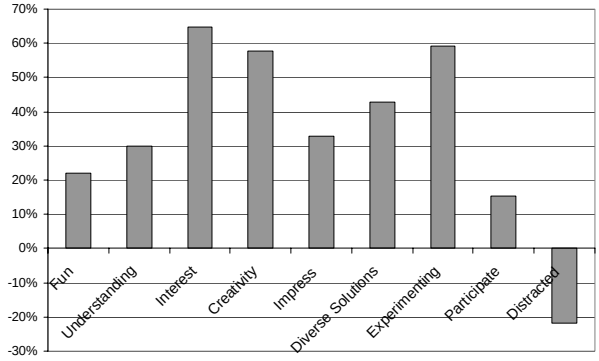

Figure 2: Agreement compared with before the lesson

[5] As a 'traditional' problem-solving approach we consider the way a majority of teachers introduce programming by assigning a sequence of convergent problem-solving tasks with increasing difficulty.

introductory programming courses at university, where programming is often a reason for failure.

Consistent with the teaching approach, a big change happened in the judgment of creativity. In response to whether they consider CS lessons as a place where they can be creative, 93% answered in the affirmative, compared with 36% before.

Computer science is generally a subject where several solutions are possible for a task and where experimentation is also involved in understanding difficult interrelations. Experimentation is playing an increasingly important role, e.g. for analyzing the behavior of software (Reed 2002). Often in school settings these aspects are not obvious to the students. This is especially true when teachers need to choose 'effective' ways of teaching, such as teacher-centered instruction with convergent problem-solving tasks to 'get through the stuff' in the shortest time. Here, too, the majority of students are not aware of these aspects prior to the creative lessons. Afterwards most agreed that 'in CS diverse solutions and solution methods are possible for a single task' (43% → 86%) and 'in CS lessons you experiment a lot' (14% → 73%). This increase is especially interesting as the students in the previous teaching unit were actually investigating, designing and experimenting with databases. Apparently, genuine designing and changing a computer system by programming in a creative way better meets the students' understanding of 'experimenting' then investigating the characteristics of a 'fixed' system such as a database with convergent problem-solving tasks.

The creation of presentable products (programs) can also have an effect on how students' friends and family consider CS lessons. From almost none at the beginning (7%), 40% of the students agreed that they could impress family or friends with results from the lessons.

Summarizing, the students' picture of computer science lessons changed positively. CS lessons in German schools – and in other parts of the world as well – often differ a lot from real CS. They are perceived as the subject where you learn how to use the computer, how to use Word and Excel and how to use the internet. The students are now more aware of the reality of CS, as a subject that involves designing and changing computer systems, experimenting, and finding a good solution where many solutions are possible. As high school also needs to prepare students for university, these factors need to be considered. Furthermore the 'technical' reputation of CS has caused a gender bias, with girls in particular not showing much interest. With the creative way of looking into CS the girls' interest was also raised and they enjoyed the tasks. Considering the answers separated by gender, it is apparent that the girls mostly answered comparably with the boys.

6.2.2 Understanding, Achievement

The perceived learning outcome was stated as high. The answers are in accord throughout the class and stand in contrast to the perceived learning outcome of the previous teaching unit. There the answers are more diverse, and half of them include reports of problems. These views are also reflected in the perceived learning outcome judgment for the course: 87% believe that all or most of the students in the class understood the programming content well or very well. In the previous teaching unit, the majority of the students checked answers reflecting problems among their classmates. This is interesting, as their own reported learning success in 'databases' was generally better than the perceived learning success of the class. Even if at least half of the students understood the matter, the class was learning in a climate of problems and 'not-understanding'. In the topic of programming, due to the many ways of presenting the students' results and achievements, the classroom climate was a more positive one. This in turn could motivate the students' persistence and desire to understand when they encountered problems.

The effective learning outcomes were measured by a test concluding the teaching unit. The test was successfully accomplished by 94% of the participating students. The course average of the test is 0.2 grades better than the class average in the first half of the semester and about one grade better than the average in the test concluding the previous teaching unit. The grades can be separated into two groups: 69% received grades 'good' (2) or 'very good' (1), 25% 'satisfactory' (3) or 'fair' (4). Considering the grades according to gender, all girls received grades of 2 and better, while the boys' grades are equally distributed through the scale. Keeping in mind the problems many CS and programming courses have with female students' achievements, this seems to be an encouraging outcome.

6.2.3 Additional Results

6.2.3.1 Questions about the lessons

Unfortunately students are generally not used to working independently in the classroom. Even though pedagogy has for decades suggested different teaching methods, the most prominent teaching style in German schools is still teacher-centered classroom instruction (Meyer 2003). Applying a new teaching method can be challenging and troublesome for all participants, as the students may not be sure about what they are expected to do, and cannot follow a common familiar schema. In this connection it is interesting to investigate the perception and the attitude of the students towards the teaching methods. Students' answers about the teaching methods, tasks, and lessons are presented in figure 3.

All students considered the presentation of the learning content as understandable. This is a desired result, but still surprising, for two reasons. First, the teacher did not put much effort into explaining and concretizing the concepts of programming. More or less anything that was learned was done so by actively engaging in programming. Content was presented briefly or collected together and applied right away. This approach is supported by the constructionist learning theory (Papert 1980) that encourages learning by design and engaging students in personal meaningful tasks. Second, programming is known for being a difficult matter to

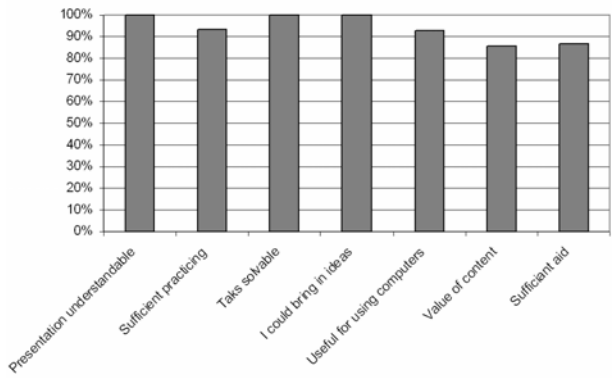

Figure 3: Appraisal of teaching methods

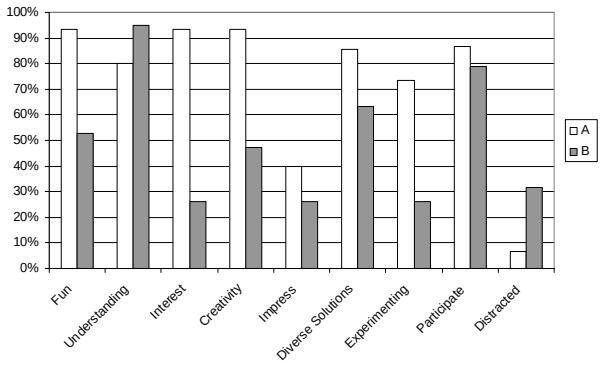

Figure 4: Agreement compared with control group

teach in schools (many teachers in high schools struggle for months and even years to teach the basic concepts of programming). The students here considered the degree of difficulty as appropriate and perceived their learning outcome as appropriate or a lot. Asked about how hard the subject matter was, half of the students responded 'sometimes easy, sometimes hard', the other half 'generally easy'.

Practice time was perceived as adequate, even if the circumstances put quite some time pressure on the students. Students appreciated that they could bring their own ideas. All students considered the tasks as solvable – even if there was no 'right' solution that they needed to find for the tasks.

The role of the teacher is reflected in the answers to the question about where the students learned most: by working at projects (60%) and dealing with the tasks (60%) in contrast to explanations of the teacher (13%). Again, it is somewhat surprising that programming, at least at this elementary level, can be learned so intuitively.

6.2.3.2 Questions about the topic

The answers about the topic are somewhat ambivalent. Today's students grow up surrounded by technology. Every student in the observed class has a personal computer at home. Nevertheless less then a third of the students stated that the topic was dealing with issues out of everyday life and only 43% of the students think they can use the learned knowledge in future. These numbers are even lower than the ratings for the previous teaching unit. In the classroom the relevance of programming concepts and the connection to everyday life have not been explicitly shown to the students by the teacher. Given the strength of real-life contexts as a major source of motivation, one would think that the students could hardly have been motivated for the lesson. Surprisingly, 80% stated that some of the issues had been very interesting to them, 87% state they learned something, and 73% said that they had fun with this topic. Obviously – without being motivated by the topic as being connected to everyday life – the tasks and the creative practice were motivating enough for the students to enjoy and learn.

6.2.4 Comparison with the control group

6.2.4.1 Comparing the questionnaires

Prior to the introduction to programming the curriculum of both courses consisted of the same topics in computer science[6]. The grades of the two courses prior to the observation were generally comparable.

Comparing the answers of the two courses about their picture of computer science prior to the introduction to programming the students answered very much alike. B had slightly more consent with the item 'fun' and a significantly higher consent with the item 'understanding'. The rest of the answers are comparable.

This changes tremendously when comparing the answers after the introduction to programming: fun rose by 22% in A but declined by 32% in B. Only half of the students of B considered CS as fun after the programming course. While programming had an enormous impact on the interest in CS of the students of A, in B 'interest' remained low for 75% of the students. Students of group B also agreed more on 'creativity' after learning about programming (+18%). This is an interesting fact, showing that even in the problem-solving approach creativity is needed, and this also becomes obvious to the learners. Similar findings are reported by Long (2007). Not all students in the problem-solving group saw that there are several ways of solving a given problem. The agreement with this item rose in B to 63%, while in A it rose to 86%. The agreements to the statements in comparison with the control group are illustrated in Figure 4.

6.2.4.2 Comparing the achievements

The course was started in both groups with the same learning objectives in mind. Unfortunately not all learning objectives could be achieved in group B. Variables are not implemented in the tool used in B. Also characteristics of algorithms were not considered by the teacher of group B due to a lack of time. Even so, it is interesting to compare the answers of the students as to how they perceived their achievements.

[6] We will refer to the course with a creative introduction to programming as A and the course following a problem-solving approach as B.

In the questionnaire both groups had to assess their learning success. While in A all students stated that it was appropriate or high, in B only two-thirds of the students did so. Nevertheless the grades of the following test were comparable. Both groups considered the difficulty of the lesson and the complexity similarly (appropriate or high).

Comparing the grades with those of the prior teaching unit, group A improved a lot while group B on average remained stable at the grades they had before. But splitting the grades by gender, the boys of group B improved their grades while the girls deteriorated. In contrast to that stand the achievements of group A, where the girls improved their grades considerably more then the boys. Before the introduction to programming, the level was equal for boys and girls in both groups.

6.3 Critical Reflection

There are two drawbacks to this study. First, the lessons were taught by different teachers. The teaching style, the teachers' personalities and the way the teacher gets along with the class can and will have an effect on the learning outcome and the students' motivation. On the other hand, as was shown by the questionnaire that the students completed before the observed lessons, motivation and achievement were equally high in both groups. But as the tasks and exercises used in the problem-solving approach have been taken from a school book that has been used by hundreds of teachers before to introduce programming, they seem to be quite typical for a course that introduces programming through problem solving.

Second, the groups had not only a different methodology but also different software tools. This is perhaps a key factor, and might be responsible for the rise in motivation and perceived creativity as well. Nevertheless, the bottom line stays the same: a creative introduction to programming is both possible and expedient. If the reason is the programming language used, it might be wise to consider creativity when choosing a programming language. As the creativity criteria fit well with many pedagogical implications, they should at least be considered. If the reason for the success of the teaching unit lies in the application of the creativity factors, it is even more strongly recommended that these factors be applied in other teaching settings. Besides, choosing a programming environment that addresses the students' interests will be helpful anyway. Scratch is obviously a candidate for that. We strongly believe that both – the application of creativity and the creativity support of Scratch – are responsible for the learning success. Hence we suggest that creativity be applied to introductory programming courses, regardless of the programming language used, but particularly if using Scratch.

Future research needs to address these questions in detail.

7 Conclusion

Analyzing the literature in CS education research we found that creativity is rarely regarded, especially in high school education. Promising results from applying creativity are described by a few authors. We applied a framework for designing creative CS lessons based on a set of creativity criteria. The conducted teaching unit in introductory programming fulfilled the expectations: the students enjoyed the lessons, the learning objectives were met and the students' picture of CS improved. This is in agreement with studies where contextualization, personalization, and choice produced dramatic increases in students' motivation, in their depth of engagement in learning, in the amount they learned in a fixed time period, and in their perceived competence and levels of aspiration (Cordova and Lepper 1996).

The students' efforts were concentrated and intrinsically motivated. Even when a lesson was over, many of them wanted to remain in the classroom in order to continue working on their project. The presentation and dissemination of the students' results led to increased motivation in the next lesson. Even another course at the school was getting to know the results of this course as many students soon started to play online the games that they had created.

Female students performed very well in the course and could engage in tasks they enjoyed. Our initial impression of a few female students was that they were likely to get distracted by designing the look of the program and less interested in focusing on the functionality; for example, that they would be more interested in making little films than interactive programs. But as soon as some programs were presented, challenged by the creative classroom climate, they caught up and applied the newly learned concepts as well. Especially contrasting the learning results to the control group it becomes obvious that female students performed better in the creative teaching setting.

Interestingly we found that sometimes it is not easy to change a firm stereotype of CS, as illustrated by the following example. After the lessons one student seemed quite unhappy and uncertain. When she was asked about what was bothering her she answered that she found the lessons a bit strange and asked when we would start 'real' CS. The experienced lessons in her opinion had been so "c-r-e-a-t-i-v-e". In her opinion, other subjects are supposed to be creative, but not CS. Asked whether she understood the content and enjoyed the lessons, she said that she had. The lessons just had not met her pre-conceived notion of CS.

After these experiences we believe that creativity can and should be applied in the long run in programming courses and can possibly serve as a principle in other fields of CS as well. We would also like to encourage educators of CS to apply creativity at the university level. The benefits of increased motivation and interest for all students, but especially for women, are worth trying and come with a low risk. Since the beginning CS has been a creative endeavor (Scragg et al. 1994, Saunders 2005). Let us show our students what this can mean.

More and more learning environments are developed that allow a smooth – and creative – introduction to programming. The full potential that lies in these powerful tools can be better tapped with regard to creativity.

In a time where standardized tests are becoming more and more common, the call for more attention to something that is seen as ineffective as creativity may seem a little odd. Nevertheless, the positive outcomes as described encourage us to further investigate how creativity can be fostered, at the same time enhancing learning in computer science education.

8 References

Bergin, S. and Reilly, R. (2005): The Influence of Motivation and Comfort-Level on Learning to Program. In *Proc. of PPIG 17.* University of Sussex, Brighton UK, 293-304.

Boden, M. A. (1990): *The creative mind: myths & mechanisms.* Basic Books, London.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A. and Miller, P. (1997): Mini-languages: a way to learn programming principles. *Education and Information Technologies,* **2,** 65-83.

Chaytor, L. and Leung, S. (2003): How to creatively communicate Microsoft.NET technologies in the IT curriculum. In *Proc. of the 4th conference on Information technology curriculum*, Lafayette, Indiana, USA, 168-173, ACM Press.

Clements, D. H. (1995): Teaching Creativity with Computers. *Educational Psychology Review*, **7**(2): 141-161.

Cordova, D. and Lepper, M. (1996): Intrinsic Motivation and the Process of Learning: Beneficial Effects of Contextualization, Personalization, and Choice. *Journal of Educational Psychology,* **88**(4): 715-730.

Curzon, P. and Rix, J. (1998): Why do Students take Programming Modules? In *Proc. of the 6th annual conference on the teaching and computing and the 3rd annual conference on integrating technology into CSE: Changing the delivery of Computer Science Education. ITICSE '98.* Dublin, Ireland, 59-63.

Engelmann, L. (2004): *Informatische Grundbildung,* Paetec, Altenburg.

Epstein, R. G. (2006): An ethics and security course for students in computer science and information technology. In *Proc. of the 37th SIGCSE technical symposium on Computer science education*, Houston, Texas, USA, 535-537, ACM Press.

Fasko, D. (2000): Education and creativity. *Creativity Research Journal,* **13**(3-4): 317-327.

Feldgen, M. and Clua, O. (2003): New motivations are required for freshman introductory programming. In *Proc. of the 33rd ASSE/IEEE Frontiers in Education Conference.* Boulder, USA, **1**: T3C-T24.

Gardner, H. (1993): *Creating minds: an anatomy of creativity seen through the lives of Freud, Einstein, Picasso, Stravinsky, Eliot, Graham, and Gandhi,* BasicBooks, New York.

Glass, R. L. (2006): *Software creativity 2.0,* developer .* Books, Atlanta.

Gu, M. and Tong, X. (2004): Towards Hypotheses on Creativity in Software Development. *Lecture Notes in Computer Science,* **3009**: 47-61.

Guzdial, M. and Soloway, E. (2002): Teaching the Nintendo generation to program. *Communications. of the ACM,* **45**(4): 17-21.

Hill, A. M. (1998): Problem solving in real-life contexts: An alternative for design in technology education. *International Journal of Technology and Design Education,* **5**(3)**,** 1-18.

Hubwieser, P. (2000): *Didaktik der Informatik: Grundlagen, Konzepte, Beispiele.* Springer, Berlin.

Humbert, L. (2003): *Zur wissenschaftlichen Fundierung der Schulinformatik,* Pad-Verl., Witten.

Janneck, M. (2006): Partizipative Systementwicklung im Informatikunterricht. *LOG IN* 138/139: 60-66.

Kaasbøll, J. J. (1998): Teaching critical thinking and problem defining skills. *Education and information Technologies*, **3**(2): 101-117.

Kaufman, J. C. and Sternberg, R. J. (2007): Creativity. *Change: The Magazine of Higher Learning,* **39**(4): 55-60.

Lakhani, K. and Wolf, R. (2005): Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects. In *Perspectives on Free and Open Source Software*. 3-22. J. Feller, B. F., S. Hissam, and K. R. Lakhani (eds). MIT Press.

Leach, R. J., Ayers, Caprice A. (2005): The Psychology of Invention in Computer Science. In *Proc. of 17th Annual Workshop of the PPIG.* University of Sussex, Brighton UK.

Lewandowski, G., Johnson, E. and Goldweber, M. (2005): Fostering a Creative Interest in Computer Science. In *Proc. of SIGCSE '05.* St. Louis, MO.

Lewis, T., Petrina, S. and Hile, A. M. (1998): Problem Posing-Adding a Creative Increment to Technological Problem Solving. *Journal of Industrial Teacher Education,* **36**(1).

Long, J. (2007): Just For Fun: Using Programming Games in Software Programming Training and Education - A Field Study of IBM Robocode Community. *Journal of Information Technology Education,* **6**: 279-290.

Maloney, B., Kafai, Rusk, Silverman, Resnick (2004): Scratch: A Sneak Preview. *IEEE Computer Society,* 104 - 109.

Mamone, S. (1992): Empirical Study of Motivation in an Entry Level Programming Course. *ACM SIGPLAN Notices,* **27**(3): 54-60.

Meisalo, V., Sutinen, E. and Tarhio, J. (1997): CLAP: teaching data structures in a creative way. In *Proc. of the 2nd conference on Integrating technology into computer science education.* Uppsala, Sweden, 117-119.

Meyer, H. (2003): *Unterrichtsmethoden II: Praxisband*, Cornelsen Scriptor, Berlin.

Mittermeir, R. (2000): Informatik-Unterricht: Bastel-Unterricht, eine intellektuelle Herausforderung oder "Preparation for the information-age". *Medienimpulse,* **9/33,** 4 – 11.

Papert, S. (1980): *Mindstorms : children, computers, and powerful ideas,* Basic Books, New York.

Peppler, K. and Kafai, Y. (2005): Creative Coding: Programming for Personal Expression. http://scratch.mit.edu/files/CreativeCoding.pdf. Accessed 19 Oct 2007.

Reed, D. (2002): The use of ill-defined problems for developing problem-solving and empirical skills in CS1 *J. Comput. Small Coll.* **18**(1): 121-133.

Resnick (2002): Rethinking Learning in the Digital Age. In *The Global Information Technology Report: Readiness for the Networked World.* 32-37. Kirkman, G. (ed). Oxford University Press, Oxford.

Resnick, M. (2007): All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. In *Proc. of the 6th ACM SIGCHI conference on Creativity & cognition*, Washington, DC, USA, 1-6, ACM Press.

Rich, L., Perry, H. and Guzdial, M. (2004): A CS1 course designed to address interests of women. In *Proc. of the 35th SIGCSE technical symposium on Computer science education*, Norfolk, Virginia, USA, 190-194, ACM Press.

Romeike, R. (2006): Creative students - What can we learn from them for teaching computer science, In A. Berglund & M. Wiggberg (Eds.) In *Proc. of the 6th Baltic Sea Conference on Computing Education Research, Koli Calling.* Uppsala University, Uppsala, Sweden. Also available at http://cs.joensuu.fi/kolistelut/

Romeike, R. (2007a): Kriterien kreativen Informatik-unterrichts. In *Proc. of the 12. GI-Fachtagung "Informatik und Schule - INFOS 2007"*. Siegen, Germany, LNI 112: 57-68. Köllen.

Romeike, R. (2007b): *Designing Animations and Games - A Creative Introduction to Programming: About flying Elephants, Dogs, Cats and Ideas!* http://www.funlearning.de/. Accessed 19 Oct 2007.

Romeike, R. (2007c): Three Drivers for Creativity in Computer Science Education. In *Proc. of the IFIP-Conference on "Informatics, Mathematics and ICT: a golden triangle"*. Boston, USA.

Romeike, R. and Schwill, A. (2006): "The studies might be too difficult for me" Intermediate Results of a Long-term Survey of Computer Science Freshmen. In *Proc. of HDI 2006: Hochschuldidaktik der Informatik.* Munich, P-100: 37-49, Lecture Notes in Informatics.

Runco, M. A. and Chand, I. (1995): Cognition and Creativity. *Educational Psychology Review,* **7**(3): 243-267.

Saunders, D. and Thagard, P. (2005): Creativity in Computer Science. In *Creativity across domains: Faces of the muse* (Ed, Baer, J. C. K. a. J.), Lawrence Erlbaum Associates, Mahwah, NJ.

Scragg, G., Baldwin, D. and Koomen, H. (1994): Computer science needs an insight-based curriculum. In *Proc. of the twenty-fifth SIGCSE symposium on Computer science education*, Phoenix, Arizona, United States, 150-154, ACM Press.

Shneiderman, B. (2000): Creating Creativity: User Interfaces for Supporting Innovation. *ACM Transactions on Computer-Human Interaction,* **7**(1): 114–138.

Sternberg, R. J. and Lubart, T. I. (1991): Creating Creative Minds. *Phi Delta Kappan,* **72,** 608-614.

Sutinen, E. and Tarhio, J. (2001): Teaching to identify problems in a creative way. In *Proc. of the Frontiers in Education Conference,* IEEE Computer Society, TID-8-TID-13vol.1.

Sweeney, R. B. (2003): Creativity in the Information Technology Curriculum Proposal. In *Proc. of the 4th conference on Information technology curriculum.* Lafayette, Indiana, USA, 139-141.

Tharp, A. L. (1981): Getting more oomph from programming exercises. *SIGCSE Bull.,* **13**(1): 91-95.

Thomas, J. C., Lee, A. and Danis, C. (2002): Enhancing Creative Design via Software Tools. *Communications of the ACM,* **45,** 112 - 115.

Van Dyke, C. (1987): Taking “computer literacy” literally. *Communications of the. ACM,* **30,** 366-374.

Westby, E. L. and Dawson, V. L. (1995): Creativity: Asset or Burden in the Classroom? *Creativity Research Journal,* **8**(1): 1-10.

Wilson, B. C. (2004): A study of learning environments associated with computer courses: can we teach them better? **20**(2): 267 - 273.