Three Drivers for Creativity in Computer Science Education

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

Abstract
Research in the field of computer science education does not mention creativity very often. We believe that computer science (CS) in schools and universities can contribute a lot to the development of this treasured soft skill. This paper considers 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. Creativity is defined as something that leads to personal new, unique and useful ideas, solutions or insights. This view is shared by students of CS at the University of Potsdam, who perceive CS as a creative domain but do not experience creativity in their studies. In research studies a lack of motivational factors was found causing unsatisfying results in introductory programming courses. As the "optimal way" for teaching programming has not been found yet, a part of the key to it may be creativity. Three drivers for creativity in CS education (person, subject, information technology) are displayed in a model showing interrelations including related factors. Motivation and interest as factors connected with the creative *person* are essential to creative practice in CS. From studies with high-intrinsic motivated Open Source programmers the implications are drawn, that regarding usage, support to build up a reputation, fostering identification with the group and promoting purposeful learning may stimulate motivation and interest in the CS classroom as well. Considering the relevance of creativity in the *subject* of CS the process of designing software is compared with the creation of music. A model for creative software design based on a model for "music-design", studies investigating the software design process and a model of creativity is proposed. This model describes a stimulus, a set of knowledge and constraints as main influencing factors of the design process and reflects its cognitive and iterative character. The perception of CS concepts as "building blocks" is described as fertile for creativity. The role of this approach for learning and its value for CS education are discussed. The creativity supporting potential of *information technology* has been widely described. IT based learning environments used in the CS classroom fulfill the demands proposed for creativity supporting software, e.g. support of exploration and experimentation and immediate and useful feedback. Vice versa an understanding of CS concepts is important for being able to work creatively with IT. Thus CS education opens a door for students to be creative in other domains as well.
The consideration of these three drivers for creativity is suggested to all levels of computer science education. Discussions of this model and future research are encouraged.

Keywords
Computer science education, creativity, human factors, software design, IT

1 INTRODUCTION
Research in the field of computer science education does not mention creativity very often. As demanded from industry and suggested from pedagogy and psychology, educators need to emphasize the training of soft-skills as well as the "hard facts". Creativity is one of the core soft-skills. Already in 1950 Guilford (1950) asked why American schools were not producing more creative persons. We believe that computer science (CS) in schools and universities can contribute a lot to that issue. This paper considers 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. Accepting 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.

2 CREATIVITY
The term creativity is used with different meanings and is discussed controversially in psychology. Common speech usually defines something as creative when coming from the arts

or something extraordinary. But not just 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 shall an educator expect new and original achievements from his students? Boden (1990) describes two aspects of creative achievements. Historical creativity (*H-creativity*) describes ideas, which 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 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 want to call something creative if it leads to personal new, unique and useful ideas, solutions or insights (cp. Runco and Chand, 1995).

2.1 CS students' understanding and experiencing of creativity

Several CS professionals and researchers state that CS requires creativity (Glass, 2006, Ford, 2004, Leach, 2005, Saunders, 2005). Thus computer science education should reflect that matter and encourage students to develop and use their creativity. We asked 13 undergraduate 2nd and 3rd year students of computer science to reflect on how they experienced their CS lessons in high school regarding creativity and how creative they perceive the engagement with CS in their studies at university. The statements suggest an interesting perspective of creativity: Besides a few students, who distinguished between artistic and logical/scientific creativity, many students see creativity in any domain as combining and reusing "things" with the goal to finally create something new. "Things" can be more or less anything - material and nonmaterial, like colors, melodies or knowledge. We will refer to that later as the building block metaphor. As examples the design of algorithms, problem solving and programming were given. Hence CS is perceived as being a creative domain.

The students attended CS lessons within the last three to five years of high school. Half of the students experienced them as creative. The other half couldn't remember creative action in the classes. They report about tasks which usually consisted of learning a programming language and reproducing examples given by the teacher. Solutions to problems were discussed in class in detail or could be found by applying simple known patterns. Creative ideas and their own attempts for finding solutions were stopped by the teacher as being "wrong" and there was a general tendency of pushing everybody to do the same. The use of Software was covered mostly in a "How to" way, reinforced by convergent instructions of how to solve simple tasks. The students who perceived creative lessons reported about opposite: Everybody had the chance to solve problems in an individual way and there was time for experimenting and designing. The students indicate it as encouraging for their creative work when they received an initial framework for the task to be done. Working in projects was perceived as fun and motivating as it was interesting and allowed self-determined working.

The students do not perceive as creative their recent learning activities in the CS courses at university. Working creatively only happens now in their free time in private projects. The courses are perceived as theoretical and only seem to serve the purpose of accumulating knowledge and skills, but there is no demand and space to apply those creatively. Assigned homework usually refers to theoretical problems and proofs. Because of this some students feel a lack of motivation. But most of them expect that the tasks will become more creative in higher semesters. Then they hope to be allowed to apply what they have learned. The chance to be creative in CS is a motivating factor for the students.

2.2 Drivers for creativity in computer science education

In literature of psychology creativity often is regarded from different perspectives: The creative person, the creative process or the creative (supporting) environment[1]. In CS Education these perspectives can be reflected as three drivers for creativity: the *person* with his motivation and interest, the environment, which is generally formed by information technology (*IT*)

[1] Another perspective is the creative product in the context of measuring creativity. With our aforementioned definition of creativity in the educational context we will not regard this perspective.

and supports creativity, and the *subject* of software design itself [2], which involves the process of software design and building blocks as a way the matter can be perceived. The factors behind the drivers are interrelated: Interest fosters motivation and vice versa, as well as the design process can raise a student's interest. For software design an understanding of the building blocks as a representation of CS concepts is necessary and this understanding will support creative work with IT as well. The creativity supporting characteristics of IT again raise motivation. In the following sections we want to describe the roles and relevance of the three drivers in detail.

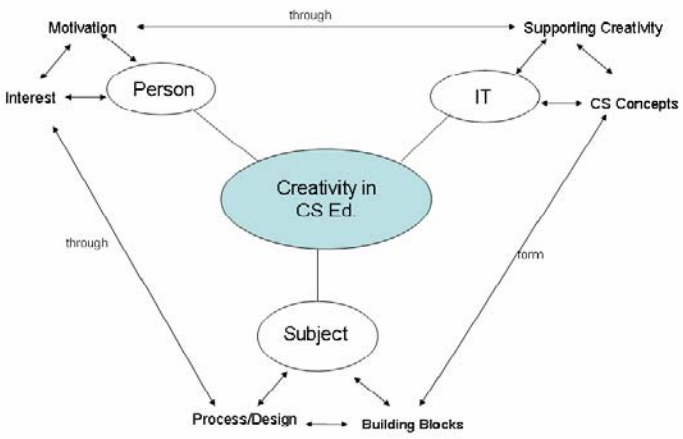

Fig. 1: Three Drivers for Creativity in CS Education

3 CREATIVITIY AND THE PERSON IN CS EDUCATION

Motivation is the driving force behind any human action. Without motivation we would not do our jobs, would not invest time in hobbies and would not learn anything exceeding simple facts. Educators know about the importance of motivation. Intelligence, aptitude and social background are factors that can influence learning, but they are out of reach of the teacher. Around 20 percent of a student's achievement can be attributed to motivation (Asmus, 1994). This possibility to raise a student's achievement may be used as a powerful tool by the teacher. Since the results of many introductory programming courses are not satisfying, a number of studies have been conducted to investigate factors causing the unpleasant situation. The findings regarding motivation include:

- High-intrinsically motivated students performed better (Bergin, 2005)
- Many students did not have a general interest in programming per se (Curzon, 1998, Mamone, 1992)
- Motivation was raised by using meaningful tasks and exercises (Rich et al., 2004, Tharp, 1981, Feldgen, 2003)
- Motivation was improved by assigning personally challenging tasks, e.g. competitions (Lawrence, 2004)

The "optimal way" for introducing programming has not been found yet; a part of the key to it may be creativity. While programming in the classroom often is not perceived as motivating and creative, outside of the classroom some interesting observations can be made that may reflect the real nature of programming: The development of software has a fascinating impact on many programmers and students in CS. They invest an enormous amount of time for writing programs, students learn, even self-educated, how to program and some Open Source programmers spend more then 20 hours of their weekly free time unpaid with the development of software. These people seem to be driven by a strong intrinsic motivation similar to artists who are dedicating themselves intensively to their work (cp. Moravcsik, 1974). Why is there such a contradiction between how programming is perceived inside and outside of the classroom? Does programming play a special role within the free-time activities? Where does the desire and dedication to write or manipulate software come from? Is there something that educators could use to raise intrinsic motivation in CS classes? Lakhani and Wolf (2005) as well as Luthiger Stoll (2006) investigated the motivation of software developers in Open Source projects. They identified enjoyment-based intrinsic motivation, namely how creative a person feels, as the strongest and most pervasive driver. Additionally they found a variety of creativity related factors that were responsible for intrinsic motivation and thus for

[2] We refer do software design as it is a central issue of CS.

the participation in those projects: Usage, reputation, identification with the group, learning and altruism. Similar factors were found to be responsible for student engagement in programming in CS classrooms – some students dedicate a lot of energy to the creation of software and spend a good deal of their free time for doing that. In an interview with one of those students similar motives for intrinsic motivation, especially the chance to be creative in programming, were mentioned (Romeike, 2006). As these motives correspond to the factors that were identified as missing in the aforementioned studies, they may serve as a key for programming motivation. Since for some students these motivators already seem to be obvious enough to raise intrinsic motivation, emphasizing and utilizing those may help to inspire the rest of the class as well. The following implications can be drawn to raise motivation and interest:

Regarding usage: An obvious reason for developing software is the wish to use it later on. If the programmer has a problem that he cannot solve with available software he needs to program it him- or herself or to modify existing software so that it fits to his or her needs: Tasks shall be meaningful and offer usage to the students. "Pseudo problem orientation" as the modeling of a flash light on a computer will not meet this motive.

Supporting to build up a reputation: One of the maxims of the Open Source community is that contributions of each developer can be exactly traced. This way every individual can build up a reputation, which may be useful for later job applications or can strengthen the person's prestige in the community. Since learners usually just receive feedback from the teacher, disseminated software can be appreciated outside of the classroom, e.g. on personal homepages and thus raise motivation.

Fostering identification with the group: An individual's close integration into a group can lead to a strong identification with the goals of that group. This motive can be successfully applied in group programming projects.

Promoting purposeful learning: The chance for Open Source programmers to learn is a reason to participate in the projects; they want to extend their experiences and improve their skills in order to become a better software developer. When students become aware of the use of the facts and concepts to be learned and when they find it interesting to build software, the goal to do this better can be motivating[3].

As seen, motivating students for learning to program can be troublesome. When creativity is regarded in the context of programming, it is the extracurricular observations that suggest it is easier to raise intrinsic motivation and interest; as tasks will become personally challenging and meaningful. Considering the motivators found in Open-Source Programmers when designing lesson may be helpful. The effects of such lessons need to be verified by empirical research.

4 CREATIVITY IN THE SOFTWARE DEVELOPMENT PROCESS

The subject of CS, with software design being central to CS, is a very creative one. While the creation e.g. in music usually is not questioned about being creative, this indeed is different with software design. To show similarities between the two we want to draw parallels from the field of musical composition and improvisation, which by no doubt is a creative one, to software design. We will call it "music design" to make parallels more obvious[4]. Finally we will focus on "building blocks" as a way to look at CS concepts applied in the design process.

4.1 A multidisciplinary viewpoint – a black box model of factors influencing design in music and software development

Design is according to the Compact Oxford English Dictionary

1. A plan or drawing produced to show the look and function or workings of something be-

[3] This can be observed in computer science lessons of especially motivated students who discovered programming as a hobby and keep asking the teacher for information that exceeds the actual lesson content.

[4] The term music design is seldom used, as the terms composition or musical improvisation are more intuitive. Anyhow, especially in the context of contemporary music and developing music with IT the term "Music design" is getting popular for music creation.

fore it is built or made.

2. *The art or action of producing such a plan or drawing.*
3. *Underlying purpose or planning: the appearance of design in the universe.*

Three times it is stated that planning is involved in design. A dedicated Jazz-lover may start wondering: Isn't at least Jazz-music all about spontaneity? How can that be planned? Isn't it common to hear from great artists sentences like "I just play what I feel"? Maybe there is no design in Jazz-music? The answer to that question comes with a quote from Wynton Marsalis: "Jazz is not just, 'Well, man, this is what I feel like playing.' It's a very structured thing that comes down from a tradition and requires a lot of thoughts and study." (Quoted in Berliner, 1994) The jazz musician falls back on a great inventory of patterns, models, concepts and outlines that he or she is applying in the moment of improvisation to the design of his music[5] and performs it real-time. The composer does the same thing: He scoops out of his inventory of knowledge (facts, concepts, etc.) and puts it into the design of his musical work. Regarding his culture group, customs and personal preferences he has to obey certain constraints such as tonality, style, harmonic rules, rhythm and others. Assuming that there usually is a reason for creating a musical work[6] we can include these factors into a model of factors influencing the musical design process. This black box model points out key factors considered in music education.

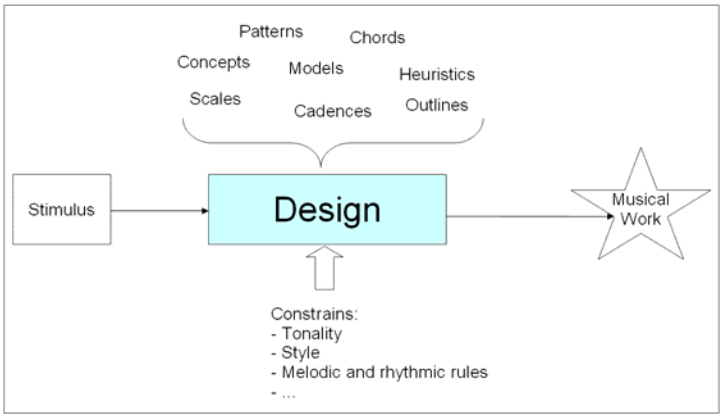

Fig. 2: Black box model of factors influencing the musical design process

The design of software is not that much different. Looking at the influencing factors in the black box model, parallels become obvious: Also in software design there is a stimulus that starts the process. Often it is called a problem, task, request or any kind of motivation for somebody to develop software. The design process is strongly influenced by constraints, which need to be obeyed by the programmer. Those constraints include circumstances of how, where and from who the software shall be used as well as limitations of the programming language he uses, resources and many more. During the design process the programmer uses a pool of problem solving patterns, heuristics and experiences and applies them in finding and realizing a software solution. This process also results in a product[7]. Support for these factors comes from a protocol analysis study with professional software designers by Guindon and Curtis (1988). Investigating the process of design they categorized and concretized the constraints (e.g. design process constraints, problem domain constraints or functional constraints) and the knowledge resources (e.g. algorithms, design schemas, design methods).

We call this process creative as long as it is applied to a new problem, new software is designed and there is no known pattern for this task. It involves creativity as jazz improvisation and musical composition demands creativity. Obviously different processes will require different amounts of creativity, but the model shows us important factors that influence the creative process: An elaborated pool of knowledge, concepts and experiences is the foundation for creative design in software development[8]. As CS education regards the importance of knowing these facts, it is also necessary to regard the way how they are applied in the design process, which will be described as follows.

---

[5] This actually is what makes a great jazz musician: there is a design behind his music.

[6] Generally at the start of a musical composition or improvisation stands an impulse, impression or task.

[7] Possibly the work of the software designer ends with the software design and the implementation is done elsewhere. This may be comparable with the written composition sketch in music.

[8] Research claims that at least 10 years of expertise in a field are necessary for H-creative achievements.

4.2 Software design and the creative process

Even though it is essential to know the influencing factors, the black box model does not tell us much about the design process itself. In an empirical study by Curtis et al (1987) investigating the design process 5 steps where found to be involved:

1. *Understanding the problem*
2. *Decomposing the problem into goals and objects*
3. *Selecting and composing plans to solve the problem*
4. *Implementing the plans*
5. *Reflecting on the design product and process*

Those steps describe the process of design very generally. For educators they imply the different phases they need to highlight when teaching how to do software design. On the other hand these findings basically describe the general process of problem-solving and have very much in common with the software life cycle as well. Important are the findings concerning the detailed investigation of the step "Selecting and composing plans" as the essence of the design process. The following steps were found (cp. Glass, 2006):

3.1. Build a mental model of a proposed solution to a problem
3.2. Mentally execute the model to see if it does indeed solve the problem (simulation)
3.3. If the sample output is incorrect the model is expanded to correct its deficiencies, then executed again
3.4. When the sample output finally becomes correct, another sample input is selected and steps two and three are repeated
3.5. When sufficient sample inputs have passed the test in step four, the model is assumed to be a suitable design model and representation of the design begins

The findings make the cognitive and iterative character of software design obvious. Thus trial-and-error and heuristic processes are natural and important to software design. This aspect is likely to be overlooked when teachers only foster a top-down approach to software design in a mechanistic way. The finding that a big part of the process is cognitive is supported by Goorhuis' theory of constructivist modeling in CS (Goorhuis, 1994), who describes steps 1 and 2 as the cogitation phase which results in an internal model. Then analogous to steps 3.1. to 3.5. a constructional phase results in a second internal model, which is the starting point for the realization of the model. More support stressing the importance of cognitive factors are contained in statements of software pioneers in interviews in Lammers (1986).

The model of the creative software design process shall be concluded by drawing parallels to Amabile's model of creativity (Amabile, 1996). This model is based on three components affecting the process: Domain-relevant skills (our pool of knowledge), creativity-relevant processes (e.g. experience, heuristics, and idea generating techniques) and task-motivation[9]. The different stages are roughly divided into problem or task identification, preparation, response generation, response validation and communication and the outcome in the end. As we consider the design of software being a creative process we can now integrate our findings into a model for a creative software design. From an educators point of view the model suggests the following implications:

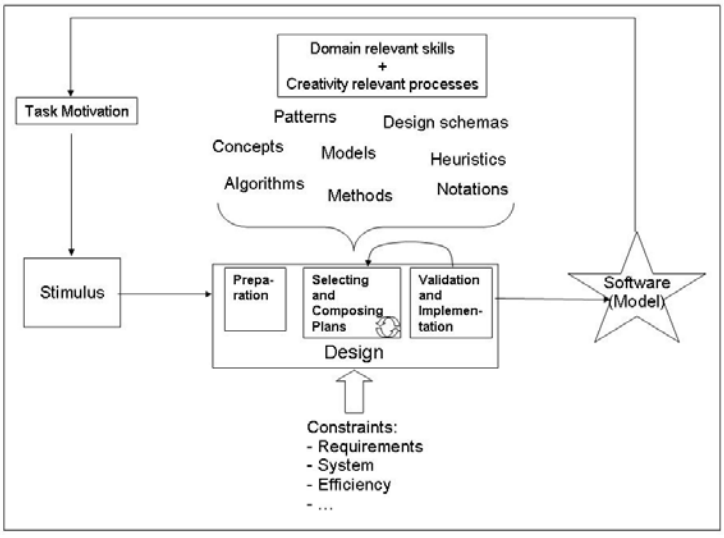

Fig. 3: Model of the creative software design process

1. Domain relevant skills and creativity relevant skills are essential to software design.
2. Motivation is an essential part of

[9] As shown in section 3, motivation is essential to creative achievement.

the creative design process.
3. Constraints which influence the design process need to receive attention.
With that in mind all the influencing factors should be regarded when teaching software design.

4.3 Building blocks and creativity

A solid foundation of knowledge in the domain is a necessary requirement for creative processes in general and in particular in software design. Beneficial for creativity in CS is that facts and concepts necessary for programming can be perceived as building blocks, are well defined and documented and thus easy to use: E.g. data types are modeled using the building block principle and for the modeling of processes the different programming paradigms with their construction kits are used. Another construction kit would be e.g. the catalogue of fundamental ideas (Schwill, 1997), which describe important concepts and strategies of CS. This metaphor can also be applied to computer systems: Computer systems function according to precise rules. As they consist of a big amount of pieces the challenge of CS lies in building and combining them. Many pieces already exist, but to put them together certainly takes creativity. Glass (2006) talks about a set of primitives every designer and programmer possesses and which varies from person to person. A primitive describes the point where a designer ends the design and implementation can start – a set of well understood elements that do not need further thinking through. Thus designing software always draws back on a certain amount of primitives that are put together like building blocks. Also programming approaches follow that principle: e.g. the combining of blocks of code in procedures in structured programming or encapsulation in object orientation.
The benefit of building blocks for creativity is obvious: Kids like to play with building blocks because they have a manageable amount of uses, the possibilities of how they can be used are clear and an unlimited number of different constructions can be built with them. In playing with such building blocks kids develop an enormous amount of creativity and learn about structures and stability as well. Bruce et al (2004) used the term “building blocks” to describe a way learners may approach writing their programs. In this approach experimentation and exploration of the material was part of the learning process. Programming environments designed for learning also make use of the same metaphor. Building up on Logo, which already involves building block like commands, Logo blocks successfully visualizes the concept. Several further developments have been done and programming environments like Scratch or Alice[10] let beginners of programming experience the act of building software in a way like playing with building blocks: Involving experimentation, learning and understanding concepts and creative play. This approach to understand software seems to be fertile for creative ideas. In this view creativity in CS is within the reach of every student, as it consists of putting together well understood building blocks to find new solutions for known and unknown problems.
As implication for educators CS concepts should be taught in a way which makes the building blocks obvious and distinguishable but shows their relations between each other.

5 CREATIVITY AND INFORMATION TECHNOLOGY IN CS EDUCATION

A creative environment is essential for creative achievements. With its development information technology (IT) is more and more used as a tool providing a fertile environment for creative work. As many people see it, IT is by is nature a powerful amplifier for creative practices (Mitchell et al., 2003). This is, as software can be easily copied and disseminated; it supports the application and creative recombination of innovation and thus easily allows to create things that would be out of reach without IT. As IT is omnipresent in the CS classroom and makes up a major part of the learning environment, there lies a big potential for creative work: On one hand, IT supports creative practice and stimulates ideas; on the other hand the knowledge of CS concepts is essential for creative achievements with IT in other fields as well.
Several programming environments used in the CS classroom do support creative practice

[10] http://scratch.mit.edu/ and http://www.alice.org/

while facilitating CS concepts. One that is used successfully in Computer Club Houses in the U.S is Scratch, a building-block-based visual programming environment developed at the MIT (Peppler, 2005). With Scratch already young students learn concepts of programming while being engaged creatively.

5.1 IT Support for creative practice

Shneiderman (2002) proposed specific tasks for software tools that should support creativity: IT should support

- pain-free exploration and experimentation
- immediate and useful feedback for one's actions
- no big penalties for mistakes, meaningful reward for success
- easy way to undo and redo
- visualizing data processes
- searching for knowledge and inspiration
- composing a work step by step
- disseminating results to gain recognition

Computer based *experimentation and exploration* can inspire new ideas and should involve design, simulation, surprise and creation (Mitchell et al., 2003). Programmers utilize the possibility of experimentation allowed by the programming environments by testing out ideas and possible paths to solutions. Using selective trial and error is an essential part of software development (Glass, 2006). Those "what if" scenarios are explored quickly and can easily be changed. This is only possible due to the support of the environment. Scratch, as other programming environments for beginners, makes exploration even more accessible; operations can easily be executed on the objects and the results can be observed in real-time. Thus also learning to program involves the exploration of the programming environment and language ("What are my possibilities?") and the experimentation with the objects and constructs they offer ("What happens if I apply them like this?").

The *immediate and useful feedback for one's actions* given by the compiler or interpreter supports the experimentation and is quite unique to software development – in other fields the tools to work with either do not provide qualitative feedback automatically or do not point out problems[11]. It can be observed with many students that the action-related feedback promotes a sense of control and with it self-regulated learning[12].

As Shneiderman requires *no big penalties* from creativity supporting software, the biggest penalty of a SDE is the error-feedback itself. As students in other fields have the teachers or class-mates as only source of feedback, when learning to program this is done by the machine in a not intimidating way. The functioning program will be rewarding for success, e.g. in Scratch this will be a nice animation or game.

Supporting for experimentation and consideration of feedback is the certitude that nothing can "break" easily. Scratch offers an easy way to undo and redo the last changes.

While the aforementioned points of creative work in other fields only are possible in simulations, in programming the experimentation is done at the product itself.

Environments for learning how to program also meet others of Shneiderman's criteria, e.g. *visualizing data and processes*, which helps to organize ideas and facts and to discover relationships. The trend to visual programming reflects that creativity-relevant aspect. In Scratch programming constructs (such as loops, operations, broadcasts) are represented as colorful building-blocks, separating the kinds of constructs in different colors. In design oriented approaches data structures and processes are modeled and viewed as diagrams with their relationships and may be automatically transformed into code and vice versa (e.g. with Fujaba). The trend to such a visual, more abstract, level of programming suggests that it will become even more emphasized in the future and thus support creativity better.

It is important for a creativity supporting environment to *make knowledge easy accessible and provide inspiration*. Programming environments allow searching in a help knowledge da-

[11] E.g. musical or art experiments need to be judged by others, not software-based experiments in physics or chemistry do not tell where the problems are if they fail.

[12] Programming is often even learned autodidactic.

tabase and provide a documentation of comments and examples. Scratch additionally offers inspiring example-programs made by other children and provides cards that explain constructs to be used and initiates experimentation with those constructs.
Composing a work step-by-step allows to slowly approach a problem by leaving plenty of possibilities open as not all important decisions are made at the beginning as with the top-down design. While top-down design is more emphasized in professional software development, the process of learning to program may be different (Kaasbøll, 1998) and involves a bottom-up step by step approach. It allows to review the work done and to evaluate it in the light of what shall be done. This is important to creative practice, since often the outcome is not set clearly from the beginning. In Scratch this matter is implemented by making it possible to run and review a program at any point of the process.
Disseminating the results to gain recognition is important for peer-recognition and for motivation of creative work. As the classroom only offers limited possibilities to present one's achievements, the internet allows to present them to a wide audience. Scratch has a built-in function to upload a work to an internet server where the programs can be shared and evaluated by other learners and also serve for inspiration to others[13].
Hence Scratch, as other environments for learning to program do, fulfills Shneiderman's tasks for creativity supporting software.

5.2 CS concepts as a base for creative practice with IT

While software tools used in CS education support creativity in CS, the understanding of CS concepts is fundamental for being creative with IT in other domains as well. Since with software tools novices can do advanced crafts (e.g. in photography or music), IT-fluency, which includes knowledge of CS concepts, is important to creative practice with IT. There is a difference between basic functional know-how of a software tool and high-level skills or fluency. As several studies show (cp. Computer Science and Telecommunications Board, 1999) important conceptual capabilities include algorithmic thinking, facility with principles of knowledge representation and adaptability to change. In the context of computer programming and creativity especially the use of Logo has been investigated and it has been shown that Logo can support the processes that underlie creative performance (Papert, 1993 Clements, 1995). Thus CS education opens a door for students to be creative in other domains as well.

6 CONCLUSION

Even though computer science education and creativity are highly interrelated, the opportunity of applying creative practice to raise motivation and interest is hardly reflected in CS education research. We suggest the consideration of the three drivers for creativity to all levels of CS education. With the identification of these three drivers for creativity (person, subject, information technology) we prepared a basis for discussion and further investigation of the relevance of creativity in CS education. Future empirical research needs to validate the impact of considering creativity in the classroom on students' achievement and other factors mentioned.

7 REFERENCES

Amabile, T. M. (1996) Creativity in context: Update to the social psychology of creativity. Westview Press, Boulder, Colo. [a.o.].

Asmus, E. P. (1994) Motivation in Music Teaching and Learning. The Quaterly, 5, 5-32.

Bergin, S., Reilly, R. (2005) The Influence of Motivation and Comfort-Level on Learning to Program. In Proceedings of the PPIG 17. J. G. In P. Romero, E. Acosta Chaparro & S. Bryant. University of Sussex, Brighton UK. 293-304.

Berliner, P. F. (1994) Thinking in jazz: the infinite art of improvisation. University of Chicago Press, Chicago.

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

Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M. and Stoodley, I. (2004) Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. Journal of Information Technology Education, 3, 143-160.

[13] As the easy dissemination of novices writings has led to an increasing amount of creative literature in the web also the dissemination of software through the web can be motivating for creative practice.

Clements, D. H. (1995) Playing with computers, playing with ideas. Educational Psychology Review, V7, 203-207.
Computer Science and Telecommunications Board, N. R. C. (1999) Being Fluent with Information Technology. National Academic Press, Washington, D.C.
Curtis, B., Guindon, R., Krasner, H., Walz, D., Elam, J., Iscoe, N. (1987) Empirical Studies of the Design Process. In Proceedings of the Second Workshop on Empirical Studies of Programmers.
Curzon, P., Rix, J. (1998) Why do Students take Programming Modules? In Proceedings of the 6th an-nual conference on the teaching and computing and the 3rd annual conference on integratingtechnology into CSE: Changing the delivery of Computer Science Education. ITICSE '98. Dublin, Ireland. 59-63.
Feldgen, M., Clua, O. (2003) New motivations are required for freshman introductory programming. In Proceedings of the 33rd ASSE/IEEE Frontiers in Education Conference. Boulder. T3C-T24.
Ford, N. (2004) Creativity and Convergence in Information Science: The roles of objectivity and subjectivity, constraint, and control. Journal of the American Society for Information Science and Technology, 55, 1169-1182.
Glass, R. L. (2006) Software creativity 2.0. developer .* Books, Atlanta.
Goorhuis, H. (1994) Konstruktivistische Modellbildung in der Informatik. Diss. Zürich.
Guilford, J. P. (1950) Creativity. American Psychologist, 5, 444-454.
Guindon, R. and Curtis, B. (1988). Control of cognitive processes during software design: what tools are needed? In Proceedings of the SIGCHI conference on Human factors in computing systems, (eds), ACM Press, Washington, D.C., United States, 263-268.
Kaasbøll, J. J. (1998) Exploring Didactic Models for Programming. In Proceedings of the Norsk Informatikk-Konferanse. Høgskolen I Agder.
Lakhani, K., 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, B. F. J. Feller, S. Hissam, and K. R. Lakhani (eds), MIT Press, 3-22.
Lammers, S. (1986) Programmers at work : interviews. Microsoft, Redmond, Wash.
Lawrence, R. (2004) Teaching data structures using competitive games. IEEE Transactions on Education, 47, 459- 466.
Leach, R. J., Ayers, Caprice A. (2005) The Psychology of Invention in Computer Science. In Proceedings of the 17th Annual Workshop of the PPIG. University of Sussex, Brighton UK.
Luthiger Stoll, B. (2006) Spass und Software-Entwicklung: Zur Motivation von Open-Source-Programmierern, Zürich.
Mamone, S. (1992) Empirical Study of Motivation in an Entry Level Programming Course. ACM SIGPLAN Notices, 27, 54-60.
Mitchell, W. J., Inouye, A. S., Blumenthal, M. S. and National Research Council (U.S.). Committee on Information Technology and Creativity. (2003) Beyond productivity : information technology, innovation, and creativity. National Academies Press, Washington, DC.
Moravcsik, M. J. (1974) Scientists and Artists: Motivations, Aspirations, Approaches and Accomplishments. Leonardo, 7, 255.
Papert, S. (1993) The children's machine : rethinking school in the age of the computer. BasicBooks, New York.
Peppler, K., & Kafai, Y. (2005) Creative Coding: Programming for Personal Expression.
Rich, L., Perry, H. and Guzdial, M. (2004). A CS1 course designed to address interests of women. In Proceedings of the 35th SIGCSE technical symposium on Computer science education, (eds), ACM Press, Norfolk, Virginia, USA, 190-194.
Romeike, R. (2006) Creative Students - What Can We Learn From Them for Teaching Computer Science? In Proceedings of the Koli Calling. Koli.
Runco, M. A. and Chand, I. (1995) Cognition and Creativity. Educational Psychology Review, 7, 243-267.
Saunders, D., Thagard, Paul. (2005). Creativity in Computer Science. In Creativity across domains: Faces of the muse, J. C. K. a. J. Baer (eds), Lawrence Erlbaum Associates, Mahwah, NJ.
Schwill, A. (1997) Fundamental ideas - Rethinking computer science education. Learning and Leading with Technology, 25, 28-31.
Shneiderman, B. (2002) Creativity support tools. Commun. ACM, 45, 116-120.
Tharp, A. L. (1981) Getting more oomph from programming exercises. SIGCSE Bull., 13, 91-95.

Biography

Ralf Romeike is a research associate at the University of Potsdam, Germany as well as a teacher for computer science at a high-school in Potsdam. He studied computer science and music and focuses his research interest on the use and encouragement of creativity in computer science education.