Tinkering in Informatics as Teaching Method

Angelika Mader Ansgar Fehnker Alma Schaafstal
Creative Technology - EEMCS - University of Twente - Netherlands

1. Creative Technology – Setting the stage

Creative Technology is one of the faster growing bachelor programmes of the University of Twente. It is a rather young programme, seven years old now, residing within the faculty of Electrical Engineering, Mathematics and Computer Science. Creative Technology is a design programme, with an engineering background in Computer Science and Electrical Engineering. The analogous programme would be Industrial Design, where the design material is provided by Mechanical Engineering. In our programme Creative Technology students learn how to deploy the latest technology to develop interactive installations geared towards an impact on human lives. The bachelor programme takes three years to complete (180 ECTS).
The University of Twente has introduced the Twente Educational Model for her Bachelor Education. This model encourages students to manage their own learning process, embodied in the concept of 'Student-Driven Learning'. Through this student-driven learning programme, we aim to produce students and graduates who take responsibility for and steer their own learning paths, and thus promote the entrepreneurial attitude we strive for at the University of Twente. Central in our system are the 'modules', integrated pieces of the curriculum of 15 ECTS, in which various topics are taught in an integrated manner and applied in a project.
The programme Creative Technology has been able to attract a wide diversity of students. Every year we have about 25% of female students and an equal number of students is international. The program is also divers in its intake in the sense that it admits students from a wide range of backgrounds, ranging from very technical students, to students who have only little mathematical or technical background in highschool and who e.g. have never programmed before. We try to challenge our students with education that matters to them, and offer them as much freedom to make their own choices as possible, also when it comes down to choosing applications that they would like to work on. We believe that it is easier to motivate students to do things that they like when there is a relevant context involved than presenting them with assignments lacking a context that they consider interesting or appealing.
The wide diversity of students poses a challenge for us. How are we able to satisfy such a broad range of students? How are we able to make students successful in an engineering discipline if they do not have a proper technical background? How do we attract even

more female students, knowing that they are traditionally underrepresented in STEM-programmes?

On the engineering side of our program, students have to master a significant portion of math and physics, and they all have to learn to program at a decent level. The goals for the programming part are globally speaking as follows:

Students should gain a:

- fluency in programming
- mastery of algorithmic thinking
- understanding of programming design
- show proper programming style
- competency in the use of programming as a medium for expressing creative ideas

Learning to program is a topic which is taught in a number of modules, thus progressively deepening the knowledge and looking at the topic from various angles and various application domains.
In this proposal, we will show how we approach the teaching of programming skills in our curriculum. This approach, Tinkering in Informatics, has proven that we are able to accomplish the aforementioned goals, but are also able to create successful programmers.

2. Description of Achievements

The main achievement described in this submission, is the introduction of tinkering as a means to teach programming. By tinkering we understand a self-directed, playful exploration of material. Often starting with a seemingly undirected investigation of the material, after a while self-chosen goals are set, experiments are defined and executed to achieve the goal, observations and interpretations lead to a next goal to choose. In an iterative process the tinkerer explores the material of a given toolbox and the possibilities how to get it working, by a series of experiments and interpretations of the results of the experiments [1], [2], [3].
While tinkering workshops are popular for children and school kids, especially for STEM education, so far the approach has not been considered as suitable for an academic environment. In the following we will, first, argue, why we are convinced that the tinkering approach is valuable in an academic environment [2]. Second, we will describe how we implemented this approach in an informatics education context. Third, we will present observations from the courses and results.

2.1. Tinkering for informatics in an academic context.

From the material point of view, we are convinced that the tinkering approach is a key to **mastering material**. Mastering the material, knowing its properties, potential, and how to use it to get thinks working, is essential for any design or engineering, being at a university or outside. While the material for tinkering can be at very different levels, from lego bricks to electronic components or concepts, we focus here on programming and algorithms.
From an academic point of view the tinkering process also stimulates **core scientific activities**, such as raising questions, performing experiments, observing, interpreting, and theory forming. These skills come short in curricula that are dominated by courses that teach existing theory, but not how to go beyond or how to apply it. Tinkering requires these activities on a small scale (depending on the level of material provided). Therefore, we consider tinkering as an essential part of academic education.
Another crucial aspect of this process is the students' **ownership of their learning process**. Classical teaching often explains solutions to students for problems they do not have. In an active learning process students get a deeper understanding of solutions if they help them to achieve the goals they have defined themselves.
The playfulness and self-directed aspect of the approach contribute to a **higher motivation** of the students.

2.2. Implementing the tinkering approach for Informatics teaching.

For informatics in an academic setting it is the question **how to implement a tinkering approach**. In earlier work [2] we identified a number of ingredients that support the process: a seed (catalyst - new technology), a tool box, and a design goal. In the following we will describe how which form we gave these ingredients, and which additional elements were found to be necessary.
While we use tinkering as element or driving mechanism in several courses on programming and also physical computing, the focus of this text is on a first year course "Algorithms for Creative Technology", after basic programming concepts have already been introduced in earlier courses.

the seed.
The programming environment chosen is Processing, built on Java, and originally developed for artists and designers. While classical concepts of programming languages are present (data structures, object orientation, recursion, etc.), Processing is tailored for graphical output and user interaction. Both of these ingredients make it an attractive programming environment: code produces visible output and interaction can easily be added, which can be rewarding for the student from the first exercises on, and has certainly a motivating effect. From a teaching aspect the language offers the concepts relevant at this stage of Informatics lecturing.

the design goal.
The students have to define their individual design goals for their projects - they do not have to solve a given problem. The framing of the individual design goal is such that they

have to write their own program using elements from different sets of building blocks, i.e. algorithms that were covered in the course. Moreover, their program has to satisfy given rules of programming style and complexity. The program is assessed during an individual oral exam in the end of the course. The students know this setting from the beginning of the course. They can start from the beginning with ideation and use (almost) each building block they create during the course for their individual end assignment.

the tool box.
The tool box consists of a range of algorithms that are arranged in different topics, such as “randomness”, “forces”, “particle systems”, “mass-spring-damper systems”, etc. Roughly once per week a new topic is introduced, and a number of assignments are included for each topic. Assignments typically consist of a fixed part (such as to use certain laws of physics to build code for a catapult) and an part that can be individually designed (such as the form of the catapult and the interaction of a user with it).

Next to programming concepts and design that have to be covered here, much focus is on an appealing selection of algorithms, stimulating the playfulness. For example, a particle system connected to a moving object can either be a fire tail of a rocket or glitters spread by a fairy. In both cases, the students learn, e.g., about how to structure code into different classes, dependencies between classes, methods and value passing, and get a first idea of complexity when a huge number of particles needs to be supported by an array management, and particle systems.
Each assignment done is considered as a building block for the end assignment, where students also can select from different topics.

Next to these general ingredients of a guided tinkering process identified, we found a couple of principles useful to support the context

ambition levels.
As described above, the variation in the student population is very high, ranging from students with no programming experience before entering university, to students who have already a semester or bachelor in informatics or another technical programme that includes programming courses. The other dimension of variation is the ambition level. A couple of students tries to optimise their work effort to just pass the course. Motivation can reach some of them, but many stick to their mind set. Others are highly motivated, independent of their prior knowledge. Experience from more traditional courses shows that often the little ambitious students consume a lot of attention to bring them to an course average, and they determine the overall level and speed.

All these observations together led to the introduction of three ambition levels, where each student can choose which level she wants to follow. For each ambition level assignments are offered. With the lowest level assignments a student can pass, with the highest level reach the highest grade, with the medium level inbetween. As said above, the assignments provide the building blocks for the final assignment to present, which is (for a major part) evaluated according to the level of the building blocks used.

For students this choice supports **ownership**, as they have taken a decision. Furthermore, it provides a **clear setting** for both the students and the lecturer, by explicit expectation management, reducing discussions about effort and ambition. And a third positive effect is that for the **very good and/or ambitious students,** first, challenges can be provided that increase also their motivation, and second also more time for attention becomes available.

personal support.
Only a few lectures are given, just for introduction of each new topic, which are in most cases not longer than 20 minutes. The rest of the time students have time to work on the assignments and to get help when needed. A team of student assistants and the lecturer(s) are busy with individual support. The main insight here is that students have a much deeper understanding of a solution when they struggle with a problem to solve, than when getting a solution without understanding the problem in depth. Additionally, as described above, the variation in the student population is very high, which also shows in the different speed students work. With individual help we can much better adapt to the individual speed and needs of the students.

responsibility.
Making students owners of their learning processes means also to give them more responsibility. Responsibility, on the other side, is also a key for motivation and creativity. In a teaching context, making students more responsible, creates space for lecturing and content instead of continuous checking whether rules are followed. In university programmes that look more and more like school, it seems to be relevant to create space to let students learn responsibility, including a safe zone for failing.

3. Evidence of Impact

The impact is about didactic methods, about motivation of students, ownership, dealing with diversity, and finally about getting talented female students on board.maturity in tinkering.
In general, while tinkering seems to be a natural way of getting to know material, it is not a way students are used to when entering university. Seemingly, school education focusses much more on reproduction of content than curiosity driven learning. As a consequence, students have to learn tinkering. This does not take only one Informatics course, but includes several courses and projects. With maturity of the students the toolboxes may contain more abstract elements (like serious gaming or social media), in the end they should be able to take the whole world as toolbox for their tinkering process. A clear defined design goal is more important for beginners, experienced tinkerers can handle the process with just a new seed (for example a new technology such as the hololense).

diversity in working.
During the tutorials the variation in speed, strategies in accessing the assignments, learning styles (ranging from using video tutorial, existing examples, tinkering, to

discussion with fellow students), and solutions found is overwhelming, supporting the approach that respects the diversity of the students.

maturity in responsibility.
While all students are positive about the possibilities in choices of ambition levels, speed and order of doing assignments, and the possibility to get help, not all of them can really cope with the responsibility well. A part uses the freedom to postpone and trying to catch up in the last weeks achieving overall weaker results. Among the students being present and working actively, a higher percentage of female students is present than male w.r.t. the distribution in the whole population. It seems that this way of learning fits better to their way of working.

variety in solutions.
The final assignments are all different, which suggests little copying behaviour of students. Checking by a plagiarism detector showed that the only allowed sources were used, **no plagiarism** could be identified (even by using automated plagiarism detection).

individual support.
Students often do not ask for help even if they got stuck. Waiting for questions often does not provide much interaction with lecturer and student assistants. Instead, walking around and asking students what they are busy with results often in interesting discussions.

quality of results.
Analysing the results of the course in the past 3 years (see table 1), the overall failure rate ranges between 7% and 14% and seems to be in the line of general introductory courses in Informatics [5]. In the three years analysed, female students, however have a slightly lower failure rate (0% - 7%).

year	gender	number of students	percentage of the population	average grade	number of failed	percentage of failed (wrt gender)	percentage of failed (total)
2015	female	27	33%	7,26	0	0%	14%
	male	54	67%	6,26	11	20%	
2016	female	27	32%	6,86	2	7%	7%
	male	57	68%	7,41	4	7%	
2017	female	32	42%	7,29	1	3%	9%
	male	45	58%	7,37	6	13%	

Table 1: Gender separated results of the course “Programming and Physical Computing”

The average grades between male and female students seem not to differ. By a number of reasons, this is a very positive result. First of all, girls and women, in average, have a lower self-image concerning their abilities in STEM subjects, which is especially manifest in the Netherlands [4] (which is the origin of the majority of the students). A low self-image, typically, has also effect on the results achieved in these subjects [4], [6]. If there is no gender gap observable for the results, the teaching approach has compensated for that. In [6] the authors report that a physical computing approach closed the gender gap in their programming course. By similar reasons, we assume that the tinkering approach with playful algorithms increases the motivation of female students, and provides a context where they can give meaning to the learning content, which is especially relevant for women in the STEM context. This assumption is confirmed by a testimonial of a female student (see next item below).
A second reason, why the absence of the gender difference in grades is a positive result, is the prior knowledge. The percentage of girls choosing for a STEM focus during school in the Netherlands is lower than the one of boys (27% versus 42% for boys, [4] p22). Accordingly, it could be expected that girls enter the university with a lower STEM background. For programmes where only few women enrol, these few are better than the average among girls. In our case, the percentage of female students is about a third, which suggests that much more of the average background is represented here. Still, we cannot observe a possible influence of a lower STEM background in the grades.

On a more qualitative level we have the following observations:
Students with little or no prior knowledge in programming could achieve excellent results. Surprisingly, students with a background in informatics (e.g. a semester or a bachelor in informatics) come up with well-structured programs, but in average little creativity. Obvious was that the students with successful results were driven by enthusiasm to make their self-chosen concept work. Theses concepts included game elements, aesthetic animations, story telling, or pure quantity, all outside the scope of programming. Surprising were also a number of assignments constructed from simple building blocks, but beautifully arranged to complex and sophisticated combinations with original concepts of interaction.

testimonials of students.

Maaike (second year student)**:** "Before I began the study Creative Technology I had never programmed before. I started with programming at a low level in the first year and it build up to harder assignments such as simulating physical systems. When I started programming I realized that it is not just for male students and that it is really enjoyable. With the tinkering method Creative Technology offers, I learned to think out of the box, solve a lot of my own problems through systematic debugging and to make more creative programs."

Margot (master student after having passed the Creative Technology bachelor, also working as a software developer next to her study): **"**I always felt that the goal of our

programming courses was not teaching us how to best program; instead the goal was to learn how to create visuals for your digital products and art installations. Programming just happened to be the right tool. I think that approach still sets me apart when I now collaborate with traditionally educated computer scientists, ánd with designers.
You don't learn an algorithm just because the assignment tells you to – you learn it because it helps you show something on screen. *Seeing* your code turn into for example physics, natural looking flocks of birds, and modern works of art is very motivating. It's like learning how to translate aspects of the world around you to code."

4. Evidence of Availability of the Outcomes of the Initiative to the Teaching Community

local publication:
At the University of Twente, the work on tinkering was identified as an excellent teaching method and included in a booklet/magazine:
Section on Tinkering in the University of Twente Magazine "A Tribute to Excellent Teachers", June 2017. pdf

conference contributions:
In Creative Technology we realized from the beginning, that we should not present the material in the same way as it is done in classical courses of Computer Science and Electrical Engineering. We approached the search for alternatives with the question "How to Educate for Creativity in Creative Technology?", which resulted in a conference paper [3] containing the experiences from the experiments during first years of Creative Technology. We identified tinkering there as one of the promising approaches that we investigated more intensively in the following years, which lead to the conference contribution [2] "Tinkering as Method in Academic Teaching", which contains most of the basis for this text.

portfolio, manual:
A portfolio of the first year programming courses of Creative Technology can be found here, especially the collection of assignments, forming the building blocks for tinkering in the course "Algorithms in Creative Technology".

References

[1] M. Resnick and E. Rosenbaum. Designing for Tinkerability, in: Design, Make, Play: Growing the Next Generation of STEM Innovators. Taylor & Francis, 2013.
[2] A. Mader and E. Dertien, "Tinkering as Method in Academic Teaching". in E Bohemia (ed.), DS 83: Proceedings of the 18th International Conference on Engineering and

Product Design Education, E&PDE 2016, (pp. 240-245), Aalborg, Denmark, 8-9 September.pdf
[3] A. Mader and E. Dertien, "How to Educate for Creativity in Creative Technology?" In E. Bohemia, A. Eger, W. Eggink, A. Kovacevic, B. Parkinson, & W. Wits (Eds.), Proceedings of the 16th International conference on Engineering and Product Design, E&PDE 2014 (pp. 568-573). pdf
[4] C. Booy, N. Jansen, G. Joukes, and E. van Schaik, "Trend Analysis Gender in Higher STEM Education", VHTO - National Expert Organisation on Girls/Women and Science/ Technology, 2012. pdf
[5] Jens Bennedsen and Michael E. Caspersen. Failure rates in introductory programming. SIGCSE Bull. 39, 2 (pp. 32-36), June, 2007.
[6] M.A. Rubio, R. Romero-Zaliz, C. Mañoso, and A. P. de Madrid. Closing the gender gap in an introductory programming course. Comput. Educ. 82, C (pp 409-420), 2015.

Appendix

Letters of Support