Tinkering in Informatics as Teaching Method

Angelika Mader, Ansgar Fehnker and Edwin Dertien
University of Twente, Enschede, The Netherlands

Keywords: Tinkering, Programming Education, Teaching Methods.

Abstract: Our university offers an IT-based design programme, with an engineering background in Computer Science and Electrical Engineering. Its focus on design and creativity, as well as the diversity of the students, requires an approach in informatics courses different from classical computer science programmes. While tinkering is an increasingly popular approach in STEM stimulation and education outside university, we argue that also in an academic setting a tinkering mindset has a relevant contribution. In this paper, we identify key elements in setting up tinkering sessions and report on their implementation for a course on algorithms. We will present and discuss results and observations of our teaching method, that are promising to continue and extend the tinkering approach in an academic setting.

1 INTRODUCTION

Our university offers an IT-based design programme with an engineering background in Computer Science and Electrical Engineering. It is one of the fasted growing programmes of our faculty. In this bachelor programme, students learn how to employ the latest technology to develop interactive installations geared towards an impact on human lives, in contexts such as health, entertainment, or environment. Knowledge, interest and skills in design, culture, human-centred problem solving, art and creativity are essential for this programme, next to sound technical knowledge.

The program attracts a very diverse student population. Many students have little STEM background. For example, in a 2019 questionnaire among first-year students of the algorithms course – which was completed by 46 of the 107 students – 39% reported that they had never, and 30% that they had rarely programmed before entering the degree. Over the five years from 2014 to 2018, 35% of all enrolments were female students. This compares to 10% in informatics, and 8% in the electrical engineering degree at our university. A similar share of students in our degree has an international background, which also means that they come from different education systems.

While we embrace the diversity for the originality and quality of the results our students deliver, it also poses a challenge for us. Especially in the courses on mathematics, informatics and electrical engineering, the prior knowledge of entering students varies considerably.

In this programme, we consider informatics as a means for students to express there ideas and concepts and focus less on the scientific questions of informatics. Still, students need to have an understanding of a computer and software, and in particular of programming. They need to be familiar with decomposition, structure, and abstraction and need to be able to apply non-trivial concepts of programming, such as classes and methods.

This setting of a programme with a very diverse student population and its specific mindset and content geared towards creativity requires a different approach to teaching informatics than classical informatics programmes. In this paper, we describe our approach in introducing *tinkering* as a teaching method for informatics that caters to the diversity of the students. Here we take the example of a first-year course on algorithms. We also report on our findings and experiences from almost 10 years.

Section 2 describe *tinkering*, emphasising both the academic and teaching perspective. Section 3 describes the principles of the tinkering process and the core ingredients for setting up tinkering sessions. How we implement them for a course on algorithms is elaborated in Section 4. Evidence and observation will be presented in Section 5, followed by Section 6 discussing the approach. Section 7 concludes this paper.

Mader, A., Fehnker, A. and Dertien, E.
Tinkering in Informatics as Teaching Method.
DOI: 10.5220/0009467304500457
In *Proceedings of the 12th International Conference on Computer Supported Education (CSEDU 2020) - Volume 1*, pages 450-457
ISBN: 978-989-758-417-6

2 TINKERING

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. Subsequently, observations and interpretations lead to the 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 (see (Resnick and Rosenbaum, 2013)). This process is guided by serendipity rather than structure (Libow Martinez and Stager, 2013) having trial and error at its core.

Conceptually, the actual notion of tinkering is shaped by the maker movement[1]. This movement was initiated by the MIT with its first *fablab* in 2002. Fablabs provide tools for prototyping, next to classical workshop environment also laser cutters, 3D-printers and open source hardware, and make them accessible for the public. Makers are often amateurs and hobbyists who create new products, for their individual usage, or for value in the community. Tinkering and making are related, where tinkering emphasises the explorative aspect of making.

The material part of tinkering is often viewed as physical. However, according to (Resnick and Rosenbaum, 2013): *We see tinkering as a style of making things, regardless of whether the things are physical or virtual. You can tinker when you are programming an animation or writing a story, not just when you are making something physical. The key issue is the style of interaction, not the media or materials being used.* One abstraction level higher, tinkering would also be possible with concepts or in completely different domains (e.g. gaming, health, music, philosophy). In the context of this paper, we use algorithms and data structures as material to tinker, which is explicitly non-physical.

We understand tinkering as a mindset for learning (Libow Martinez and Stager, 2013). The goal of the tinkering process for the student is to create an original prototype. Our, the teachers', goal is that the students get familiar with the material of the toolbox and learns to apply it, as well as training their serendipity and creativity.

From the *maker perspective* tinkering is at the heart of *making* things. The maker movement[2] seems to have adopted tinkering as one of its core activities, as the way of learning new skills, working with physical materials and source of inventiveness. For example *The Exploratorium* (Wilkinson and Petrich, 2014) has been developing kits, books and materials aimed at STEM education for all ages, focusing on sparking curiosity, enabling creativity and making fun. Tinkering from a maker perspective usually starts with (re)building an existing idea or concept (from a kit, book or other sources) and use that as 'seed' or starting point for new things to build.

From an *engineering perspective* mastering material is a key skill. Mastering the material, knowing its properties, potential, and how to use it to get things to work, is essential for any design or engineering, being at a university or outside. Engineering typically strives for an effective solution to a problem. We are convinced that the tinkering approach is a key to mastering material. Exploration and experimentation are at the core of tinkering and results precisely in the knowledge about the material. It generates hands-on knowledge and reflection on the experiments conducted, (as *knowing in action* in (Schön, 1983)), it cultivates serendipity for what could be a working solution. Reflection on the experiments is an important step in the learning process, as in (Tawfik and Kolodner, 2016) stated: *the more effort the reasoner has put into identifying what can be learnt from experience and when the lessons might be useful to pertain, the better the learner will be able to label the experiences and apply them for future use.* In addition, mature tinkerers and engineers have a technological theory and scientific foundations in their toolbox, and also know how to apply these within their design processes (Boon, 2006).

From an *academic perspective* the tinkering mindset 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 focus on teaching 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) and trains them. Especially, setting up a meaningful experiment also requires tinkering. Therefore, we consider tinkering as an essential part of academic education, next to science. To go even a step further on the perspectives on engineering and academia, in (Libow Martinez and Stager, 2013) (p41) the claim can be found that *tinkering is exactly how real science and engineering are done.*

From a *teaching perspective* we understand tinkering as a mindset for learning (Libow Martinez and Stager, 2013). For the students, the goal of the tinkering process is to create an original prototype. Our, the teachers', goal is in the first place that the students get

[1] https://makerfaire.com/maker-movement/

[2] See https://en.wikipedia.org/wiki/Maker_culture

familiar with the material of the toolbox and learn to apply it. In the second place, we intend it as training of their serendipity and creativity, as well as their ability to reflect. i.e., to internalise the tinkering mindset.

A tinkering approach supports *ownership*. As suggested in (Savery and Duffy, 1995), *learners must have ownership of the learning or problem-solving process as well as having ownership of the problem itself*. Taking own decisions is a cornerstone of ownership and, with it, motivation. This is also one of the arguments in project-based learning (Savery and Duffy, 1995; Savery, 2006): students deciding about their own process and solutions have a more active learning experience. Often, classical teaching is about telling students solutions to problems they do not have. Accordingly, the understanding of the solution and its peculiarities is not deep. Instead, *having* a problem gives a completely different understanding of the possible solutions and their different qualities. Tinkering even allows to define the own problems, and also how to solve them. Problem, solution and the process are in the hands of the student, the setting is what has to be provided by the teaching environment.

Another factor in the teaching perspective is the possibility of making mistakes. Understanding the nature of mistakes can be very effective for the learning process. In contrast, in classical teaching making mistakes has no single positive connotation. In tinkering the whole concept of iteration is built on mistakes and improvement, thus also here tinkering contributes to a better learning process.

Obviously, tinkering also is an excellent approach to cope with diversity: within the setting given, students can choose their problems individually, as well as their speed, experiments, depth, horizon etc.

From the *life long learning* perspective self-directedness is a main ingredient (Loyens, 2008). As an element of tinkering will be a crucial method to master new technologies and explore its potential. Due to fast technological progress and severe global problems, students and the whole society has to keep learning all life. School and universities are not the places where students can learn everything they need later in life. Education here can strive for giving a basis and teach students to find their own ways and methods of learning, and foster serendipity. In the context of informatics, there is continuous innovation in languages, paradigms, and platforms. A student having internalised a tinkering mindset will always find her/his way in these new developments.

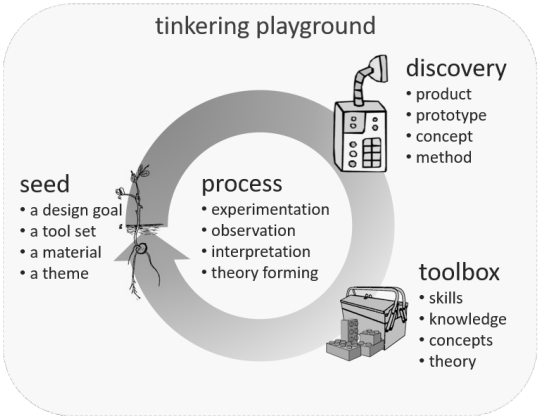

Figure 1: The iterative process of tinkering prompts exploration of the toolbox.

3 PRINCIPLES OF TINKERING EDUCATION

In this section, we first describe shortly the ingredients of a tinkering setting in general. While we use these ingredients for tinkering approaches in several courses, this paper focuses here on a first-year course on algorithms. For this course, we describe the implementation chosen in the next section.

Setting up education with a tinkering approach can be done by following a number of principles, here considered in a broad context of application domains. Building upon results of (Mader and Dertien, 2016), we consider the principles of tinkering education consisting of the following:

The Seed. A stimulant serving as a starting point. This may be a first goal, it may be a toolset or material, or a theme (e.g., tinkering with time). The nature of a seed depends very much on the maturity of the participants: those used to a tinkering mindset need less, e.g. just a new building block, than beginners in tinkering.

The Discovery. The self-defined outcome of the process. It may be a new product or prototype, or a new way of doing things, a new concept or the (even intended) learning outcomes proposed by the teacher or facilitator.

The Toolbox. It includes physical or conceptual building blocks, but also skills, knowledge and templates. A toolbox is for one part provided in a dedicated tinkering session, for another part it is the experience and knowledge an individual participant brings in.

The Process. This is the path from the seed to the discovery, guided by the tinkering mindset, it is a hands-on approach, playful and driven by serendipity, performed in an iterative way.

The Facilitators. Are guarding the process and the mindset, set the mood, introduce the seed, keep the threshold low and the ceiling high, give feedback, keep the flow, and stimulate reflection as an important part of the learning process (Tawfik and Kolodner, 2016).

The Playground. Although tinkering can be viewed as a mindset (Libow Martinez and Stager, 2013) - as activity or process it is usually bounded in time and place. With the playground, the *situatedness* of the activity is defined. For example, the environment where tinkering takes place should be supportive and inspiring (Doorley et al., 2011).

Figure 1 illustrates the iterative nature of the tinkering approach. The process is driven by core tinkering activities that use trial and error to explore and extend the toolbox. Prior seeds and discoveries become part of the toolbox, new iterations are fuelled by seeds emerging from prior discoveries through applying the toolbox, thus closing the loop for the next iteration.

4 CHOICES AND IMPLEMENTATION

In this section, we describe the choices taken to implement the principles of tinkering education mentioned above in an informatics context. We use tinkering in a number of different courses; for this paper, we focus on a course on algorithms, offered at the end of the first year of the programme.

The Seed. The programming environment chosen is Processing, a language 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 for education: 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. It comes with a great number of examples that demonstrate the scope of applications and stimulate to set levels.

The Discovery. The students have to define their individual design goals for their final assignment - 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. From a given set of building blocks treated in the course, they may choose a subset for their final assignment, or use other algorithms of comparable complexity. Moreover, their program has to satisfy given rules of programming style and complexity. The program is assessed during an individual oral exam at 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 assignments are given each week allow for some freedom in the solution, e.g. the assignment *attach your particle system to a moving object* lead to solutions consisting a flying rocket with smoke as particles, and of a unicorn with glitters as particles, both equally good.

The Toolbox. The toolbox 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 a part that can be individually designed (such as the form of the catapult and the interaction of a user with it).

In addition to programming concepts and design that have to be covered here, much focus is on an appealing selection of algorithms, stimulating the playfulness. while the individual applications of the algorithms may differ, 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 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.

The Facilitator. These are here the lecturer(s) together with a great number of student assistants. The student assistants are students of higher years, some of them already finished with our study programme

and following a different master programme. They are all experienced, with toolbox and mindset, and most are truly enthusiast about programming and the tinkering way of working. In contrast to other informatics courses, we always have a lecturer present, and sufficiently many assistants to prevent too long waiting periods, which might be demotivating to students. Grades are only given for the end assignments. During the course only material is provided, explained, feedback and help given.

The Playground. Tinkering takes place in time and space. Typically in education, this is bound to lecture rooms and scheduled hours. In general, a stimulating environment with flexible working space would be ideal. Practically, we get only minimal requirements fulfilled such as that each student is physically accessible by a student assistant or a teacher.

Next to these general ingredients of a guided tinkering process identified, we found a couple of concepts useful extending the line of education described so far.

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 completed a semester or bachelor in informatics or another technical programme that includes programming courses. The other dimension of variation is the ambition level. Some students try to optimise their work effort to just pass the course. Motivation can reach some of them, but many stick to their mindset. 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 a 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 or he wants to follow. Assignments are offered for each ambition level. With the lowest level assignments, a student can pass, with the highest level they can reach the highest grade, with the medium level in between. As said above, the assignments specify the building blocks the final assignment is expected to include, 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 made the decision themselves. Furthermore, it provides a clear setting for both the students and the lecturer, by explicitly managing expectations, and thus it reduces discussions about effort and ambition. 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 the introduction of each new topic, which are in most cases no 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 driving 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.

The tinkering approach was also applied to the two programming courses that precede the course used for the discussion in this chapter. The first introductory programming course includes two main iterations. Students are given as first seed the goal to make an interactive creature. Every week a new concept is introduced to the toolbox, from variables, loops, decisions to classes and objects. The second iteration is the final project. The seed is the goal to animate an artwork, each year with a new theme. For the final students have to use everything that was introduced previously, with the addition that the new animation should incorporate proper composition of objects. The second course adds various aspects of physical computing to the toolbox, and the seed defined by a theme, for example, to build a musical instrument. This illustrates that the tinkering approach can be implemented in different contexts for students of different maturity.

5 EVIDENCE AND OBSERVATIONS

The impact is about didactic methods, about the motivation of students, ownership, dealing with diversity, and finally about getting talented female students on board.

Learning Tinkering. In general, while tinkering seems to be a natural way of getting to know the material, it is not a way students are used to when

they enter university. Seemingly, school education focuses much more on the 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. The course in the focus of this paper is the third course in the programme where tinkering concepts are used. Students who are motivated for learning, have taken up the concept.

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), is overwhelming, supporting the approach that respects the diversity of the students.

Variety in Solutions. The solutions students present in the end also show a huge variation. Popular themes are storytelling, games, or simply aesthetic scenarios, most often a mixture of two or three of these elements. It is not uncommon that novel types of solutions emerge in a course.

Plagiarism. The final assignments are all different, which suggests little copying behaviour of students. Checked with a plagiarism detector we found no shared code between students, except for the pairs that worked together, and sources allowed. However, there are still students who let friends or family write their code when they find it too difficult. Some also do not add sufficient own content beyond the sources allowed.

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

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. Some students use 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.

Debugging. Most students show poor skills in debugging. Typically, they build a piece of software and call for help when it is not working.

Quality of Results. Considering the results of the course in the past 3 years (see table 1), an overall failure rate below 20% and seem to be in the line of general introductory courses in Informatics (Bennedsen and Caspersen, 2007). In the years analysed, female students, have a slightly lower failure rate than males.

The average grades between male and female students seem not to differ significantly. For a number of reasons, this is a very positive result. First of all, girls and women, on average, have a lower self-image concerning their abilities in STEM subjects, which is especially manifest in the Netherlands (C. Booy and van Schaik, 2012) (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 (C. Booy and van Schaik, 2012; Rubiol et al., 2015). If there is no gender gap observable for the results, the teaching approach has compensated for that. In (Rubiol et al., 2015) 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, (C. Booy and van Schaik, 2012) p22). Accordingly, it could be expected fewer girls enter the university with a STEM background. For programmes where only a 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 can 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. These con-

Table 1: Gender separated results for the course “Algorithms”.

Year	Gender	Number of Students	Precentage of Total	Average Grade	Precentage Failed by Gender	Precentage Failed Total
2015	f	27	34%	7.3	0%	13%
	m	53	66%	6.4	19%	
2016	f	27	32%	6.9	7%	7%
	m	57	68%	7.4	7%	
2017	f	32	42%	7.3	3%	9%
	m	45	58%	7.4	13%	
2018	f	45	43%	6.9	4%	7%
	m	59	57%	6.8	8%	
2019	f	36	34%	6.5	17%	19%
	m	71	66%	6.6	20%	

cepts included game elements, aesthetic animations, storytelling, 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 builds up to harder assignments such as simulating physical systems. When I started programming I realised 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, and 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.

6 DISCUSSION

With our approach, we reach many students, but not all of them. Still, there is a number who just want to pass the course with minimal effort, who are not taking the step to playing and tinkering, and often do not show up for lectures. We expect that we still could reach some of them using different, or additional strategies.

Teaching students debugging remains an ongoing problem. In interviews student assistants, also from informatics courses, report this as a basic shortcoming in the skill-set of most students. However, for a proper tinkering process identifying the errors and bugs, reflecting and learning from them, and improving on them are core elements.

The toolbox for a course does not contain a fixed collection of building blocks. For the course on algorithms, often enough solutions for assignments pop up on the internet about two years after the assignment came into the course. This means that there is a need for a continuous update, finding or developing new, appealing assignments or increase the level of difficulty for some assignments.

The main question, however, is, how tinkering can be realised in a setting where teaching goals have to be achieved and work graded, in a given time frame, contrary to the openness of the process. We are trying to circumvent this contradiction by giving open assignments with the requirement to use a subset of the building blocks of the toolbox, and teachers and students assistants steering in the direction of quality of results or giving students a choice in the ambition level.

Teaching tinkering to students is a time-intensive

process for lecturers and student assistants. Individual trajectories require individual feedback, and, in the end, individual evaluation. It is obvious that this approach does not scale up straightforwardly. When the team of teaching assistants is growing, also consistency is becoming an increasing problem. To overcome this, we currently are working on a tool that supports teaching assistants in giving feedback and sharing it with each other.

Seeing surprising work of students is very rewarding. Also, seeing students who are surprised about their own achievements makes this time-intensive way of teaching worthwhile. In the final projects of the study programme, we also see that many students have a tinkering mindset, even if for some of them this was not obvious in the preceding courses.

The promising results and observations presented in the previous section, like the quality of results, the possibility to cater populations with a diverse background, the varieties in solutions and working styles observed, plagiarism nearly eliminated, convince us that this is the right approach to follow, keeping working on shortcomings.

7 CONCLUSION

In this paper, we reported on our approach to applying a tinkering mindset in informatics teaching. We elaborated the role of tinkering in teaching in general, at academia and for engineering. For the implementation of tinkering in teaching, we identified key ingredients, such as the toolbox, the seed, the design goal and the facilitator. While we use this schema or elements from it in a number of courses, we focused in this paper on the implementation in a first-year course on algorithms. The results experienced so far are very promising, little drop-out, gender-differences in grading are disappearing, no plagiarism due to individual assignments, and a number of very enthusiast students with surprising results.

Still, there are challenges to work on, like getting more students in a mindset of playing. The next step we will address is the scalability of the approach: the process is feedback intensive which cannot be extended straightforwardly to higher numbers of students. Currently, we are identifying elements of feedback that can be tool-supported, giving more space for the inherently necessary feedback on the individual learning processes.

REFERENCES

Bennedsen, J. and Caspersen, M. E. (2007). Failure rates in introductory programming. *SIGCSE Bull.*, 39(2):32–36.

Boon, M. (2006). How science is applied in technology. *International studies in the philosophy of science*, 20(1):27–47.

C. Booy, N. Jansen, G. J. and van Schaik, E. (2012). Trend analysis: Gender in stem education. VHTO (National Expert Organisation on Girls / Women and Science / Technology).

Doorley, S., Witthoft, S., University, H., and Kelley, D. (2011). *Make Space: How to Set the Stage for Creative Collaboration*. Wiley.

Libow Martinez, S. and Stager, G. (2013). *Invent to learn: Making, tinkering, and engineering in the classroom*. Constructing modern knowledge press Torrance, CA.

Loyens, Sofie M. M.and Magda, J. R. R. M. J. P. (2008). Self-directed learning in problem-based learning and its relationships with self-regulated learning. *Educational Psychology Review*, 20(4):411–427.

Mader, A. and Dertien, E. (2016). Tinkering as method in academic teaching. In Bohemia, E., editor, *DS 83: E&PDE 16*, pages 240–245.

Resnick, M. and Rosenbaum, E. (2013). *Design, Make, Play: Growing the Next Generation of STEM Innovators*, chapter Designing for Tinkerability. Taylor & Francis.

Rubiol, M. A., Romero-Zaliz, R., noso, C. M., and de Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. *Comput. Educ.*, 82(C):409–420.

Savery, J. R. (2006). Overview of problem-based learning: definition and distinctions, the interdisciplinary. *Journal of Problem-based Learning*, pages 9–20.

Savery, J. R. and Duffy, T. M. (1995). *Problem Based Learning: An instructional model and its constructivist framework*. Educational Technology Publications, Englewood Cliffs, NJ.

Schön, D. (1983). *The reflective practitoner*. Basic Books.

Tawfik, A. A. and Kolodner, J. L. (2016). Systematizing scaffolding for problem-based learning: A view from case-based reasoning. 10(1).

Wilkinson, K. and Petrich, M. (2014). *The Art of Tinkering*. Weldon Owen.