Teaching Theoretical Computer Science using a Cognitive Apprenticeship Approach

Maria Knobelsdorf
New York University
Computer Science Department
719 Broadway
New York, NY, 10003, USA
(+1) 212 998 3497
maria.knobelsdorf@cs.nyu.edu

Christoph Kreitz
University of Potsdam
Department of Computer Science
August-Bebel Str. 89
D-14482 Potsdam, Germany
(+49) 331 977 3060
kreitz@cs.uni-potsdam.de

Sebastian Böhne
University of Potsdam
Department of Computer Science
August-Bebel Str. 89
D-14482 Potsdam, Germany
(+49) 331 977 3014
boehne@uni-potsdam.de

ABSTRACT
High failure rates in introductory courses on theoretical computer science are a common problem at universities in Germany, Europe, and North America, as students often have difficulties coping with the contents of such courses due to their abstract and theoretical nature. This paper describes modifications to the pedagogy of a theory course held at the University of Potsdam, Germany that are motivated by a cognitive apprenticeship approach and have led to a significant reduction of the course's failure rates. Since our approach is based on the typical infrastructure for teaching introductory computer science courses and does not require additional expenses or special resources, it can be replicated by other institutions. We believe that it is a serious contribution to better support teaching as well as student learning success in this field.

Categories and Subject Descriptors
K.3.2 **[Computer and Information Science Education]**: Computers and Education - Computer and Information Science Education

General Terms
Human Factors

Keywords
Theoretical Computer Science, Theory of Computation, Cognitive Apprenticeship, Pedagogy, High Failure Rates.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCSE'14, March 5–8, 2014, Atlanta, Georgia, USA.
Copyright © 2014 ACM 978-1-4503-2605-6/14/03…$15.00.
http://dx.doi.org/10.1145/2538862.2538944

1. INTRODUCTION
At German universities, theoretical Computer Science (CS) is considered one of the fundaments of undergraduate CS education. Introductory courses cover the foundations of automata, programming languages, computability, and computational complexity. Introducing idealized mathematical models of the computer and discussing methods for designing and analyzing them, students are supposed to develop the ability of thinking abstractly about computational processes. However, many students have problems understanding and following the course topics and failure rates in final exams are usually very high. This was also the case at the department of CS at the University of Potsdam, Germany. The introduction to theoretical CS is covered by an undergraduate course (briefly "Theory I") that initially had very high failure rates (usually between 30-60%). Teaching assistants who worked weekly with the students reported on their difficulties in coping with the course material, observing a learning edge momentum very early during the course, see [10]. High failure rates in theoretical CS at the University of Potsdam are not a single phenomenon. Failure rates in other undergraduate CS courses tend to be very high as well and dropout rates from the CS major are not much lower than the national average, which is about 40-50% with most students giving up during the first four semesters, see [7].

Reasons for failing a final exam can be very different and change over time. In addition, various factors can have different influence on a student's overall progress and final learning outcome. In the past 30 years, a considerable number of studies investigated possible influencing factors on student success in CS courses like mathematical pre-knowledge, abstract thinking abilities, gender or social aspects, see ([12], p. 224-228) and ([2], p. 30-32). These and further recent studies indicate that there is no silver bullet to explain student success just by specific factors. Instead, the whole teaching and learning process needs to be focused on.

Standing in the tradition of constructivism and student-oriented pedagogy, there is a variety of approaches to improving learning success in courses on theoretical computer science. Chesñevar et al. [3] focused on student engagement for deeper understanding and introduced "the historical context in which the theory of computing emerged as a new discipline" (p. 8). Hämäläinen [6] used problem-based learning and Korte et al. [8] developed a constructionist approach with game-building. In addition, Rodger et al. [11] suggested the visualization tool JFLAP for formal languages and automata theory, which shall enable students to

interact more with the introduced theoretical concepts. All approaches succeeded to engage students in more learning activities resulting in lower dropout and failure rates. Since we did not have access to additional resources, we focused on developing new ways to engage students in more learning activities with the resources we had.

During the winter semesters 2011/2012[1] and 2012/2013 we modified the setup of Theory I by consolidating experiences from ad-hoc changes to the courses in the past eight years and taking into account the pedagogical approach of cognitive apprenticeship, a well known approach in education that has already been applied to CS education; see for example [2]. In the literature, however, there is no report that it has been used to improve theoretical CS education at university level. The result of our adjustments to the course was that less than 10% of all students failed the final exam while the remaining students' scores were evenly distributed. Apart from a small amount of additional contact time our approach does not require any extra resources beyond the common infrastructure given at German universities for an introductory CS course. Therefore, our approach can be replicated by other institutions as well. In this paper, we will introduce cognitive apprenticeship and describe the adjustments to the setup of our course and discuss the final assessment that indicates the credibility and impact of our approach.

2. THE THEORY I COURSE

In this section we will introduce the Theory I course, the students who attend it, as well as the traditional pedagogy applied before we introduced our first modifications.

2.1 Course Content and Learning Outcome

The Theory I course focuses on the theory of automata and formal languages. It begins with an investigation of simple automata models and increases the level of complexity until the models reach the capabilities of modern computers. Specifically it covers the following topics:

- Regular languages: finite automata, non-determinism, regular expressions, type 3 grammars, closure properties, and limitations.
- Context free languages and type 2 grammars, pushdown automata, normal forms, parsing algorithms, closure properties, and limitations.
- General and context sensitive languages and grammars, Turing machines, linear-bounded automata, closure properties.

In the course of the lectures, students are supposed to develop specific skills that enable them to work with the models introduced in the course. For this reason, the department of CS of the University of Potsdam defined in 2010 normatively the intended learning outcomes that are described by three fields of specific skills ([9], p. 36-38):

- Domain specific skills, for example: students are able to analyze deterministic and non-deterministic automata and grammars with mathematical methods and to prove their correctness; students know methods for translating between different automata models, grammars, and regular expressions and are able to justify the correctness of these conversions.
- Methods skills, for example: students can employ mathematical proof techniques for the analysis of automata and formal languages; they can transform grammars into normal forms; they are able to prove whether a given formal language is regular (context free) or not.
- Behavioral skills, for example: students are capable of working in small teams when developing solutions for given problems; they are able to give an oral presentation of their solutions and they can write them down in a precise language.

Through the course, students are supposed to develop these skills and the final exam is supposed to assess these.

2.2 The Theory I students

Before entering an undergraduate program at a university in Germany, students are required to finish upper secondary education, which, depending on the school type, takes two or three years (grade 11-13 where students usually are 17-19 years old). With specific mandatory basic and advanced courses, upper secondary schooling can be roughly compared to the first one or two years of attending a college in the US. In consequence, entering a German university students immediately start with their major. The predominant majority of German universities are public and students do not pay tuition. Students are admitted to university in Germany based on their final grades at upper secondary school.

The number of students attending the Theory I course usually varies between 150-300 (which is a rather small number for an introductory course at a German university) and consists of 50-150 students majoring in CS, about 60-100 students majoring in Software-Engineering, 40-80 students majoring in business informatics, 10-25 prospective teachers with one of their majors in CS, and approximately ten students majoring in different fields (like mathematics, linguistics or others). CS and business informatics majors are supposed to attend the course in their first semester and software engineering majors in their third. While at many other German universities, CS majors attend an introductory course in theoretical CS in their third or fourth semester, the CS curriculum at the University of Potsdam places the course at the beginning of undergraduate studies, since many subsequent courses expect students to be familiar with automata theory and Turing machines.

2.3 Traditional Course Setup and Problems

Until we modified the course setup and its pedagogical approach, the course consisted of the following components, which are typical for an introductory CS course of this size in Germany:

- 135 minutes of lectures peer week given by a faculty member who presents the course topics, central concepts, algorithms, and their proofs and illustrates them with examples.
- Weekly homework assignments based on the current lecture topics, which students are expected to solve individually and submit in writing for reviewing and grading by tutors (usually senior students). Handing in homework can but doesn't need to be mandatory.

[1] The German academic system has a winter and a summer semester with teaching periods between mid October and mid February (with a two weeks Christmas break) and between mid April and mid July, respectively.

- 90 minutes student session every other week attended by approx. 25-30 students and chaired by tutors, during which attending students are expected to present their solutions to last weeks' homework assignment. This shall give them an opportunity to check the correctness of their solutions and discuss them with the group.
- One final written exam during the assessment period after the end of the lectures, which in the winter takes place at the end of February, determines the grade students receive for the course.

The pedagogical approach behind these course components assumes that students understand the presented concepts, theorems, and proofs during the lectures and are able to deal with the homework assignments by themselves. There are students who respond positively to this approach. However, our tutors reported that most students remained very passive during the student sessions and did not participate in discussions even when their own solutions contained mistakes. Taking the high failure rate in the finals into account, we concluded that based on the course's pedagogical approach most of our students are not able to build understanding and become well prepared for the finals. This forced us to reconsider all elements of the course and seek for improvement.

3. FIRST COURSE ADJUSTEMENTS

Starting in 2003, we began introducing a variety of modifications to the traditional setup of the course. We converted the bi-weekly student sessions to weekly sessions that focused on student activities instead of the presentation of solutions. We reduced the time for lectures to 90 minutes per week, made homework submissions mandatory, and offered an additional weekly "tutorial". We hoped that these modifications would help reducing the failure rates without reducing the requirements for passing the final exams.

3.1 Tutorial and Exercise Sessions

As described in Sec. 2.3, the goal of the Theory I lecture is to present new topics, central concepts, algorithms, and their proofs and to illustrate them by examples. We noticed that, due to the number of attendees, the lecture lacks interaction between students and the instructor. Also, there was only little time for individual questions and discussions.. Therefore, one of our first changes was to install an additional weekly 90 minutes session that we named *tutorial*. In the tutorial, held by the instructor, students are encouraged to ask questions and discuss issues that were still unclear after lecture and student sessions. The topics discussed in the tutorial are not prepared in advance but suggested by the attending students, as the main objective is not the solution to a problem but the process of producing it.

We also believe that student interaction is much easier to accomplish during student sessions attended by approx. 25 students than during a lecture. Therefore, we also shortened the lecture to 90 minutes and expanded the exercise session to 90 minutes per week. In order to enable the teacher to present the same amount of topics in less time, lecture topics started to be presented on slides that are prepared in advance instead of being written onto the blackboard. In addition, the use of a tablet PC enabled the lecturer to add details and illustrations to the slides during the lecture and thus preserve the advantages of a "blackboard lecture". In order to offer students the possibility to prepare before and after the lecture, the slides were made available at the beginning of the entire course. Furthermore, in 2006/07 and again in 2011/2012 the lectures were recorded on video and made available on the course webpage.

3.2 Weekly Exercises for Student Sessions

In the traditional approach, the student session was focused on discussing students' solutions to homework assignments. However, we were (and still are) convinced that students need to engage more in joint work in order to be better prepared for the actual homework assignments. In order to accomplish this, we began offering additional exercises that were meant to be solved jointly *during* the student session. The tutors responsible for these weekly *exercise* sessions encourage the attending students to participate actively in ongoing discussions and ask questions. To support the latter, we also offered quiz questions. *Quizzes* contain approximately five right-or-wrong statements related to the topics of the previous week's lecture. Answers are discussed in the first 15 minutes of the exercises session. This is supposed to help students to check their understanding and serve as a warm-up and first discussion before working with exercises.

3.3 Homework assignments

Experience with teaching introductory CS courses has shown that it is very important to make homework submissions mandatory. Otherwise students do not work regularly enough to succeed in the final exam. To encourage teamwork and reduce the student's workload homework assignments are to be solved by teams of 2-4 students without support from tutors or instructors and to be submitted for grading in written form. Each submission is checked and commented by the tutors and students receive points for each homework assignment. Students must obtain at least 50% of the possible points to be admitted to final exam.

Since the final exam is a situated activity, students must experience and learn how to deal with this as well, especially when this is their very first university exam. For this reason, we also started to offer a *pre-exam* shortly before the two weeks of Christmas break. The pre-exam has the same form and amount of assignments as the final exam, counts as one submitted homework, but does not count to the course grade. It gives the students an opportunity to practice the assessment situation and explicates what will be expected from them during the finals. The pre-exam is reviewed the same way as the finals and gives students direct feedback about their current effort and achievements.

3.4 Results of first Adjustments

All the adjustments to the course mentioned above were made permanent by the end of 2007. However, the failure rate in the final exams was still varying as table 1 shows.

Table 1: Distribution of student scores (in %) before 2012 (where 1.0 is the highest and 4.0 the lowest score corresponding to A+ and D respectively)

Grade/ Year	1.0	1.3-1.7	2.0-2.3	2.7.-3.3	3.7-4.0	failed
2008	4.0%	8.1%	11.4%	36.2%	24.2%	16.1%
2009	4.9%	7.4%	18.9%	24.6%	21.3%	23.0%
2010	18.3%	13.6%	22.5%	28.2%	9.9%	**7.5%**
2011	0%	0%	3.4%	17.6%	19.6%	**59.5%**

At a first glance this variation in grades and failure rates seemed hard to explain, since most parameters of the course were identical in all these years. However, each year a different faculty member was responsible for supervising the tutors and for designing the exercises, homework, and exams. We observed that in the years with low failure rates the problem sets in the exercises, homework, and finals were similar in nature. In 2011, the correspondence between these problem sets was a lot weaker and it seemed that this was one of the reasons why the students were less prepared for what was expected from them in the final exam.

4. DEVELOPING A PEDAGOGY

Given our student-oriented approach in teaching and a strong focus situated cognition theories, we approached the adjustments done in Theory I from a perspective of learning sciences. Here, we found the cognitive apprenticeship approach as the most helpful theory to reflect on our adjustments and improve them.

4.1 Cognitive Apprenticeship

Collins et al. [5] argue that for the longest time in human history the natural form of learning was apprenticeship: a master-students relationship that focuses on practicing contextualized knowledge and skills in authentic situations that provide meaning to the activities involved. Learning was not only a form of "acquiring" knowledge and developing skills to handle it but also a form of enculturation into a certain community. Being set in a specific workplace, tasks and problems arose not from pedagogical concerns but from the demands of the authentic environment ([4], p. 48ff). Of this master-student-relationship Collins et al. describe the following techniques to be essential:

- *Modeling,* where the teacher demonstrates how to do something and makes single steps of a process visible such that students can observe and be than able to imitate it.
- *Scaffolding & Fading* is the framework or certain plan the teacher provides students with in order to carry out specific tasks. Once students are on their own, the scaffolding framework is gradually removed in the process of fading.
- *Coaching* where the teacher guides or supervises students' activities, efforts, and experiences evaluating them, offering encouragement and feedback.

Nowadays, master-student-relationships can be found for example in learning to play an instrument or to do research as well as multiple forms of trades like plumbing or tailoring.

Collins et al. argue that typical forms of academic formal schooling are the opposite of apprenticeship and reflect mostly a view of learning that focuses on knowledge presentation ([5], p. 38-39), ([4], p. 47ff). In a lecture the teacher introduces and demonstrates knowledge as a stand-alone generalized product, which is decontextualized from the situated activities in which it was once developed and in which it is supposed to be used by the students in future. As a consequence, the expertise to create and use the knowledge remains tacit since a specific context is not emphasized. But for the members of the scientific research community who created this knowledge, it is not a decontextualized stand-alone product, but a cognitive tool they use and apply in activities where it is meaningful and relevant to them. Bereiter ([1], p. 295ff) argues that learning in formal schooling is particularly difficult for two reasons. First, there is no sufficient distinction between knowledge as the *material* or item of inquiry and knowledge and skills that are needed to handle this material; second, the latter remains mostly tacit since little attention is paid "to the reasoning and strategies that experts employ when they acquire knowledge or put it to work to solve complex or real-life tasks. […] To make real differences in students' skill, we need both to understand the nature of expert practice and to devise methods that are appropriate to learning that practice" ([5], p. 38-39).

Collins et al. argue that in contrast to craft, activities that handle "knowledge material" are in part invisible and that therefore it is almost impossible for students to observe and imitate them in the same way as in traditional apprenticeship. Still, students in higher education institutions are supposed to adopt the expertise of a particular scientific community and to be able to work with the domain knowledge introduced in a lecture. As a pedagogical approach that focuses on the gap between process and product Collins et al. suggest *cognitive apprenticeship*, [4], [5]. The idea of this approach is to focus not just on domain knowledge as a decontextualized product but to make the required skills and knowledge more explicit and the thought processes of teachers and students more visible ([5], p. 40). For this reason, they propose to use the key methods of traditional apprenticeship as well as additional methods specific to their approach ([4], p. 50ff): The teacher is supposed to explain and demonstrate knowledge and skills (modeling and articulation), which students can observe and then repeat under the guidance of the teacher (coaching). Different frameworks are proposed to the students that help them to orientate themselves and their learning activities (scaffolding). Since most of the activities are cognitive and not visible, it is important that teachers and students develop the ability to articulate and reflect their activities.

Another important aspect of cognitive apprenticeship is providing the context in which domain knowledge is meaningful. In the workspace of traditional apprenticeship, reasons for specific activities are much better understood than in formal schooling: students "are motivated to work and to learn the subcomponents of the task, because […] they have seen the expert's model of the finished product, and so the subcomponents of the task make sense. But in school, teachers are working with a curriculum centered around reading, writing, science, math, history, etc. that is, in large part, separated from what students and most adults do in their lives. In cognitive apprenticeship, the challenge is to situate the abstract tasks of the school curriculum in contexts that make sense to students" ([4], p. 50).

4.2 Improving the Adjustments

The Cognitive Apprenticeship approach helped us better understand how our adjustments could support our students in their learning efforts and how we could improve them to create a pedagogical approach for the Theory I course.

4.2.1 Modeling

The Cognitive Apprenticeship approach helped us understand that with a course setup as described in section 2.3 the emphasis lies mostly on presenting theoretical CS domain knowledge (the concepts, definitions, algorithms, etc.) as stand-alone products and does not sufficiently explicate the methods, approaches, and strategies of dealing with this knowledge as well as techniques how to learn them. Since all lecture topics are prepared in advance for a smooth presentation, students experience them as knowledge products without seeing the enormous effort it took to create them. Skills to handle and work with this knowledge that students are supposed to develop in the course are only implicitly

demonstrated during the lecture and are not addressed and exposed sufficiently. Therefore, we started to focus on theory practices and demonstrate them more explicitly with regard to *modeling*. While the lecture presents the ready-made "products" of theoretical CS, the tutorial is the natural place in the course where the process of creating them is shown. So, we understood that the tutorial is mostly helpful to students when it serves as a modeling session where students can experience that even a professor has to try different approaches and alternatives before accomplishing a solution.

4.2.2 Scaffolding & Fading

Our main idea was to use the exercise sessions to enable students to solve their homework by themselves and to motivate them to become more actively involved in the exercise and tutorial sessions. Therefore, we redesigned the weekly exercise sessions and assignments entirely with regard to the *scaffolding & fading* method.

We offered specific *preparatory exercises* that are to be solved jointly during the exercise session and serve as preparation for homework. After the exercise sessions we also posted detailed written solutions to the preparatory exercises on the web pages. These solutions demonstrate what a correct solution should look like *in written* form and what is expected from the students when they turn in their homework. Especially for the latter there is not enough time during the exercise session, since possible solutions to the exercises are developed together and only sketched on the blackboard. But students must also learn how to produce a complete and precisely formulated solution using mathematical formalisms.

Redesigning the exercise sessions, the major change was to align the preparatory exercises with the homework with respect to structure and content. This means that in each session the same type and amount of problems is used for the exercises as well as for the homework assignments. This way, it makes sense for students to work with the preparatory exercises, to participate during tutorial and exercises sessions, since all these activities prepare them to do their homework assignments.

In the traditional approach, it was assumed that students gain understanding from attending the lecture. The cognitive apprenticeship approach taught us that cognitive methods and strategies used in theoretical CS needs to be explicated. The alignment between exercises and homework together with the written solutions to the exercises is the *scaffolding* framework for students. However, this framework's objective is to get students more self-reliant in working with the theory of computation. Therefore, in the second part of the course, we gradually removed the strong alignment between preparatory exercises and homework. In addition, we also started reducing detailed information in written solutions.

4.2.3 Coaching

The weekly homework grading as well as the tutors' support during the exercise session is a form of *coaching*. In addition, it corresponds to students' activities during final exam, where students have to formulate a written solution for an assignment, which then will be graded by tutors under supervision of the instructors. In order to understand the expectations, especially with respect to the very strict and formal character of solutions to assignments, students need to train this skill as well and to receive a weekly feedback about their efforts.

4.3 The Final Exam

In the Theory I course, the measure of students' learning success is a student's performance on homework, pre-exam, and final assessment. The points or grade students obtain are supposed to measure the learning outcome see sec. 2.1 and [9]. We are aware that a valid measurement can only be achieved by methods and tools that diagnose the defined competences in ([9], p. 35-38). However, these competences have been defined normatively by the Department of CS and lack a competences model as well as diagnostic methods. Therefore, we rely on assignments that we developed in the past years and that represent typical assignments for a Theory I course.

The course staff recorded student performance on each individual question on the homework assignments as well as performance on each question on the practice and final exam. During the whole course 14 homework exercises (8 before and 6 after the Christmas break) were offered, each with 3 mandatory and one more difficult optional assignment. For each assignment students could score 1-3 points and altogether they needed at least 50% of all points from obligatory assignments in order to attend finals. Most of the assignments have been used in similar form in the past five years and included only minor alterations, like exchanging a concrete language or Turing machine.

The final exam contained 15 simple questions focusing on definitions and methods knowledge and 5 assignments with 3-7 sub-assignments, which are aligned with the homework assignments students are already familiar with. Each sub-assignment in the exam assesses one particular skill without relying on results from previous sub-assignments. In addition, we use typical automata, regular languages or Turing machines with a simple structure and without containing unusual cases. Each of the 15 simple questions has been used for final exams in the last five years. Therefore the requirements of the final exam correspond to the requirements of the last five years.

Table 2: Distribution of student scores in 2012 and 2013

grade	1.0	1.3-1.7	2.0-2.3	2.7.-3.3	3.7-4.0	failed
2012	10.7%	20.5%	30.2%	22.9%	9.8%	**6.3%**
2013	1.9%	8.3%	25.5%	34.4%	20.4%	**9.6%**

Among the 205 students attending the finals during winter semester 2011/2012 only 6% of the students failed to pass. Among the 157 students attending the finals one year later only 9.6% failed to pass, see Table 2. Altogether, all student grades are evenly distributed. This finding strengthens the assertion that the course's final exam assessed student performance in a similar way it did in the previous years. Note that in 2013 there is a reduction in better grade due to slightly increased requirements for receiving a straight 1.0 grade (i.e. A+) in the final exam. These requirements were based on our belief that top students should demonstrate the ability of "thinking out of the box". The design of the final exam made sure that the requirements for passing remained unchanged.

5. CONCLUSION

In this paper we introduced a pedagogical approach for a theory of computation course at the University of Potsdam based on cognitive apprenticeship. Our goal was to make the practices of

theoretical CS more visible to the students and therefore easier to adopt. We reinforced modeling by introducing a tutorial session; we scaffold students' activities through a strong alignment between exercises, homework, and the final exam; and we tried to coach students' activities in the exercise sessions and by providing feedback for their homework submissions. At the same time, we kept the requirements for final exams comparable to those of the last 5 years and had a failure rate below 10% two years in a row. Therefore our approach demonstrates that it is possible to enhance students' success and reduce failure rates in a theoretical CS course while keeping the requirements high.

Most students who attended the course during the last two winter semesters appeared regularly during exercise sessions and participated actively during the quiz and working with the exercises. The tutorial was attended by more students than in the years before, when we did not align homework assignments with the exercises. During the tutorial, students asked very precise and concrete questions related to the contents of the lectures and the exercises. They also asked for solutions to the more difficult assignments already submitted. In the years before, students mostly asked questions about the *current* homework assignments, which were difficult to answer without providing solutions to tasks still ahead of them. With the alignment between homework and exercises, students seemed to be more successful in solving homework assignments by themselves once they had attended the exercise sessions, seen a written solution to the exercises, and discussed the more difficult aspects in the tutorial. Altogether, our impression was that our changes helped students to be more focused, to stay motivated, and to keep working on their weekly assignments.

We were not yet able to accomplish all aspects of cognitive apprenticeship. For example, it is important that students articulate their activities and reflect on them ([4], p. 51). Since we cannot change the teacher-learner-ratio and introduce more study sessions, most of students' activities can be only modeled and scaffold. The rest has to happen during their individual studies. We only have little influence on the latter and know that when students manage to find an adequate study group they perform very well. But a lot of students are not able to do this and besides encouraging them to look for a study group, we have not found a method how to support this better. Furthermore, it remains open if and how to situate ([4], p. 52ff) the course topics and abstract tasks of the weekly assignments in contexts that make sense to the students. In its role as a cognitive tool the domain knowledge of theoretical CS will become meaningful for them as they rely on it during their whole professional career. But it remains open how this can be more explicitly highlighted during the course. Altogether, it is important for us to further elaborate on these issues in future.

Since none of our modifications to the course were specific to the University of Potsdam or to the topic of the course we believe that our approach could be also applied in introductory courses to theory of computation at other departments of CS as well as be adapted to other introductory courses in CS in order to contribute to student learning success.

REFERENCES

[1] Bereiter, C. 1997. Situated cognition and how to overcome it. In Situated cognition: Social, semiotic, and psychological perspectives, Kirshner, D. and Whitson, J. A., Eds. NJ: Erlbaum, Hillsdale, 281-300.

[2] Caspersen, M. and Bennedsen, J. 2007. Instructional design of a programming course: a learning theoretic approach. In Proceedings of the third international workshop on Computing education research (ICER '07). ACM, New York, NY, USA, 111-122

[3] Chesñevar, C. I., González, M. P., and Maguitman, A. G. 2004. Didactic strategies for promoting significant learning in formal languages and automata theory. In Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '04). ACM, 7-11.

[4] Collins, A. 2006. Cognitive apprenticeship. In *Cambridge Handbook of the Learning Sciences*, Sawyer, R. K., Ed. Cambridge University Press, 47-60.

[5] Collins, A., Brown, J. S., and Holum, A. 1991. Cognitive apprenticeship: Making thinking visible. American Educator, 6(11), 38-46.

[6] Hämäläinen, W. 2004. Problem-based learning of theoretical computer science. In Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference, S1H/1 -S1H/6 Vol. 3.

[7] Heublein, U., Richter, J., Schmelzer, R. and Sommer, D. 2012. Die Entwicklung der Schwund- und Studienabbruch-quoten an den deutschen Hochschulen, HIS: Forum Hochschule 3/2012.

[8] Korte, L., Anderson, S., Pain, H., and Good, J. 2007. Learning by game-building: a novel approach to theoretical computer science education. In Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '07). ACM, 53-57.

[9] Modulhandbuch für den Bachelor- und Masterstudiengang Informatik an der Universität Potsdam. 2011. www.uni-potsdam.de/fileadmin/projects/mnfakul/assets/Studium/Modulhandbuch_Informatik.pdf

[10] Robins, A. 2010. Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, *20*(1), 37–71.

[11] Rodger, S. H., Bressler, B., Finley, T., and Reading, S. 2006. Turning automata theory into a hands-on course. In Proceedings of the 37th SIGCSE technical symposium on Computer science education (SIGCSE '06). ACM, 379-383.

[12] Ventura, P. R. 2005. Identifying predictors of success for an objects-frst CS1. In: Computer Science Education 15, 3, 223-243.