Didactic Redesign of a University Course on Algorithms and Data Structures

Torsten Brinda, torsten.brinda@informatik.uni-erlangen.de
University of Erlangen-Nuremberg, Didactics of Informatics, Germany

Abstract
Many informatics study programs at universities worldwide start with a course, in which “algorithms and data structures” (AD) are introduced together with and by means of a concrete imperative, object-oriented or functional programming language. In this paper a historically grown concept of the AD course at the University of Erlangen-Nuremberg in Germany is analyzed from a didactic perspective and the didactic redesign of this course is described. A first implementation of the new approach is made in summer semester 2011.

Keywords
Informatics education, higher education, algorithms, data structures, instructional design

MOTIVATION
Many informatics study programs at universities worldwide start with a course, in which “algorithms and data structures” (AD) are introduced together with and by means of a concrete imperative, object-oriented or functional programming language. This is also the case at the University of Erlangen-Nuremberg in Germany, where in a large lecture with currently up to about 350 students algorithms and data structures are taught using the programming language Java. The audience is made up of students from a variety of study programs, such as informatics, computational engineering, information and communication technology, informatics teacher degree programs, business informatics, and medical technology. Some of these courses have informatics as a focus, others as a minor subject. This results in a broad spectrum of pre-knowledge and interests of listeners of the AD course. The author of this paper has been working at the University of Erlangen-Nuremberg since spring 2005 and since then he observed periodically recurring patterns of problems in the context of this course. Students with little or no programming skills obviously found it very difficult to follow the course, whereas students with sound programming knowledge seemed to have no major difficulties. This was also reflected in the results of student course evaluations at the end of each semester. The lecture is currently read by four different informatics professors, a different one each semester. When asked to read the lecture in summer semester 2011, the author decided to agree, and to thoroughly analyze and if necessary to didactically revise the entire course conception.

PREVIOUS COURSE CONCEPT
The course consists of four lecture and four lab units per semester week (units of 45 mins). In the past, a set of about 1400 lecture slides was developed for the lecture and adjusted annually at certain points. The course rotates currently each term between four professors and all use the same set of slides. In the past, the lecture was only offered in winter semesters, but currently it is read each semester to allow students to begin their studies also in summer semesters. Each semester week an exercise sheet is published, with which the lecture material of the week should be practiced. To complete the course the students have to pass a written exam of 120 mins and to reach 60% of the overall available points of the exercise sheets. One half of the weekly four lab units are two black board lab units, which take place in
small groups of 20 to 30 students and which are led by a (usually student) tutor. In these black board labs the weekly homework is discussed and presented. In the other two lab units, the computer labs, the students work in a computer pool on programming homework and can, if required, contact a present tutor to clarify any technical or programming related problems. In the past, it was often a problem that too few tutors were available in these labs. Because of the organizational effort and the experiences of the past, the labs are organized and coordinated by a stable team regardless of whether the lecturer changes. To meet the identified difficulties of programming novices, in the past a one-day course about programming basics for beginners (up to variables and control structures) was installed before the official semester start. Moreover, an extra weekly lab was offered, in which programming examples were discussed and questions of students could be clarified.

**DIDACTIC ANALYSIS AND REDESIGN OF THE LECTURE CONCEPT**

A large number of teaching approaches for the design of introductory AD courses can be found in the literature, e.g. in the conference series of the ACM special interest group for computer science education, so that it would possibly have been easy to take over a concept described there. The problem was that further lectures of various study programs are built upon the given course conception. So, for acceptance reasons a complete renewal of the lecture was not considered, but strategies designed to enhance the existing course. The student course evaluations showed that the lecture slides together with a lecture of the main producer of the slides were evaluated positively, while other faculty were evaluated clearly worse with the same set of slides. Students also reported that the slides “are an excellent reference material, once one has understood it”. All this suggested that the course material had to be optimized in the present form in relation to the learning process and in terms of its suitability as a self-learning material.

**Inductive and problem-oriented approach**

In the lecture slides concepts of algorithms and data structures and the language Java were presented, occasionally with example fragments. In the exercises the students were expected to solve problems, which was not or too little taught in the lecture. Many students said, they did not learn how to develop a program from a given problem. To deal with this, the slides were converted to an inductive approach and the theory derived from given problems.

**Order of content**

Java basics were only marginally taught and distributed over several chapters of the first third of the course. In the labs, however, the students were required to program in Java from the first semester week. That led to difficulties. In the evaluations, the students wanted an earlier and more detailed introduction into the programming language. This led to a rearrangement of the lecture contents. The lecture starts now with algorithmic thinking and the algorithmic description of solutions of real life problems (e.g. sorting a set of given, unknown weights only by the use of a balance by weight). After that, an introduction into the Java syntax and algorithmic control structures follows by means of the learning environment JavaKara [1] (a robot carol derivative). Stand-alone Java programs follow after that.

**Structure of the lecture units**

The analysis of the lecture slides showed that it is a historically grown slide set mainly oriented and structured by technical criteria. For example, the lecture consisted of units such as the UML notation, recursion, abstract data types (ADTs), lists, trees, graphs, and algorithms based on them. From an educational viewpoint, it seemed more appropriate to integrate the UML notation and recursion in the other chapters and to introduce them, where a first use was required. In the chapter on
ADTs the ADT concept was introduced relatively abstract using the data structures described in detail in later chapters. In the course evaluations the students reported to have had understanding difficulties in the ADT context. This suggests to integrate the ADT theory into the units about the data structures and to talk about ADT concepts where it fits contentually (see Fig. 1).

Figure 1: Rearranging 1400 lecture slides according to didactic criteria

DIDACTIC ANALYSIS OF THE LAB CONCEPT
The didactic analysis of the lab concept also uncovered a number of problem areas.

Technical problems
Students without programming experience often had problems at the semester start to install and to configure the required software on their respective operating systems and to successfully compile and execute at least one example project successfully using a recommended programming environment. Tutors had to help. The students on the other hand were losing valuable learning time.

Coordination of lecture units and related labs
In the course evaluations it was reported that there were often discrepancies between the learning outcomes from the lectures and the expected prior knowledge of the related labs. In the labs obviously sometimes knowledge was expected, that the students could not have by just attending the lecture. This was due to a partially sub-optimal coordination between lectures and labs.

Workload through the homework and the use of a plagiarism checker
The weekly homework consisted of theory and programming tasks. All solutions were to be submitted electronically via an exercise submission tool (EST). The programming tasks were tested automatically for compilability and the fulfilment of pre-defined test cases. From about the 4th semester week additionally a plagiarism checker was used to identify too similar submissions. Too similar solutions were
assumed to be plagiats and each assigned 0 points. The students knew about the plagiarism checker from the beginning of the semester.

This led to the consequence that subject-related communication about the tasks and solution ideas between the students was hardly possible, because especially the higher-performing students were afraid to produce plagiats and to get 0 points for their particular solutions. The reasoning of the lab organizers was that in the final exams all students are required to produce individual solutions. From a didactic point of view this argument is problematic. From a constructivist learning point of view learning takes place a lot in cooperative discussing and developing solution ideas. As a consequence, novice programmers had to try to find programming solutions all by themselves and in a very tedious way and reported of up to 20 hours to complete a single exercise sheet. A significant number of students moved to other study programs as a consequence.

Course of black board labs
In the black board labs the students were mostly receptive. A solution to last week’s exercise sheet was presented by the tutor and questions of students to the next sheet answered by him.

REVISED LAB CONCEPT
Based upon this analysis changes to the lab concept were made in informing the students, the design of homework and the didactical design of the black board labs, following university didactic recommendations [2].

Setting up the personal programming infrastructure
In order to avoid difficulties in setting up the personal programming infrastructure required by the course, at the begin of the semester screencasts are offered for the different operating systems, in which download, installation and configuration of the Java Development Kit (JDK) [3] and the Eclipse programming environment [4] are demonstrated. Moreover, it is also presented, how a given Java file can be compiled using the Java compiler and then be executed. Finally it is also shown, how a new Eclipse project can be created, a Java file added, compiled and executed. This relieves the labs, in which this was partially presented in the past, as well as the tutors, who were also asked for help when problems occurred.

Definition of learning objectives for lecture units and labs
To improve the lack of coordination between lectures and labs, all course units were provided with detailed learning objectives, both to make the students clear what is expected of them and to give the lab coordinators orientation in the creation of exercise sheets. In addition, the lecturer is now included in the release of an exercise sheet.

Subject-related communication and cooperation
The initial difficulties and the exploding workload by the fear of plagiarism especially for novice programmers led to reflect on group solutions for the exercise sheets. Group tasks provide the possibility of subject-related communication and cooperation. Especially novice programmers can benefit from such tasks. Critics fear that one can of course only control to a certain extend, who did the work in a group. High-achievement students could solve the exercise sheets in weekly change and thereby reduce their effort. In a similar lecture at the University of Osnabrücke in Germany for example, each student team has to present its solution to a tutor in a half hour session. The result is that both team members must be well versed in the particular solution, however, it requires a significant number of student tutors, which is costly. Any form of mandatory presentations in the lab sessions leads to a
corresponding attendance duty, which was seen as problematic. On the other hand, it is also a relevant learning objective to lead the students to independent programming. The solution idea arose from the hypothesis that at the beginning of term, the need for exchange is higher than at the end of the semester, when the students have made the necessary experience. Consequently, it is planned to divide the weekly exercise sheets now in individual and group tasks. Individual tasks are to be solved individually. For the group tasks teams are formed consisting of 2 persons in the first black board labs. The plagiarism checker is still used from about the fourth week of the semester between all individual and all group solutions. The shares of group and individual tasks per sheet change over the semester from "more group tasks" at the beginning to "more individual tasks" at the end. Overall, 50% of the tasks are individual and 50% are group tasks. To successfully complete the course, it is necessary to achieve each 60% of the points achievable in the individual and the group tasks. In addition to the educational reasons, a further advantage of this solution is that the correction effort per tutor decreases about 30%. The released time will be used to increase the presence of tutors in the computer labs by 30%.

Activation of the students in the black board exercises
It is is planned to experiment with two standard patterns of the 90 black board lab minutes. In the standard model students present a solution of the last sheet in the plenary. The tutor gives additional information if required. That is the extent of about 60 minutes. An activity promoting alternative provides for the random formation of teams of three, within which the students present their respective solutions each other and compare them. This will activate many students and professional communication is encouraged and required. If the comparison is completed, the tutor uses the remainder of the first 60 minutes to respond to questions of students and to demonstrate particular aspects based on the expected solution. The first half of the remaining 30 minutes are used to give students in small groups the opportunity to think through and to discuss the tasks of the new exercise sheet. In the remaining 15 minutes, questions can be addressed to the tutor, and he may provide helpful information. The two patterns for the black board exercises will be used depending on the suitability of the tasks.

OUTLOOK
The first term of the AD course with the modified concept takes place in summer semester 2011. If it ends positive and leads to appropriate results in the student course evaluations and in the final exam results, it is planned to take over the modified approach in regular operation.

REFERENCES
Biography

Torsten Brinda studied Informatics at the University of Dortmund (Germany) until 1998 before he worked as a scientist in the “Didactics of Informatics” groups at the Universities of Dortmund (until 2002) and Siegen (2002 to 2005). In 2005 he became an associate professor for “Didactics of Informatics” at the University of Erlangen-Nuremberg. He is the chairman of IFIP WG 3.2 “Informatics and ICT in Higher Education” and a member of IFIP WG 3.1 (secondary education).

Copyright
This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivative Works 3.0 unported License. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/3.0/.