REDESIGNING INTRODUCTORY COMPUTER SCIENCE COURSES TO USE TUTORIAL-BASED LEARNING

Erno Lokkila[1], Erkki Kaila[1], Ville Karavirta[1], Tapio Salakoski[1], Mikko-Jussi Laakso[1]

[1]University of Turku (FINLAND)

Abstract

In the beginning of the studies, it is important to get students motivated and interested in the subject they are studying. Getting them past the first courses with good basic understanding of the topic provides them with a fruitful foundation for success in the rest of their education. To reach these somewhat high goals, we redesigned our two introductory courses to computer science. The courses Introduction to Computer Science I and II provide student with the basic understanding of algorithms, programming, and the functioning of the computer when a program is executed in the machine. At the heart of our redesign is the tutorial-based learning approach we have successfully used on other courses. The courses are arranged around weekly topics, each of which are covered by a teacher-lead lecture, student-focused tutorial worked out in pairs, as well as individual exercises. Moreover, we discuss the technological solutions behind the automated assessment systems used on the courses in order to make the teacher workload manageable. In this paper, we describe the ViLLE environment used on the courses, the redesigned courses and their content, as well as the response they got from the students.

Keywords: Automatic assessment, Tutorial-based learning.

1 INTRODUCTION

Introductory programming courses are often considered difficult and uninteresting by students. However, as introductory courses, they provide students with their first touch to the world of Computer Science. Providing students with a positive experience is vital in order to encourage students to keep studying. The current preferred method of teaching undergraduate programming courses remains to be lecturing wherein students passively sit in lectures and listen to the orator. This method forces students to memorize facts and leaves them without an idea of what Computer Science truly is: problem solving.

Due to a curriculum change, we were presented with a chance to redesign our introductory courses to computer science. In order to promote a more active learning environment, tutorial-based learning was employed to the course. The reason why we chose tutorials was two-fold: firstly, tutorial-based learning provides students a chance to actively solve problems, much like those seen in the industry, albeit on a much smaller scale. Secondly, our experiences with tutorial-based learning on programming courses have been very encouraging.

First the ViLLE learning platform is shortly described, as it played a central role in the redesign. Then the different aspects of the courses and its elements are described; automatically assessed exercises, tutorials and the electronic examination. Finally, student perceptions of the new courses are analyzed using the student feedback collected at the end of both courses.

2 RELATED WORKS

Active learning has been defined as any activity in which the student takes part actively in order to form a solution [1]. This active learning approach has been employed on other CS courses. A meta-analysis studying the effects of an active learning approach reports decreasing drop rates and increased test scores [1]. Active learning is an integral part in the constructivist approach to learning, wherein students actively assimilate new information with what they currently know. In an ideal constructivist-learning situation, this new information creates a mental conflict with the old knowledge forcing the student to critically assess the new information and assimilate this with any old information. Any old information the student has thus creates a scaffold for which the student may construct new information [2].

Immediate feedback is a necessary part of the tutorial-based learning approach. It has been found to improve student performance and raise engagement with the task at hand[3][4]. Immediate feedback is one useful tool in the constructivist approach: feedback can assist in creating both the mental conflicts and the informational scaffolds for the student [2].

Another useful tool in the constructivist approach to learning is collaboration. Collaboration has been found to be highly beneficial in supporting the learning process of students when learning programming. [3][5][6]. This is because collaboration, by its nature, is active learning: students are working together in order to find answers to a problem. Around 60% of the discussion between students when solving a problem is about the problem itself [5].

Moving away from lectures and utilizing a more active learning-approach is described to improve results. Kay et al. managed to improve results somewhat by utilizing a Problem-Based Learning (PBL) approach to introductory courses. Their approach was to provide students with large, open-ended problems from the real world. Their full-fledged move to PBL decreased failure rates. [7]

3 VILLE

ViLLE is an exercise-based, collaborative learning environment. It contains a myriad of automatically assessed exercise types for computer science, language and mathematics learning. In addition, there are several exercises that can be used to teach any topic. All exercises provide immediate feedback to support constructivist learning. ViLLE supports both, teacher and student collaboration. Teachers can share their courses and exercises with other registered teachers, and students can solve assignments in collaboration with other students.

ViLLE is currently utilized by more than 1,200 teachers and over 15,000 students around the world. A complete description of the tool can be found in [8].

4 REDESIGNING THE COURSES

The redesigned courses were introductory courses to computer science, aptly named Introduction to computer science I and II. These courses are intended to be taken as the first real computer science courses for new students and ICS1 is obligatory for all major and minor students in IT. The number of participants is, as a result, relatively high especially on the first course.

The old versions of the courses were typical introductory courses to CS, taught completely by lecturing. The lectures were given twice a week for 3 weeks and a final exam at the end of the course determined the final grade for students. Main topics covered during the first course were basic concepts in computing (e.g. variable, program execution, method and so on) and number systems used in computing. The second course concentrated on deepening the programming skills of the students by introducing e.g. recursion and non-imperative programming.

The new versions of the courses were designed with active learning in mind. The new version of the course lasted twice as long, and consisted of 12h of lectures and 12h of tutorials. The lecture introduced the theory behind the topic for the week, and the tutorial deepened the students' understanding of the topic discussed on the lecture. The final grade for the students was given based on a final exam, as in the old course. However, the exam in new course was conducted electronically using the ViLLE system instead of the more traditional pen-and-paper method utilized on the old course. See Table 1 for a side-by-side comparison of the structure of the courses.

Table 1: Comparison of the structure of the old and redesigned courses

Component	Old CS1 course	Old CS2 course	New CS1 course	New CS2 course
Study points	2	3	3	5
Lectures	12h	20h	12h	16h
Tutorials	None	None	12h	16h
Final Exam	pen-and-paper	pen-and-paper	electronic, using ViLLE	electronic, using ViLLE

Feedback collected	None	None	2 short weekly surveys + final course feedback	2 short weekly surveys + final course feedback

Overall, more taught content was present in the redesigned courses when compared to the old courses to justify the increase in study points. The aim of these redesigned courses was to teach basic understanding of algorithms, programming and the operation of computers in general; how they work, what they can and can not do. Additional emphasis is given to how programs are translated to machine language and then executed. Previously, before the redesign, the main focus on these courses was to teach the basic concepts in computing and to provide an understanding of the technological foundation for modern computing. Due to the increase in content and study points, more time was also allotted for the courses. The extent of the changes performed on the courses makes a scientific comparison between the two versions impossible. Instead, the focus will be on evaluating the feedback received from students regarding the new methods and the course as a whole.

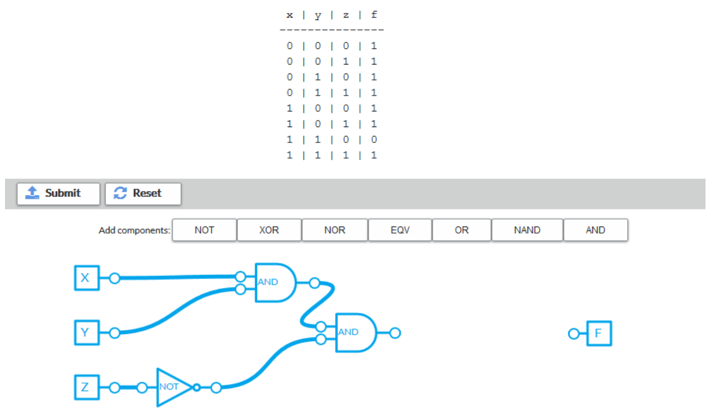
4.1 Elements of the redesign

Automatic assessment played a major role in the redesign. Automatic assessment of exercises allowed the teaching staff to include more exercises to the courses with no additional resources spent on grading them. Furthermore, students also received immediate feedback on each and every exercise they submitted to the ViLLE system. This feedback consisted of the correctness of the student's answer, an example of the correct answer and possibly additional information about what to do in order to achieve a higher score. ViLLE offers a myriad of exercise types, several of which were used on the redesigned course. All the exercises were automatically assessed. These automatically assessed exercises were employed in both the tutorials and the electronic exam. The rest of this chapter concentrates first on describing the exercises used on the course, followed by how they were utilized in the tutorials and examination.

4.2 Exercises

ViLLE offers teachers a bountiful selection of exercise types, most of which are automatically assessed and can be parametrized to provide students an assignment with different values each time. The exercise types used on the redesigned course mainly consisted of coding exercises, Parson's problems and quizzes, although many other types of exercises were included for variety so students' engagement was maintained. The only manually graded exercise on these courses was a short essay in the final exam.

Because ICS2 also discusses computers on the hardware level, the logic gate exercise type (Figure 1) was utilized on the course. The logic gate exercise allows students to construct their own logic circuit and simulate its operation. The feedback given by the exercise consists of a visual description of the input sets for which the circuit produced the correct output and conversely, the wrong output.

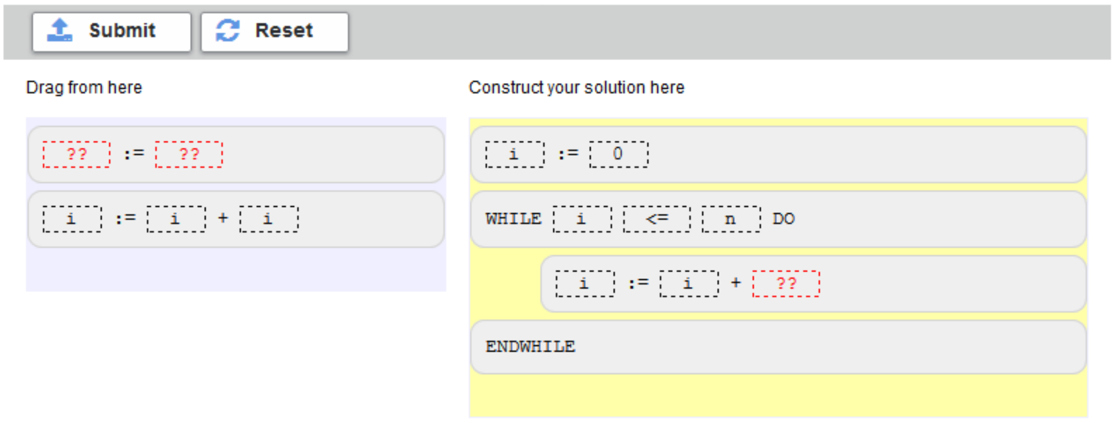


```
x | y | z | f
---------------
0 | 0 | 0 | 1
0 | 0 | 1 | 1
0 | 1 | 0 | 1
0 | 1 | 1 | 1
1 | 0 | 0 | 1
1 | 0 | 1 | 1
1 | 1 | 0 | 0
1 | 1 | 1 | 1
```

Figure 1: Logic gate exercise

The coding exercises used on the course allow students to either write their own program code or simulate the internal state of a computer during a given program. Both these exercise types provide the student with feedback. In the coding exercise, the given feedback consists of either an authentic compiler error if the code did not compile, or the output of the written program shown side-by-side with the output from the teacher's solution. The program simulation exercise notifies the user whenever the program state is incorrect after each line of program code. The aim in these exercises is to facilitate active learning by automatically showing the students where they erred and the possibility to improve on their answer until a satisfactory solution is found.

Another exercise type that was extensively used on the new courses was Parson's programming puzzles [9]. This exercise type was used to teach students both python and microcode. While we also used 'normal' Parson's problems in our tutorials, we also used a slightly modified version of the puzzles: students were not only to drag and drop lines of code to create a working program, but also had to select the variables and some of the operators used in these lines. This modified version of the puzzle (Figure 2) placed students responsible for not only the logic of the algorithm, but also of the internal state of the computer during the execution of the algorithm – all while providing a controlled environment where students learn good coding habits.



```
?? := ??
i := i + i
```

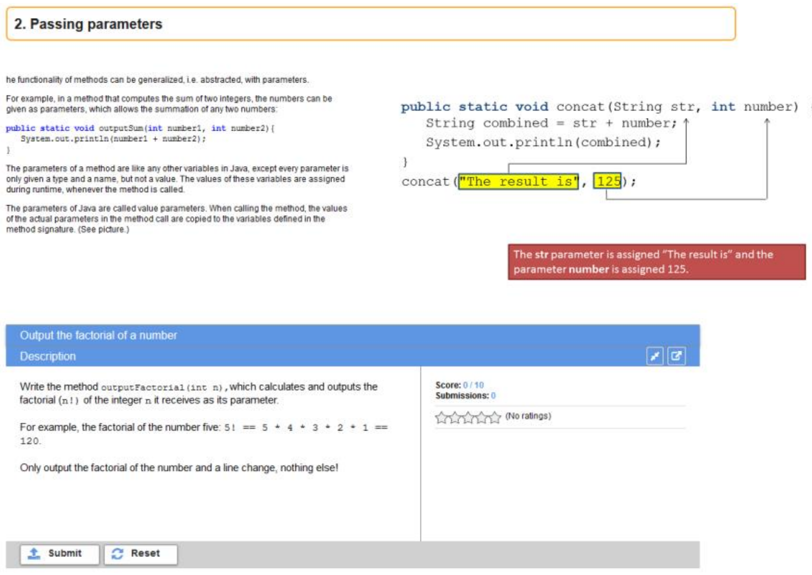
```
i := 0
WHILE i <= n DO
    i := i + ??
ENDWHILE
```

Figure 2: Modified Parson's Puzzle

In addition to these three exercise types, a variety of other exercise types were used on the courses. The types most often used were Matrix selection exercises and quizzes. Calculate-in-a-column exercises were used to teach binary addition, subtraction and multiplication. All exercise types are explained in more detail in the Book of ViLLE (see [10]).

4.3 Tutorials

Tutorials on the redesigned courses consisted of a combination of exercises and the related study material. Students completed the tutorials on their own laptops, or alternatively paired up with someone who had one. The students formed pairs to complete all tutorials, as collaboration is an important aspect in active learning. The tutorials were linked to lectures in such a way that the week's lecture provided with the theoretical and general background and the tutorial gave assignments and hands-on training. Tutorials provided all the necessary information to solve the given assignments without giving direct answers, as shown in Figure 3. Additionally, the lecturer and several mentors (i.e. older students) were present at the tutorials to provide assistance to those having trouble with the assignments or the ViLLE system.


2. Passing parameters

he functionality of methods can be generalized, i.e. abstracted, with parameters.

For example, in a method that computes the sum of two integers, the numbers can be given as parameters, which allows the summation of any two numbers:

```
public static void outputSum(int number1, int number2){
   System.out.println(number1 + number2);
}
```

The parameters of a method are like any other variables in Java, except every parameter is only given a type and a name, but not a value. The values of these variables are assigned during runtime, whenever the method is called.

The parameters of Java are called value parameters. When calling the method, the values of the actual parameters in the method call are copied to the variables defined in the method signature. (See picture.)

```
public static void concat(String str, int number) {
   String combined = str + number;
   System.out.println(combined);
}
concat("The result is", 125);
```

The **str** parameter is assigned "The result is" and the parameter **number** is assigned 125.

Output the factorial of a number

Description

Write the method `outputFactorial(int n)`, which calculates and outputs the factorial (`n!`) of the integer `n` it receives as its parameter.

For example, the factorial of the number five: `5! == 5 * 4 * 3 * 2 * 1 == 120`.

Only output the factorial of the number and a line change, nothing else!

Score: 0 / 10
Submissions: 0

(No ratings)

Figure 3: Example of a part of a ViLLE Tutorial translated into English

In order to successfully organize these tutorials, several points must be kept in mind. The tutorials must be kept in a spacious hall. All our tutorials were held in a 250-seat lecture room, which made it possible to have students sit on every other row so the teaching staff was able to reach every student – even those sitting in the middle of the row. Additionally, students should be provided with Ethernet cables. This allowed us to provide everyone with a stable internet connection, as the Wi-Fi routers present were not able to handle the connections from all devices present (cell phones, tablets, students' laptops, etc.). Furthermore, using the local area network of the University made it possible to block access to all sites but ViLLE. This was not strictly required for normal tutorials, but was an extremely useful preparation for the exam.

4.4 Examination

The final examination for the course was held as an electronic examination using the ViLLE learning system. The examination was held online, which meant students were allowed to bring their own computer and complete the examination on it. For the students who did not own a laptop, access to the internet was provided from the computer lab. The examination was held in the same lecture hall as the tutorials, which meant we were able to block all unwanted internet sites. Nevertheless, as students used their own computers, the possibility of course materials present on the computer was taken into account by having several examiners, who made sure no one cheated.

The examination itself consisted of the same exercise types students were already familiar with from the tutorials, only with slightly harder parameters. As all the assignments, apart from one short essay, were automatically assessed, students received their final course marks in record time.

5 STUDENT FEEDBACK

Student feedback was collected extensively during both courses. For this paper, only the final course feedback is presented, along with a few hand-picked comments from students. The course feedback consisted of general questions about the course and students answered them on a Likert-scale of 1 to 4, where 1 is very negative and 4 is very positive. Table 2 presents the results.

Table 1: Student feedback on the new courses

Question	ICS1 (n=100)	ICS2 (n=75)
1. Rate the lectures as a whole	3,15	2,71
2. Rate the tutorials as a whole	3,14	2,92
3. Rate the course homework as a whole	3,17	2,68

4. Rate the course as a whole	3,32	2,8
5. The course was too easy	2,27	1,65
6. The learning method (lectures+tutorials) suited me	3,28	3,09
7. Tutorial-based learning should be used in the future	3,47	3,47
8. There were too many exercises	1,98	2,03
9. The tutorial exercises had too much variety	1,72	2,04
10. I achieved the learning goals I set for myself	3,36	2,53

The results clearly show that the first course was easier than the second. This was an expected result, as the ICS1 is obligatory to both minor and major students, while ICS2 only to majors. ICS1 gives a shallow, but broad view of computer science in general, whereas ICS2 deepens understanding of the topics learned in the previous course while introducing additional topics. The hardness of ICS2 is reflected in the overall grades: all but one question received lower scores. Interestingly, this question is number 7, regarding the continued use of tutorial-based learning. Students thus enjoy tutorials and regard them as an effective learning method regardless of the difficulty of the course.

Students not only preferred the tutorial-based learning methods used, but also felt lectures worked well when complementing the tutorials. Tutorials coupled with lectures as a learning method was generally accepted as a welcome teaching method. This is shown by the well-above average answers to question 6, which was rated over 3 on both courses.

A variety of exercise types is a welcome addition to courses, as shown by student answers to questions 8 and 9. Both courses utilized the same types of exercises. However, the students' rating for question 9 - regarding exercise variety - rises. This indicates that students became familiar with the exercise types used and their engagement with the exercises dropped as a result.

6 CONCLUSIONS

While this course redesign well received by the students, the reasons for the noticeable lower scores for the second course are rather unclear. The most likely reason for the decrease in the scores is the difficulty of the course; more advanced, and thus more difficult, topics were discussed. Due to this added difficulty, more work was required by the students and this would naturally cause students to like the course less. Another reason for the lower scores could be due to the courses having different teaching staff.

Students enjoyed the tutorials and the variety of exercises. As some skills are best learning by drilling [9], having different types of exercises train the same skill, students are less likely to get bored and give up, thus learning the skill being taught better.

In the future, we plan to keep gathering data from future instances for a more thorough scientific analysis of the benefits of moving to active-learning based teaching. The data gathered will include not only students' feedback on the course, but also exam results and other course statistics, such as points gathered. This data should allow us to analyze the effectiveness of an active learning based approach to student performance.

REFERENCES

[1] S. Freeman, et al. "Active learning increases student performance in science, engineering, and mathematics." *Proceedings of the National Academy of Sciences* 111.23 (2014): 8410-8415.

[2] J. Wertsch, V. Vygotsky and the social formation of mind. Harvard University Press, 1985.M.L.

[3] M.J Laakso, (2010). Promoting Programming Learning. Engagement, Automatic Assessment with Immediate Feedback in Visualizations. TUCS Dissertations no 131.

[4] Epstein, B.B. Epstein and G.M. Brosvic, Immediate feedback during academic testing. Psychological Reports 88, No. 3, pp. 889-894, 2001.

[5] T. Rajala, E. Kaila, M.J. Laakso, and T. Salakoski, Effects of Collaboration in Program Visualization. *Technology Enhanced Learning Conference 2009,* TELearn 2009, Academia Sinica, Taipei, Taiwan, 2009.

[6] Q. Wang, Design and evaluation of a collaborative learning environment. Computers & Education Vol. 53, No.4, pp. 1138-1146, 2009.

[7] J. Kay et al. "Problem-based learning for foundation computer science courses." *Computer Science Education* Vol. 10 No. 2, pp. 109-128, 2000.

[8] M.J. Laakso, E. Kaila, and T. Rajala, ViLLE - Designing and utilizing a collaborative learning environment. Submitted to Computers & Education, 2015

[9] P. Dale, and P. Haden. "Parson's programming puzzles: a fun and effective learning tool for first programming courses." *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. Australian Computer Society, Inc., 2006.

[10] E. Kaila et al, *ViLLE: Teacher’s handbook*, pp. 68-116, 2014 https://ville.cs.utu.fi/doc/TheBookOfViLLE.pdf, Last accessed 26 March 2015