Redesigning an Object-Oriented Programming Course

ERKKI KAILA, University of Turku
EINARI KURVINEN, University of Turku
ERNO LOKKILA, University of Turku
MIKKO-JUSSI LAAKSO, University of Turku

Educational technology offers several potential benefits for programming education. Still, to facilitate the technology properly, integration into a course must be carefully designed. In this article, we present a redesign of an object-oriented university level programming course. In the redesign, a collaborative education tool was utilized to enhance active learning, to facilitate communication between students and teachers, and to remodel the evaluation procedure by utilizing automatically assessed tasks. The redesign was based on the best practices found on the earlier research of ours and the research community, with focus on facilitating active learning methods and student collaboration. The redesign was evaluated by comparing two instances of the redesigned course against two instances using the old methodology. The drop-out rate decreased statistically significantly in the redesigned course instances. Moreover, there was a trend towards higher grade average in the redesigned instances. Based on the results, we can conclude that the utilization of educational technology has a highly positive effect to student performance. Still, making major changes to course methodology does not come without certain difficulties. Hence, we also present our experiences and suggestions for the course redesign to help other educators and researchers perform similar design changes.

Categories and Subject Descriptors: K.3.1 [Computers and Education]: Computer and Information Science Education

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Object-oriented programming, course redesign, programming education, course methodology

ACM Reference Format:

Erkki Kaila, Einari Kurvinen, Erno Lokkila and Mikko-Jussi Laakso. 2016. Redesigning an Object-Oriented Programming Course. *ACM Transactions on Computing Education.*

1. INTRODUCTION

During recent years, an imminent need for redesigning teaching methods in information technology education has become obvious. The students find the topics difficult and seem to have problems with the abstract concepts [Dunican 2002]. The problems are evident in programming courses, as typically the drop-out rates are high. Though most of the research done on learning programming is about introductory programming, the same difficulties are often present when advancing to topics such as object oriented programming. The underlying reason is often the nature of learning programming: students need to actively engage in programming to learn how to program.

Traditional programming courses are often taught via lectures and assignments. The assignments in a traditional setting are done in a computer lab or similar and assessed by the teaching staff. Utilizing educational technology makes it possible to increase the level of active learning: by facilitating features such as automatic

Authors' address: Department of Information Technology, University of Turku. 20014-Turun yliopisto, Finland.

assessment and immediate feedback, the number of active tasks in the course can be increased significantly. This enables an active approach to programming, where students learn by writing programs and completing other assignments instead of sitting passively in lectures.

In this paper, we describe a comprehensive redesign of an object oriented programming course. The approach chosen in the redesign was to facilitate active learning by changing half of the lectures into active learning sessions. We also decided to encourage student collaboration, as various earlier studies have proven that it has a highly positive effect on learning. A third aspect in the redesign was to enhance teacher-student communication by utilizing weekly surveys to collect students' perceptions on the lectures and the active learning sessions. Finally, we decided that the exam methodology needed to be adjusted as well: a traditional pen and paper approach was deemed unsatisfactory in a course with a lot of programming tasks. Hence, an automatically assessed electronic exam was utilized.

All elements in the redesign are based on the best practices of existing computing education research conducted by ourselves or by the research community. We start by evaluating these practices followed by a detailed description of the redesign. Then, student performance in the old instances of the course is compared with that in the redesigned instances. Next, the results are analyzed accompanied by our experiences on implementing the redesign. Finally, we present suggestions for other educators who are planning to adapt similar features in their courses.

2. RELATED WORK

Neither learning nor teaching programming is considered an easy task [Ben-Ari 2001], [Jenkis 2002], [Pattis 1993], [Caspersen & Bennedsen 2004]. [McCracken et. al. 2001] raise concerns whether university level introductory courses to programming achieve the expected results. Students may, for example, believe that after assigning the value from one variable to another, the first variable no longer holds a value, or that variables may hold more than one value [Ben-Ari 2001]. Moreover, [Gomes & Mendes 2014] state that students lack intrinsic motivation due to natural difficulties associated with programming. There obviously is room for improvement in the current landscape of computer science education.

The difficulty in programming lies partly in the fact that programming is not merely a single skill, but a composition of several processes. [Jenkins 2002] recognizes the required skills to not form a simple set, but rather a hierarchy from which several separate skills are utilized simultaneously. These skills have been classified in various ways, for example Bloom's taxonomy [Bloom 1954]. Hence, teaching programming as a single skill is a futile attempt. However, good results can be achieved with a combination of different teaching methods [McCracken 2001], [Carsten 2003], [Giraffa et al. 2014].

Lectures, reading and other passive forms of learning are not useful in conveying the skills or the thought processes required for programming [Jenkins 2002]. According to the constructivist theory of learning, the way teaching should be done is to let students build upon their old experiences and knowledge [Wertsch 1985]. Thus, active methods of learning are preferred. According to [Freeman 2014], active learning can be thought of as any activity wherein the student actively partakes in the process of forming a solution to a given problem. This sharply contrasts with traditional behavioristic lecturing, where lecturers recite facts and students are expected to learn these facts. A concrete example of utilizing active learning is the concept of flipped

classroom, where lectures are served as video clips, and the time spent traditionally on lectures is dedicated to active assignment sessions [Amresh et al. 2013], [Sarawagi 2013].

Instead of lectures, educators are encouraged to embrace new teaching methods [Grissom 2013]. New technology provides new affordances for teaching and one such affordance is the ability to give students feedback immediately after their answer. This ability to automatically assess student's answers is a key benefit gained from utilizing educational technology [Laakso 2010]. Immediate feedback has also been found to improve learning results in students [Epstein, Epstein & Brosvic 2001]. Immediate feedback can, in the best case, provide students a cognitive conflict between what they thought was correct and what actually is. Such a conflict forces the student to reassess their beliefs and possibly seek out new information and finally assimilate all newly gathered and re-evaluated information with their old knowledge to resolve the conflict [Wertch 1985].

Recently, there has been an increasing interest in collaborative learning. This is reflected partly in an increasing number of articles on collaborative learning in computer science (CS). Collaboration has been found to be highly beneficial in supporting the learning process of students [Rajala 2009], [Wang 2009], [Raitman et. al. 2005], [Hwang et al. 2012]. This is expected in light of the constructivist learning theories. When working together with other students, the participants will inevitably form new knowledge from their interaction based on their old knowledge. [Damon 1984] argues that different opinions and assumptions force students to argue and reassess their beliefs. He also describes four aspects in peer collaboration that promote learning. First, students are able to talk to each other on the same level and thus understand each other. Second, they can talk directly to one another without feeling threatened. Third, students are more likely to accept feedback from their peers which may cause them to reassess their beliefs. Finally, the communication between students is more equal than that between students and their instructor. As a result, the students are more willing to challenge ideas from their peers than their instructor [Damon 1984]. Collaboration does not only benefit student learning: even teachers have been found to benefit from teacher-to-teacher collaboration in their work [Johnson 2003].

[Beck & Chizhik 2014] utilized cooperative learning and instructional methods in a CS1 level programming course. They divided students into small groups giving each group member a specific task in the group. Students then worked in these groups on exercises designed to be solved cooperatively. After the group session, the lecturer held a debriefing session for the whole class. This debriefing was designed to work in tandem with the group processing to promote the learning and understanding of the material for the student. The cooperative learning students outperformed traditional lecture students, although the improvement was partly instructor dependent.

Pair programming has been found effective for student performance in programming courses (see for example [Salleh et al. 2011] or [McDowell et al. 2002]). Moreover, [Nagappan et al. 2003] found out, that pair programming can also improve student experience in the course, as pair programmers were more self-sufficient and more likely to complete the class than students working alone. In our research group's previous study, [Rajala et al. 2011] provided strong evidence that students benefit from working in collaboration. We analyzed the screen captures and conversations of 112 CS students' two hour lab work. The control group consisted of 50 students working alone and the treatment group of 62 students working in pairs. We found out that students working collaboratively spent more time on demanding tasks and were more engaged than their individually working peers.

Collaborative learning has also been studied in lower- and upper-division computer science. Several studies have been made on the effect of collaborative learning on student performance and feedback received from students. The results show a marked positive difference from the control group [Lee et al. 2013], [Simon et al., 2010]. Students, in general, seem to prefer collaborative learning methods over individual learning. [Renaud and Cutts 2013] were able to improve student decision making in security issues using Peer Instruction. In Peer Instruction students answer multiple choice questions first individually but are allowed to change their answers after a group discussion. The group discussion enables the students to reflect on what they answered and solve potential cognitive conflicts. [Hundhausen et al. 2013] were able to improve students' critical thinking and code-analysis skills using an active learning approach to code review, inspired by the same process used in the industry. In a pedagogical code review session, the students present code they have written to an experienced instructor as well as other students, who then offer suggestions on how to improve the code. The aim in a pedagogical code review session is to merge the views of the novice student and an expert programmer.

However, as [Grissom 2013] points out, it is not merely enough that educators know about alternatives to passive lecturing, they should also adopt and utilize these new methods in their teaching to realize any potential gains in learning. This is also emphasized in a case study made by [Goode and Margolis 2011], in which they study a school reform. However, they point out that instilling change in any educational system is challenging. Regardless, their reform succeeded in, for instance, increasing student perceptions of the usefulness of computer science and motivating students to stick with difficult problems instead of giving up.

[Vihavainen et al. 2014] present a systematic review of redesign approaches and on their quantitative effect on student performance in CS1 courses. They discuss several methods of intervention (including for example collaboration, content change and peer support), and discuss their effect on pass rates reported in different case studies by other authors. The authors conclude that teaching interventions can improve the pass rates by as much as one third, which can be seen as a remarkable result. Moreover, they state that while no statistical differences between intervention methods can be found, the courses with relatable content combined with cooperative elements were most effective.

3. VILLE – COLLABORATIVE EDUCATION TOOL

This paper describes the refactoring of an object oriented programming course. The refactoring was decided to be based on ViLLE, a collaborative education tool developed at the Department of Information Technology, University of Turku. ViLLE is a web-based collaborative education tool that supports a variety of different exercise types. Most of the exercises are automatically assessed and give immediate feedback when submitted. Additionally, to support active learning, ViLLE does not limit the number of submissions. ViLLE is currently used by more than 2,000 teachers and 25,000 students around the world. A comprehensive description of the tool can be found in [Laakso et al. 2016]. The ViLLE exercise types selected for refactoring were

Coding exercise: automatically assesses the student solution against the model solution written by the teacher. The exercise provides authentic compiler and run-time exceptions, enabling students to fix the potential problems before resubmitting.

Quiz exercise: provides multiple choice and open questions. Quizzes are particularly useful for lecture summaries and for testing code tracing and theoretical skills in the exams.

Visualization: a program visualization exercise where graphical tracing of code execution is accompanied with different types of questions.

Program simulation: an exercise type where the students need to simulate the program execution one code line at a time by creating and manipulating variables and methods.

Sorting exercise: ViLLE supports Parsons puzzles [Parsons & Haden 2006], where shuffled code lines need to be ordered according to given tasks. Additionally, other types of sorting tasks (such as connecting the variable definition and value) can be created.

Survey: ViLLE also supports surveys with optional embedding of pictures, audio and video.

ViLLE can also be used to record student attendances with RFID readers. This functionality can be extended to demonstrations, where students can use ViLLE to record the assignments they have completed. This allows the course staff and the students to see all obtained scores in real time.

4. REFACTORING THE COURSE

"The basic course of object-oriented programming" is a typical object-oriented programming course taught in the University of Turku. The course is mandatory for all computer science majors (and for some of the other students in the faculty, including mathematics, physics and chemistry majors) and the students typically take it in their first year. Before the course, all students attend the "Basic course for algorithms and programming", which is a typical CS1 course, containing the basics of programming in Java. In the second course, the fundamental concepts of object oriented programming, including (but not limited to) writing classes, inheritance and polymorphism, are taught. Java is still used as the programming language, but the focus of the course is more on the general concepts instead of features typical for any particular language.

In this section, we describe the changes made in the course between the instances of 2011-12 (the old course) and 2013-2014 (the new course). The topics covered by the course as well as the learning goals and the credits awarded remained the same between all instances. The goal of the refactoring was to lower the drop-out rates by facilitating active learning, student collaboration and communication between the students and the teachers. By refactoring, we hence try to find the answers to two research questions:

1) *Does the refactoring lead to lower drop-out rate in the course*, and
2) *Does the refactoring lead to higher course grades?*

The corresponding null hypotheses are that the refactoring has no effect on the drop-out rates, and if it does, the grade average will drop accordingly. Before the steps taken in the refactoring are described, the old instance is briefly summarized.

4.1 Old Version of the Course

The old version of the course (from now on C1) was taught until the spring of 2013. In this paper, we use the instances of 2011 and 2012 for comparison. The old version of the course consisted of lectures, demonstrations, a course project and an exam. The course lasted for eight weeks, but the final week was reserved for the final exam. The final project could be submitted after the final exam. Each week of the course consisted of four hours of lectures (2 x 2h). Additionally, there were four weekly demonstrations,

starting from the third week. In these 2-hour sessions, the students provided their answers to programming tasks given earlier. At least 50 percent of demonstrations needed to be completed to attend the exam.

The old course structure is displayed in Table 1.

Table I. A summary of the old instance of the course

Component	Amount	Description
Lectures	7 x 2 x 2h = 28h	Traditional lectures in a lecture hall
Demonstrations	4 x 2h = 8h	Programming assignment presentation in front of class; done in smaller groups (typically 20 to 30 students)
Final project	1	Programming project (typically a simple game or similar)
Final exam	1 with two possibilities to retake the exam	Two to three programming tasks or essays, completed using pen and paper

As seen in the table, the course follows a typical structure of most programming courses.

4.2 Step 1: Enhancing Active Learning

In refactoring, the first step was to facilitate active learning. As proven by various educational researchers, learning performance can be enhanced when students perform tasks actively instead of passively listening to lectures. The first step in the redesign was implemented by changing half of the lectures into tutorials. The tutorials, created in ViLLE, are a combination of study material (such as text, images, tables and videos) and ViLLE exercises. An example of a tutorial is displayed in Figure 1. The latter lecture of each week was replaced with a tutorial session, where the students brought their computers with them. The session was organized in a lecture hall and was supervised by course personnel aided by older students mentoring the participants when needed. The attendance to tutorial sessions was made mandatory (though one absence was allowed). The attendances were recorded by delivering an RFID tag to each of the students in the course, and by utilizing RFID readers which were directly connected to ViLLE server. This enabled an up-to-date view of the attendances to both, teachers and students.

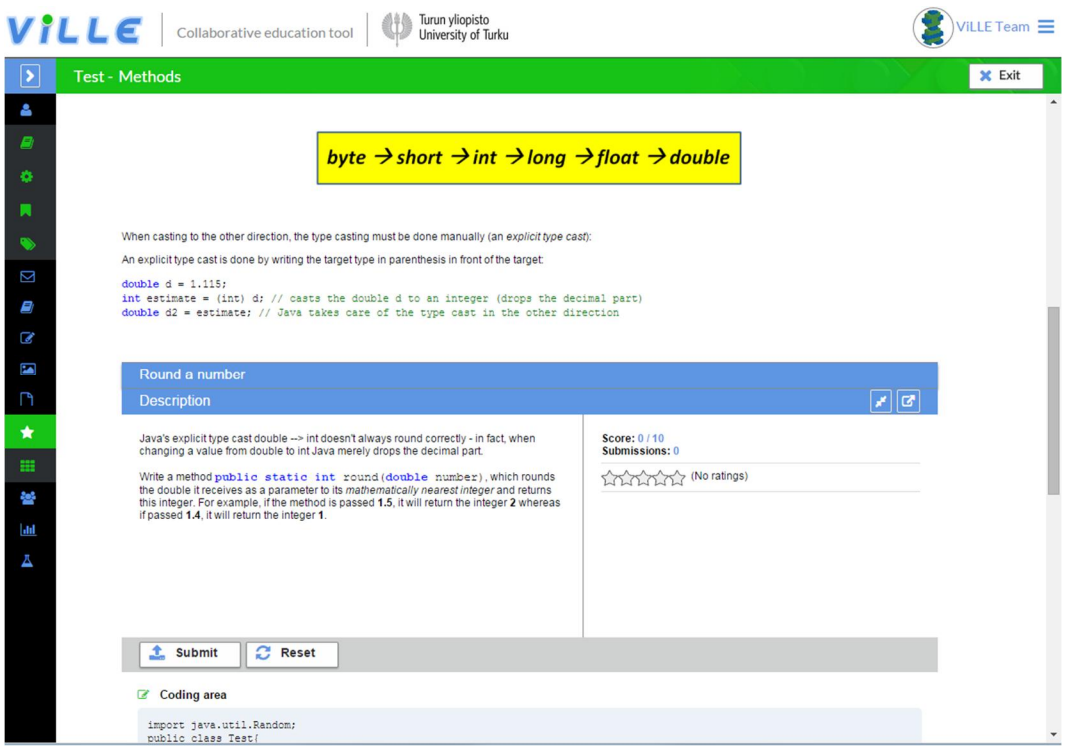

Fig. 1. Example of a tutorial in ViLLE

Each tutorial was designed to underline the topics discussed in the lecture that week. Hence, the lecture and the tutorial formed a holistic module each week, where the theory was first offered, and then the topic could be rehearsed with relevant exercises. For each tutorial, different kinds of exercises were prepared. While most of the exercises were about writing and executing program code, some quizzes and sorting exercises were also used to keep the task more varied and interesting. The tutorials were opened when the session started, and kept open until the next tutorial. This way the students were able to finish the tutorial after the session if necessary.

Moreover, a small set of additional ViLLE exercises were prepared for each week. These exercises opened during the lecture and were meant to be used as a reminder about the topics covered in the lecture. These additional exercises were designed to be easier than the exercises in the tutorials. ViLLE was also used to collect the lecture and tutorial attendances. Lecture attendances were not made mandatory, but a small bonus (less than 0.17 % of the grade) was offered to students who attended all lectures.

4.3 Step 2: Encouraging Collaboration

The second step was to facilitate student collaboration. As seen before in (Rajala et al. 2009, Rajala et al. 2011), learning performance can be significantly improved if the students do the exercises in collaboration with other students. With this in mind, the tutorials were built to support collaboration. In the tutorial sessions, the students paired up with another student to work together using one computer. Discussion during the sessions was encouraged and the controller (i.e. the student who used the mouse and keyboard) was switched every fifteen minutes. The points collected from the tutorial exercises were awarded to both students.

The demonstration sessions were kept in the redesigned model as well. The reason for this is the possibility to offer more complex programming tasks as well as problems that do not have straightforward solutions to them. Still, the demonstration sessions were modified to facilitate collaboration and discussion. In the old course, the session started with students writing down the assignments they completed, and the demonstrator then picked the students to present their solutions. In the new model, the students still started the session by registering the completed assignments, though this time using ViLLE. The demonstrator then used ViLLE to divide the students randomly into groups (an algorithm was used to ensure that each group contained the answers to each assignment). Next, the students had approximately 20 minutes to discuss and compare their solutions before the presenters were chosen. The presenters then displayed and discussed their solutions to the programming tasks, with additional discussion encouraged. Moreover, an extra assignment was provided after the previously given ones were presented. The remaining time in the session (usually around 15 to 30 minutes) was spent in groups solving this additional task

4.4 Step 3: Facilitating Student-Teacher Communication

The third step was to facilitate communication between the course staff and the students. This was thought to be especially important since several new features were presented in the course. Hence, two surveys were conducted each week. The first one was used to collect opinions about the lecture, and the second one from the tutorials. The lecture survey contained the same three questions each week:

- What did you learn in this week's lecture?
- Which things remain unclear after this week's lecture?
- How would you improve the lecture?

Similarly, after each tutorial the students answered the same three questions:

- What did you learn in this tutorial?
- Which things remain unclear after the tutorial?
- How would you improve the tutorial?

The feedback was analyzed after each week and the topics that seemed to remain unclear were re-addressed in the next lecture. Also, the tutorials were modified between instances and the final tutorial (containing a summary of all topics in the course) was built based on the issues the students reported on the surveys.

Also, after each of the demonstration sessions a short survey was conducted. In this survey, the students could simply give feedback on the demonstrations. Again, the answers were analyzed and the consecutive demonstrations were modified based on the feedback. In practice, this meant for example giving more detailed instructions on assignments if some of the previous ones were seen as too vague. Finally, after the course exam a survey about it was conducted. This was deemed important since this was the first time an electronic exam was used.

For additional assistance between lectures and tutorials, a group of more experienced (typically second or third year) students were nominated as mentors. A special mentoring session was arranged once a week. In this session the students participating in the course could come by to ask assistance for tutorials, demonstrations or the weekly assignments. At least two mentors were present to assist the students in these sessions. These mentors were also present at the tutorial sessions to assist the students with problems.

4.5 Remodeling Evaluation Methodology

Since the students spent the entire course writing a great number of programs either with an IDE or especially in ViLLE, a normal paper exam was deemed to be unsatisfactory. Instead, an electronic exam was implemented with ViLLE. This way the students could actually write, compile and test their code and see the results as well as any compiler error messages on screen. The main purpose for this was to create an environment where writing code was as close to an authentic situation as possible in an exam. Besides programming assignments, some other tasks – such as quizzes and sorting exercises – were also included.

The typical programming task in the exam contained a task description and (in some cases) a pre-defined set of code. The students were then asked to write the required code using the code editor in ViLLE. The code could be submitted (i.e. compiled and executed) as many times as wanted. The exam score was based on the final submission. An example of the exam task (translated into English) is displayed in Figure 2.

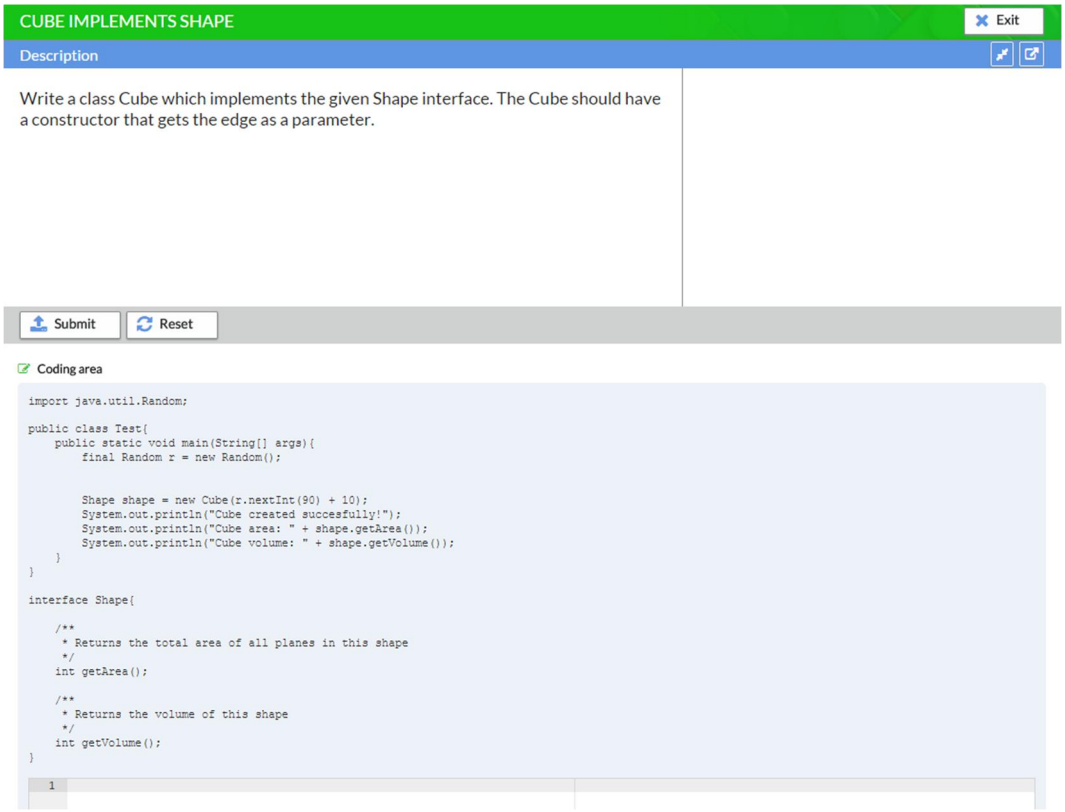

Fig. 2. Example of a task from the final exam, translated into English

The electronic exam was fully automatically assessed. Unlike in the tutorial exercises, the students couldn't see the feedback (except for the program output and possible compiler and run-time errors) or the score when submitting their answers. Besides providing the students the possibility for resubmitting their answers as many times as wanted within the time limit, automatic assessment meant that the exam results could be published immediately after the exam.

To confirm the comparability to earlier instances of the course, the exam was designed to be more challenging. The exams of C1 typically consisted of two to three questions, where one or two were programming tasks. In the redesigned C2, the exam consisted of seven to eight programming tasks with one or two additional questions. To ensure that the exam in C2 was at least as difficult as in the previous instance, the exam was audited by three non-affiliated university level teachers and researchers, who all agreed respectively, that the new exam is at least as challenging as the old one.

4.6 Conclusion: Redesigned Course

The list of topics taught remained the same after the redesign. The method was however changed significantly. The course structure is displayed in the Table II.

Table II. A summary of the redesigned course

Component	Amount	Description
Lectures	7 x 2 = 14 h	Traditional lectures in a lecture hall
Tutorials	7 x 2 = 14 h	Tutorial sessions done in collaboration in lecture hall
Demonstrations	4 x 2 = 8 h	Programming assignment presentation in front of class; done in smaller groups (typically 20 to 30 students). Enhanced with collaborative work at the beginning and the end.
Additional ViLLE exercises	7	Simple tasks opened during the lecture to underline lecture topics
Final project	1	Programming project (typically a simple game or similar)
Final exam	1 (but with 3 options to take it)	7 to 8 programming tasks with one or two additional tasks, done electronically in ViLLE.

The comparison of old (C1) and new (C2) course methodologies is displayed in the Table III.

Table III. Comparison of old (C1) and new (C2) course methodologies

Component	C1	C2
Lectures	4h each week	2h each week
Tutorials	-	2h each week
Demonstrations	4 x 2h	4 x 2h, collaboration
Additional exercises	None	Approx. 3 each week
Final Project	1	1
Final exam	Pen and paper	Electronic
Feedback collected	None	2-3 short surveys a week + one after the exam
Mentoring	None	2h each week

As seen in the table, active learning, collaboration and communication are enhanced heavily in the new version of the course. In the next section, we present the earlier studies which the redesign was based on, followed by the results on student performance in old and new instances. In the section after that, the effects and our experiences are discussed.

4.7 Earlier Studies on Methodology

The redesign was based on the best practices of the e-learning research community as well as on our own previous research. We have previously shown in 2-hour controlled tests [Rajala et al. 2008], [Kaila et al. 2009] that educational technology can be highly

beneficial for learning, but only if used in the higher levels of engagement. This seems to be in line with the engagement taxonomy represented by [Naps et al. 2001]. The positive effects of engagement and immediate feedback were concluded in [Kaila et al. 2009]. We have also shown in [Rajala et al. 2009] and [Rajala et al. 2011] that collaboration can have a significant effect on learning, and that the students doing exercises in collaboration seem to be highly engaged in the task at hand. On [Laakso et al. 2008] we found out that the cognitive load of learning to use a new tool has a significant negative effect on learning. With this in mind, the redesign was based on a single comprehensive learning environment. Some of our first studies on technology-enhanced programming and computer science courses were published in [Kaila et al. 2014] and [Kaila et al. 2015].

5. RESULTS

To compare the learning performance between C1 and C2, the grades and pass rates of all four instances were acquired from the university offices. The instances are displayed in Table IV.

Table IV. Course instances

Instance	Methodology	N
2011	C1	186
2012	C1	201
2013	C2	191
2014	C2	158

The total number of students in the instances of the old methodology (C1) was 387, and in the instances of redesigned methodology (C2) 349. Most of the students were computer science majors, but some mathematics and physics majors also took the course as part of their minor studies. The number of students attending the course varies from year to year due to changes in the number of accepted students in the IT and other departments. Also, the smaller amount of students taking the course in 2014 is likely due to higher pass rate in 2013 (and hence smaller number of students re-taking the course the following year). The course is typically taken as part of the first year studies and is the second programming course in the curriculum.

The course was graded in scale of 1 to 5, with 5 being the best grade, and 1 the lowest passing grade. Some bonus points were awarded for completing the majority of tutorials, attendances and ViLLE exercises, but the bonus points were only awarded if the student passed the exam by collecting at least 50 % of the maximum points. Similar method was used in the old instances of the course with the demonstrations.

Moreover, it was possible to pass the course without taking the exam by collecting at least 90 % of all possible points in the course. However, almost all of the students who collected this many points also took the exam. The grades from all course instances are displayed in Table V.

Table V. Grades from all course instances, with 5 being the best grade

	Fail	1	2	3	4	5
2011 (C1)	58.06 %	8.06 %	4.84 %	8.06 %	9.68 %	11.29 %
2012 (C1)	49.25 %	7.46 %	9.45 %	7.46 %	9.45 %	16.92 %
2013 (C2)	23.04 %	14.66 %	8.90 %	7.33 %	11.52 %	34.55 %
2014 (C2)	25.95 %	5.70 %	8.23 %	5.70 %	10.76 %	43.67 %

As seen in the table, the number of failed students is significantly higher in the older instances. The pass rates for all instances are visualized in Figure 3.

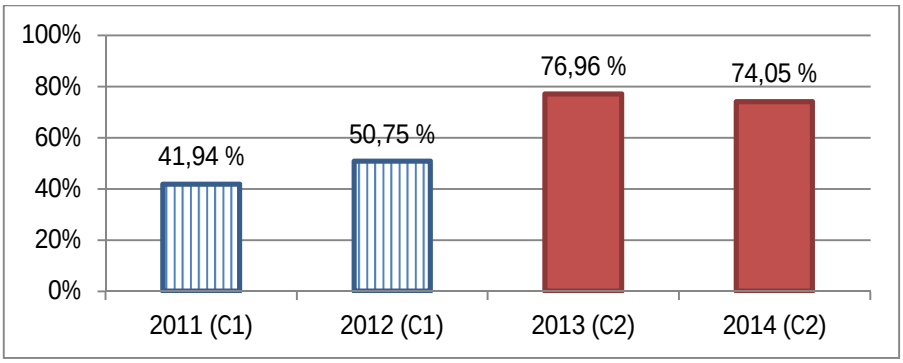

Fig. 3. Pass rates of all instances visualized

As seen in the figure, the pass rate was significantly higher in both instances of the course using the new methodology. This also applies to total average of C2 (75,64 %) compared with average of C1 (46.51 %).

For the statistical analysis of pass rates between the treatment and control groups we used the Test of Equal or Given Proportions, implemented in R. For this, we merged the two instances for both C1 and C2, respectively. This gave us 78 + 102 = 180 passes in C1, versus 147 + 117 = 264 passes in the C2, with 186 + 201 = 387 and 191+158 = 349 total participants, respectively. The test showed that the pass rates have a clear, statistically significant difference between each other (p-value < 0.001).

The grade averages of all instances of the course are visualized in Figure 4. Only students who passed the course are included in the average.

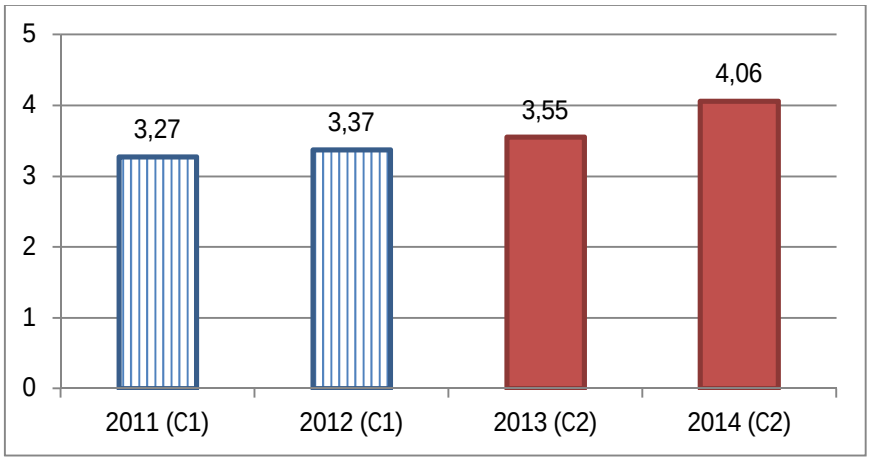

Fig. 4. Grade averages of all instances visualized

As seen in the figure, there seems to be a trend towards higher grade average in individual instances of C2. The statistical differences between course instance grades were calculated with the Chi-square test. The results are displayed in Table VI.

Table VI. The statistical differences of grade distributions between course instances

	2012 (C1)	2013 (C2)	2014 (C2)
2011 (C1)	0.049	2.8084×10^{-27}	4.611×10^{-33}
2012 (C1)		4.109×10^{-18}	2.767×10^{-22}
2013 (C2)			0.004

Notably, all differences are statistically significant ($p < 0.05$), though the difference between instances of C1 is only barely significant. Moreover, the difference between instances of C2 is a lot smaller than difference between instances of different methodologies.

6. DISCUSSION

It seems that redesigning the course methodology had a highly positive effect on pass rates and grade averages. Hence, we can reject the null hypotheses and answer the research questions positively. Facilitating active learning methods seems to be highly beneficial over passive lectures. Notably, there was also a significant difference between the grade averages of course instances facilitating the new methodology. It is possible, that this is due to some changes made to the course materials (such as increasing or decreasing the number of tasks in the tutorials or modifying the lecture slides) based on student feedback and our own experiences. However, there was also a slight drop in pass rates between instances of C2 though the difference was not statistically significant.

6.1 Enhancing Active Learning with Tutorials

The tutorials were carefully designed to supplement and train the topics taught in the lectures. Still, it was somewhat difficult to come up with an appropriate number of exercises and to determine a proper difficulty level for them. The goal was to make the tutorials challenging enough for even the more experienced students, but still easy

enough so that the less experienced students can complete them within the given one week time frame. As the aforementioned equation is close to impossible to solve, a more practical goal was set to the time spent by students on tutorials: if the best students took at least half the session (approximately 50 minutes) to complete the tutorial, and the worst ones could still complete the tutorial within a week, the difficulty could be seen as appropriate. In fact, most of the tutorials fell into this category. The tutorials were slightly modified between the 2013 and 2014 instances if they seemed too difficult or too easy.

The student feedback on the tutorials was also analyzed after each tutorial session. This was then used to fine-tune the difficulty level of consecutive tutorials. Typically, the feedback was quite heterogeneous about the difficulty: some students reported the tutorial as too difficult and some too easy. Still, most of the feedback on the tutorials was still positive. The feedback from tutorials could be roughly divided into seven categories. Some students think that tutorial exercises are too hard or there are too many exercises: *"[I would like to have] easier exercises, as always. Also, bring back 'puzzle exercises'; they are a nice change to writing."* On the other hand, some students think that the exercises are too easy or that there should be more of them: *"Tutorial was short, it took only an hour to finish the exercises. Exercises were pretty straightforward."*

Most students have a very positive attitude towards learning via tutorials: *"A lot of work (there is still a lot to do – which is not usual for me) but I'm sure I'll learn a lot and it's worth the effort...."* And then there are students who are not seeing the benefits of having to work hard*: "There are too many tutorial exercises. We made it barely half way through with my partner."* Still, nobody seemed to argue against tutorials as a method in the feedback. Students think that tutorials fit well together with lectures: *"Tutorials are an amazing invention; you couldn't learn these things otherwise. Lectures are sometimes really boring and you don't really learn from them, that's why it's really great that we have an opportunity to work together and learn."* T

From a technical perspective, the tutorials worked quite well. The sessions were organized in a 250-seat lecture hall where the major concern was the network connectivity. The wireless router in the hall wasn't powerful enough to handle enough simultaneous connections. Luckily, the lecture hall was equipped with enough LAN ports to provide a connection for all the pairs and the only thing needed was to supply the LAN cables. This also meant that the network traffic from the hall was routed through a single switch and firewall, so blocking all sites besides ViLLE and the Java API in exam situation was effortless. Some of the students' laptops did not contain LAN connectors, but since there were only a few cases like this, they were allowed to use the wireless network.

The student collaboration in tutorials seemed to work really well. The discussion in the lecture hall was continuous, and for the most part the students seemed to discuss the topic at hand. We have previously analyzed discussion in student collaboration [Rajala et al. 2011] during controlled tests. In the analysis it seemed that almost all of the discussion was about the topic. The students also seemed to ask for assistance quite actively when needed. While approximately 150 students attended the tutorial, we quickly found out that four mentors (some from course staff and some older students) were enough to provide assistance. A minor issue we realized after the first session was that the mentors could not reach those sitting in the middle of the rows. To solve the issue, the students were seated on every other row in consequent tutorials.

6.2 Demonstrations and Lecture Attendance

In the demonstration sessions, the collaboration seemed to work equally well. Again, there was a lot of discussion among the groups before the students presented their solutions. In the feedback, the students wished that the groups would remain the same when completing the additional assignment after the presentations, as "forming new groups is too much of a hassle". One issue the students pointed out in the first two sessions was that "the tasks are sometimes too vague". This was addressed in two latter sessions by trying to write the assignments as clearly as possible. Another issue was on demonstration number three, where one of the tasks was to write a comparison method for a class modeling a player's hand in a game of poker. Though completing this task awarded the students half of the points available from that session, many students reported the task as too laborious. Still, most of the feedback from the demonstrations was highly positive.

The students had a chance to attend the mentoring sessions, if the tutorial or the demonstration tasks seemed too difficult. Approximately 30 to 40 students were present in each of the sessions. The mentors reported that a significant part of students attending the sessions actually had no specific questions about the tasks, but instead utilized the session to work collaboratively on the tasks – a practice that was highly encouraged by the course personnel. The mentoring offered during the tutorial sessions probably decreased the number of students needing additional tutoring as well.

The feedback collected from the lectures proved to be extremely useful. After each lecture the feedback was briefly observed, and the issues that were raised addressed at the beginning of the next lecture. Typical issues collected from the feedback were either general, like "more examples should be shown", or highly specific, such as "the difference between interfaces and abstract classes should be underlined". Again, most of the feedback was positive and the students seemed to be especially happy about the fact that the problems reported were actually addressed in the next lecture. The students participated in the lectures quite actively, as seen on Figure 5.

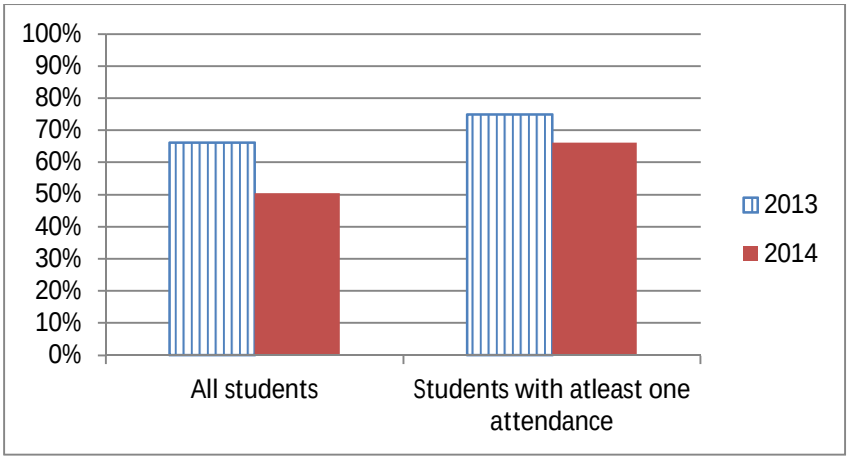

Fig. 5. The average number of lectures the students attended in instances of C2

In the figure, the average of all students as well as the average of students who had at least one attendance is calculated. The latter likely depicts the average of students who passed the course more accurately. Notably, the lecture average was a little lower in the 2014 instance, as was the pass rate. Still, finding actual correlation between two variables would require analyzing more instances. The overall high level of participations in both instances is probably partially due to bonus points that could be

collected on lecture attendances. Still, the bonus at maximum was less than 0.17 % percent of the course final grade, so a procedure like this can definitely be recommended.

6.3 Electronic Exam

The exam was also well perceived. There were practically no technical issues, but then again, the students had used ViLLE extensively in the course before the exam, so they were familiar with the features by the exam time. Since all students did not own a laptop (and the department couldn't provide enough spare ones), two computer labs were also reserved for taking the exam. The most beneficial feature in an electronic exam compared with a traditional pen and paper approach is the possibility to test and refine the program code as many times as needed. The submission numbers for all exam exercises are displayed in Figure 6.

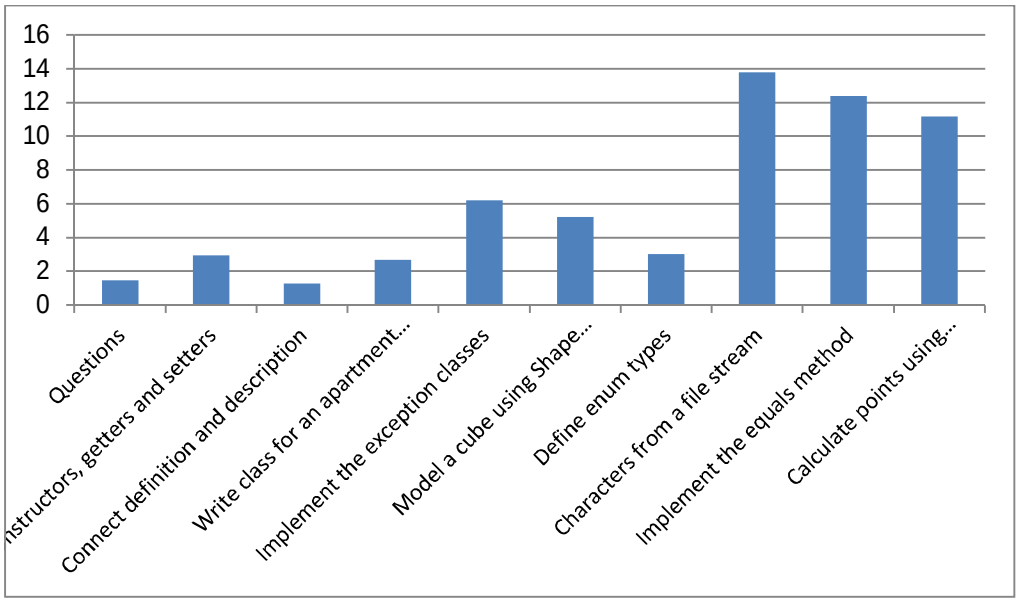

Fig. 6. Average number of submission made to exam questions in instances of C2 (combined)

All exercises except for 1 and 3 were coding exercises. As seen in the figure, there is a lot of variation in the submission numbers. For non-coding exercise types, the average was close to one, but for all coding exercises the students submitted (i.e. translated and executed) their code at least three times on average. For the most difficult questions, the average number was more than ten. The average exam scores for individual tasks are displayed in Figure 7.

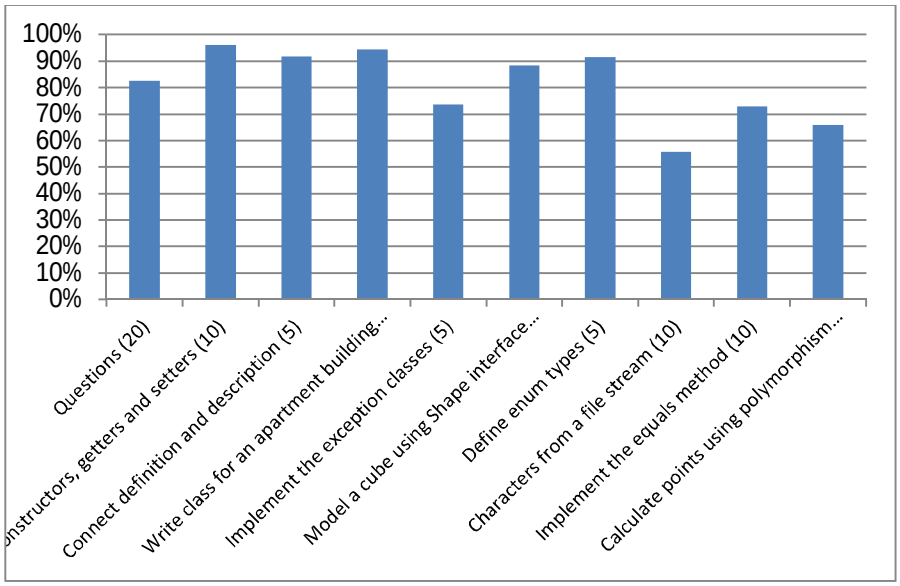

Fig. 7. Average scores for each question in exams of C2. Maximum points are displayed in parentheses.

The students also answered a survey after the exam. There were eleven questions answered in the Likert scale of 1 to 5 (1 = completely disagree, 5 = completely agree). There were three exam instances each year. The results obtained after the first exam instance are displayed in Table VII. Averages and standard deviations (in parentheses) are displayed.

Table VII. Students' perceptions after the first exam on both instances of C2.

	2013 (N=104)	2014 (N=83)
There was enough time to finish the exam	4.52 (0.75)	4.58 (0.70)
It was easy to do the exam	4.00 (1.01)	3.92 (0.95)
ViLLE was easy to use	4.40 (0.65)	4.29 (0.85)
I would rather do the exam on paper than in ViLLE	1.37 (0.89)	1.23 (0.65)
ViLLE suits well for this courses exam	4.80 (0.45)	4.66 (0.65)
I would do this test as an online-exam at home, if it was possible	3.74 (1.12)	3.65 (1.10)
I would recommend ViLLE to other students	4.52 (0.61)	4.35 (0.82)
From a technical point of view, ViLLE is an excellent solution	4.18 (0.73)	3.95 (0.90)
Which grade would you give to ViLLE as an exam system (1-5)	4.27 (0.59)	4.11 (0.75)
I got enough training to ViLLE before the exam	4.78 (0.50)	4.51 (0.83)
Evaluate the difficulty level of the exam (1=easy, 3=suitable, 5=hard)	2.97 (0.84)	2.88 (0.85)

As seen in the table, the students had very little technical issues, thought that ViLLE was easy to use, and had sufficient training to use the system before the exam. Most importantly, the students seemed to think that the exam on a programming course should be taken in an electronic form instead of pen and paper.

6.4 Results in Relation to Related Work

How do the results relate to previous studies in the field of computer science education research? In [Vihavainen et al. 2014] the authors quantitatively analyze several approaches for teaching introductory programming courses. Of analyzed methods, at

least *collaboration*, *content change*, *group work* and (peer) *support* were also applied in our redesign. Moreover, the authors conclude, that on average the redesign can improve the pass rate by one third, which seems to be in line with our results (the combined pass rate of C2 courses was 75,64 % in comparison to 46,51 % in C1). The authors also found out, that pair-programming as an only intervention method seems to have a worse effect, which underlines the importance of holistic redesign.

Collaborative learning has been found useful in various studies beforehand. The positive effect of adapting collaboration into tutorials and demonstrations in our redesign seems to confirm the findings of [Lee et al. 2013], [Simon et al . 2010] and [Hundhausen et al. 2013], to name a few. Though the redesign described in this article cannot be categorized as flipped learning (see for example [Sarawagi 2013], some similarities (such as emphasizing active learning) can be found. Though electronic exams in programming courses have not been studied widely, [Barros et al. 2003] and [Navrat & Tvarozek 2013] have had similarly encouraging experiences when they have replaced traditional pen-and-paper approach with more authentic approach. Finally, the results obtained seem to confirm the results of our research group from introductory programming courses (see for example [Rajala et al. 2009] and [Rajala et al. 2011]).

The experiences of [Haatainen et al . 2013] for providing additional support to students in the CS1 course seem to support the inclusion of mentoring in our course redesign. They found out, that the additional support received positive feedback from the students and the student mentors, but found no significant difference in learning results. This seems to indicate that mentoring is an important addition in redesign, but might not have an effect on the student performance if not accompanied with other methods. However, the positive effect might lead to better motivation, which [Nikula et al. 2011] found to be crucial in the programming courses: they state, that the lack of motivation leads to high drop-out rates. The decrease in drop-out rates was the single most important outcome of our redesign. [Keijonen et al. 2013] present an approach called *Extreme Apprenticeship*, which emphasizes active learning combined with personal advising. The authors found out that (similarly to our experiences) applying the approach lead to higher student performance in the introductory programming course. Moreover, the authors state that the method leads to a carry-on effect with more credits gained during the 13 month period after the course. In the future, we are curious to find out whether the positive effect reported in this research will carry on similarly.

7. SUGGESTIONS ON COURSE REDESIGN

Finally, we would like to readdress the factors considered in the redesign process based on the results and experiences presented in the previous sections. Though the changes made in the course method seemed to be highly effective in increasing student performance (and were generally well perceived by the students) we found some issues that need to be addressed when making changes in the methodology.

First, enhancing active learning with tutorials and other ViLLE exercises seemed to have a positive effect on the outcome of the course. This was somewhat expected, as various other studies have already proven that active learning methods are more effective than passive listening in the lectures. For example, the whole concept of flipped classrooms (though not utilized in our setup) relies on this fact. All in all, we faced only a few issues when implementing the tutorial sessions, partly because we were quite well prepared. Still, the factors that should be considered by other educators

are the technical implementation (electricity, LAN or WLAN connectivity, access to participants for guidance) and the proper difficulty of the tutorials. Unfortunately, there is no strict advice on the difficulty level, as the only way to properly evaluate the difficulty is to test the tutorials with a large enough, heterogeneous group of students.

Second, enhancing collaboration seemed to work as we intended. As we found out in [Rajala et al. 2011], when students work collaboratively the discussion is almost completely about the topic in hand. Naturally, some students were not happy about the idea of needing to communicate with anyone, but mostly the collaboration was quite well received. Still, a suggestion we need to make to fellow educators is to actively monitor the switching of the controller role once every fifteen minutes in the sessions. Quite a few students were very keen to be the ones using the mouse and the keyboard, while some seemed to be quite happy in the passive role. The collaboration was also utilized in our demonstration sessions. We found out that the tasks prepared for this need to be very carefully planned beforehand, as too difficult or too easy tasks can frustrate students and will bring no additional value.

Facilitating communication was very well perceived by the students. Our suggestion is to keep the weekly surveys brief (we came up with three simple questions per survey), and offer a small reward on filling them (in our case a couple of ViLLE points). Still, the most important suggestion we can make is that the results collected via surveys need to be analyzed and utilized during the course. The students are likely to be more motivated to report proper issues with the surveys if they think that this has an actual effect. Even more importantly, the surveys are extremely useful for making small adjustments in the course method and materials during the course. Hence, our suggestion is to reserve some man-hours for such adjustments (though we do realize that most lecturers have their hands full with teaching and course administration during the course).

The remodeling of the evaluation was one of the most important factors considered by us. In our opinion, answering exam questions using pen and paper is not a proper way to measure programming skill. When refactoring an exam, there are three suggestions we feel we should make. First, use a proper tool: the students should be able to compile and execute their programs with proper feedback and error messages, and the possibility of technical errors needs to be minimized. Also, if possible, a tool that supports automatic assessment makes life a lot easier for teachers. Second, the difficulty level of the exam needs to be properly evaluated by non-affiliated teachers and/or researchers. When the exam methodology is changed, it is possible (or even likely), that the difficulty level may not remain the same. Fortunately, there is a lot of good quality research done on how to design exams for programming courses, for example [Sheard et al. 2011], [Simon et al. 2015]. Finally, it is a good idea to prepare for cheating: if the exam is taken with a computer, it is important to block unwanted sites and communication as well as possible, preferably by whitelisting only the sites necessary for the exam. Also, supervision in the classroom where the exam is organized should be provided, just like in any exam.

The refactoring can of course be a huge mountain to climb for educators who traditionally have their hands full already. We do not necessarily suggest making the complete refactoring at once, as the individual steps can be utilized separately as well. In our experience, the redesign was worth the effort: the course total pass rate increased significantly, and the number of drop outs during the course decreased likewise. The student perception was also enthusiastic, as the students thought that the methods of active learning were useful, and the feedback they gave was properly addressed. Also, most of the work needed for refactoring the course is done before the

first instance. Forthcoming instances are a lot easier to organize. Still, it is wise to prepare for adjustments after the first instance, as it is likely that there will be a need for some.

8. STUDY'S LIMITATIONS

There are naturally some limitations in the study, mainly due to nature of the holistic redesign. First, there are the external concerns: the data from the earlier instances of C1 is not conclusive: for example, the lecture attendances were not recorded, and the detailed statistics or feedback about the exam is not available. Hence, the corresponding statistics from C2 in this article cannot be compared to earlier instances, but are merely provided to illustrate students' active participation and general contentment in the redesigned course. Similarly, the effect of the previous course in the curriculum cannot be measured validly. The CS1 course which most students take before this course was also redesigned during 2013 and 2014 (see [Kaila et al. 2015]). However, the contents of the courses are very different, and especially in the 2013 instance of the redesigned course there were numerous (48 to be precise) students who had taken the earlier instances of CS1. Still, it is possible that the redesign of the previous course has an effect on the results, although it is difficult to measure. Finally, the statistics from the consecutive programming courses in the curriculum are not detailed enough to observe the effect of redesign on them.

There are also internal concerns. First limitation is the number of factors in the redesign: since the changes were made into various aspects of the course, the effect of individual changes cannot be isolated. Hence, it is difficult to say whether some modifications are more useful than others or whether some of them had no effect at all. Also, the evaluation method was different. Although external experts evaluated the exam at least as difficult as the exam in the old course, there are still differences between an electronic exam and a pen and paper one. Still, it is really unlikely that passing the electronic version would be easier, as the evaluation was stricter (for example no points were awarded if the code did not compile) and there were a lot more exercises to complete.

Finally, one could question the novelty value of the methodology used in the redesign. As seen in Sections 2 and 6.4 for example, most of the interventions used for redesign have been tried out and studied before. However, the same argument could be applied to a lot of research in the field of computer science education. The novelty of this study comes from the implementation of the tool used as well as the complete design, application and (most importantly) validation of the methodology in an object oriented programming course. After all, teaching interventions that are proven to be effective are what most educators should be looking for – hence validating research should never be underrated.

9. CONCLUSIONS AND FUTURE WORK

We redesigned a programming course based on the best practices obtained from the research of ours and the research community. The redesign was done in four areas: by enhancing active learning, collaboration, student communication and by refactoring the evaluation procedure. All in all, the course redesign proved to be successful. The pass rate increased by more than 20 percent units in both instances of the redesigned course. While the increase in grade average was smaller, the overall trend was still positive, especially since the exam was likely more challenging. The feedback collected from the students was also mainly positive.

From a researcher's point of view, some critique on the setup should be given. The changes made to the course were holistic, since almost all elements in the teaching method were somehow addressed. While the change in whole appears to be really effective, it is impossible to isolate the effects of individual factors. After all, when course long performances are measured, there are of course various variables affecting them. Still, as the content of the course remained the same, and the number of participants was high enough, we feel confident about the significance of the results.

In future, we are planning to investigate the possibilities of tutorial-based learning in other types of courses, starting with an introductory database course and a course for algorithms and data structures. We are also planning to keep on fine tuning the methodology and the materials used in this course. Some comprehensive surveys will be conducted over the course instances (and over different courses) to collect holistic data on student perceptions. When this is joined with performance data (as well as data collected automatically by ViLLE) it will be possible to come up with more general suggestions on redesigning course methodology.

REFERENCES

Ashish Amresh, Adam R. Carberry, and John Femiani. 2013. Evaluating the effectiveness of flipped classrooms for teaching CS1. In Frontiers in Education Conference, 2013 IEEE, pp. 733-735. IEEE, 2013.

João Paulo Barros, Luís Estevens, Rui Dias, Rui Pais, and Elisabete Soeiro. 2003. Using lab exams to ensure programming practice in an introductory programming course. In Proceedings of the 8th annual conference on Innovation and technology in computer science education (ITiCSE '03), David Finkel (Ed.). ACM, New York, NY, USA, 16-20. DOI=http://dx.doi.org/10.1145/961511.961519

Leland Beck and Alexander Chizhik. 2013. Cooperative learning instructional methods for CS1: Design, implementation, and evaluation. ACM Transactions on Computing Education (TOCE), 13(3), 10.

Mordechai Ben-Ari. 2001. "Constructivism in computer science education." Journal of Computers in Mathematics and Science Teaching 20.1, 45-73.

Jens Bennedsen and Michael E. Caspersen. 2004. Teaching object-oriented programming-Towards teaching a systematic programming process. Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts. Affiliated with 18th European Conference on Object-Oriented Programming (ECOOP 2004).

Benjamin S. Bloom. 1956. Taxonomy of educational objectives. Vol. 1: Cognitive domain. New York: McKay.

Enda Dunican, 2002. Making The Analogy: Alternative Delivery Techniques for First Year Programming Courses. In J. Kuljis, L. Baldwin, & R. Scoble (Ed.), PPIG 2002: Proceedings of the 14th Annual Workshop of the Psychology of Programming Interest Group (Brunel University, London, UK, June 18--21, 2002). PPIG'02, 89--99.

Michael L. Epstein, Beth B. Epstein, and Gary M. Brosvic. 2001. Immediate feedback during academic testing. Psychological Reports 88(3), 889-894.

Scott Freeman, Sarah L. Eddy, Miles McDonough, Michelle K. Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. 2014. Active learning increases student performance in science, engineering, and mathematics. Proceedings of the National Academy of Sciences, 201319030.

Lucia MM Giraffa, Marcia Cristina Moraes, and Lorna Uden. 2014. Teaching Object-Oriented Programming in First-Year Undergraduate Courses Supported By Virtual Classrooms. The 2nd International Workshop on Learning Technology for Education in Cloud. Springer Netherlands.

Anabela Gomes and Antonio Mendes. 2014. A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations. Frontiers in Education Conference (FIE), 2014 IEEE. IEEE, 2014.

Scott Grissom. 2013. Introduction to special issue on alternatives to lecture in the computer science classroom. ACM Transactions on Computing Education (TOCE) 13.3, 9.

Jan Herrington and Peter Standen. 1999. Moving from an instructivist to a constructivist multimedia learning environment. In: World Conference on Educational Multimedia, Hypermedia and Telecommunications (EDMEDIA) 1999, 19 - 24 June 1999, Seattle, U.S.A.

Simo Haatainen, Antti-Jussi Lakanen, Ville Isomottonen, and Vesa Lappalainen. 2013. A practice for providing additional support in CS1. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2013*, pp. 178-183. IEEE, 2013.

Christopher D. Hundhausen, Anukrati Agrawal and Pawan Agarwal. 2013. Talking about code: Integrating pedagogical code reviews into early computing courses. ACM Transactions on Computing Education (TOCE) 13.3, 14.

Wu-Yuin Hwang, Rustam Shadiev, Chin-Yu Wang, and Zhi-Hua Huang. 2012. A pilot study of cooperative

programming learning behavior and its relationship with students' learning performance. Computers & education 58, no. 4 (2012): 1267-1281.

Tony Jenkins. 2002. On the difficulty of learning to program. Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences. Vol. 4.

Bruce Johnson. 2003. Teacher collaboration: good for some, not so good for others. Educational Studies 29(4), 337-350.

Erkki Kaila, Mikko-Jussi Laakso, Teemu Rajala and Tapio Salakoski. 2009. Evaluation of Learner Engagement in Program Visualization. 12th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2009), November 22 - 24, 2009, St. Thomas, US Virgin Islands

Erkki Kaila, Teemu Rajala, Mikko-Jussi Laakso, Rolf Lindén, Einari Kurvinen and Tapio Salakoski. 2014. Utilizing an Exercise-based Learning Tool Effectively in Computer Science Courses. Olympiads in Informatics, Volume 8.

Erkki Kaila, Teemu Rajala, Mikko-Jussi Laakso, Rolf Lindén, Einari Kurvinen, Ville Karavirta and Tapio Salakoski. 2015. Comparing student performance between traditional and technologically enhanced programming course. Proceedings of the Seventeenth Australasian Computing Education Conference (ACE2015), Sydney, Australia.

Hansi Keijonen, Jaakko Kurhila, and Arto Vihavainen. 2013. Carry-on effect in extreme apprenticeship. In Frontiers in Education Conference, 2013 IEEE, pp. 1150-1155. IEEE, 2013.

Mikko-Jussi Laakso, Teemu Rajala, Erkki Kaila and Tapio Salakoski. 2008. The Impact of Prior Experience in Using a Visualization Tool on Learning to Program. Proceedings of CELDA 2008, Freiburg, Germany: 129–136.

Mikko-Jussi Laakso. 2010. Promoting Programming Learning. Engagement, Automatic Assessment with Immediate Feedback in Visualizations. TUCS Dissertations no 131.

Mikko-Jussi Laakso, Erkki Kaila and Teemu Rajala. 2016. ViLLE – Designing and Utilizing a Collaborative Education Tool. Submitted for publication into British Journal of Educational Technology.

Cynthia Bailey Lee, Saturnino Garcia and Leo Porter. 2013. Can peer instruction be effective in upper-division computer science courses?. ACM Transactions on Computing Education (TOCE), 13(3), 12.

Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. ACM SIGCSE Bulletin 33, no. 4 (2001): 125-180.

Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. 2002. The effects of pair-programming on performance in an introductory programming course. In ACM SIGCSE Bulletin, vol. 34, no. 1, pp. 38-42. ACM, 2

Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. 2003. Improving the CS1 experience with pair programming. In Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03). ACM, New York, NY, USA, 359-362. DOI=http://dx.doi.org/10.1145/611892.612006

Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger and J. Ángel Velázquez-Iturbide. 2002. Exploring the Role of Visualization and Engagement in Computer Science Education. In Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, 35, 2, 131-152.

Pavol Navrat and Jozef Tvarozek. 2014. Online programming exercises for summative assessment in university courses. In Proceedings of the 15th International Conference on Computer Systems and Technologies (CompSysTech '14), Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, 341-348. DOI=http://dx.doi.org/10.1145/2659532.2659628

Uolevi Nikula, Orlena Gotel, and Jussi Kasurinen. 2011. A Motivation Guided Holistic Rehabilitation of the First Programming Course. Trans. Comput. Educ. 11, 4, Article 24 (November 2011), 38 pages. DOI=http://dx.doi.org/10.1145/2048931.2048935

Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In Proceedings of the 8th Australasian Conference on Computing Education-Volume 52 (pp. 157-163). Australian Computer Society, Inc..

Richard E. Pattis. 1993. The “procedures early” approach in CS 1: a heresy. ACM SIGCSE Bulletin 25.1 (1993): 122-126.

Ruth Raitman, Naomi Augar and Wanlei Zhou. 2005. Employing Wikis for Online Collaboration in the E-Learning Environment: Case Study. Proceedings of the Third International Conference on Information Technology and Applications (ICITA’05), Sydney, Australia.

Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila and Tapio Salakoski. 2008. Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. Journal of Information Technology Education: Innovations in Practice, IIP, Volume 7: 15–32.

Teemu Rajala, Erkki Kaila, Mikko-Jussi Laakso and Tapio Salakoski. 2009. Effects of Collaboration in Program Visualization. Appeared in the Technology Enhanced Learning Conference 2009 (TELearn 2009), October 6 to 8, 2009, Academia Sinica, Taipei, Taiwan

Teemu Rajala, Erkki Kaila, Johannes Holvitie, Riku Haavisto, Mikko-Jussi Laakso and Tapio Salakoski. 2011. Comparing the collaborative and independent viewing of program visualizations. Frontiers in Education 2011 conference, October 12-15, Rapid City, South Dakota, USA.

Karen Renaud and Quintin Cutts. 2013. Teaching human-centered security using nontraditional techniques. ACM Transactions on Computing Education (TOCE) 13.3 (2013): 11.

Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. Software Engineering, IEEE Transactions on 37.4 (2011): 509-525.

Namita Sarawagi. 2013. "Flipping an introductory programming course: yes you can!." Journal of Computing Sciences in Colleges 28.6 (2013): 186-188.

Carsten Schulte, Johannes Magenheim, Jörg Niere and Wilhelm Schäfer. 2003. Thinking in Objects and their Collaboration: Introducing Object-Oriented Technology, Computer Science Education, 13(4), 269-288

Judy Sheard, Simon, Angela Carbone, Donald Chinn, Mikko-Jussi Laakso, Tony Clear, Michael de Raadt, Daryl D'Souza, James Harland, Raymond Lister, Anne Philpott and Geoff Warburton. 2011. Exploring programming assessment instruments: a classification scheme for examination questions. In Proceedings of the seventh international workshop on Computing education research (pp. 33-38). ACM.

Simon, Judy Sheard, Daryl D'Souza, Mike Lopez, Andrew Luxton-Reilly, Iwan Handoyo Putro, Phil Robbins, Donna Teague, and Jacqueline Whalley. 2015. How (not) to write an introductory programming exam. In proceedings of the Seventeenth Australasian Computing Education Conference (ACE2015), Sydney, Australia.

Beth Simon, Michael Kohanfars, Jeff Lee, Karen Tamayo and Quintin Cutts. 2010. Experience report: peer instruction in introductory computing. In Proceedings of the 41st ACM technical symposium on Computer science education (pp. 341-345). ACM.

Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A systematic review of approaches for teaching introductory programming and their influence on success. In Proceedings of the tenth annual conference on International computing education research (ICER '14). ACM, New York, NY, USA, 19-26. DOI=http://dx.doi.org/10.1145/2632320.2632349

James V. Wertsch, 1985. Vygotsky and the social formation of mind. Harvard University Press.

Qiyun Wang. 2009. Design and evaluation of a collaborative learning environment. Computers & Education 53(4), 1138-1146.