Comparing student performance between traditional and technologically enhanced programming course

Erkki Kaila, Teemu Rajala, Mikko-Jussi Laakso,
Rolf Lindén, Einari Kurvinen, Ville Karavirta, Tapio Salakoski
Department of Information Technology &
University of Turku Graduation School (UTUGS)
University of Turku
20014 Turun yliopisto, Finland
{ertaka, temira, milaak, rolint, emarkur, visaka, sala} @utu.fi

Abstract

Educational technology can potentially be used to engage students deeper into learning process, and hence improve the motivation and the learning results. In this paper, we present a study, where an introductory programming course was renewed by using a collaborative learning tool called ViLLE holistically throughout the course. The redesign was done in three main areas: first, half of the lectures were replaced with tutorial sessions, where students completed automatically assessed tasks in collaboration with other students. Second, remaining lectures were accompanied with a group of exercises designed to emphasize the topics introduced. We also collected feedback via short survey after each lecture to find out which topics or issues needed to be addressed again later. Third, the exam was changed into electronic version with automatically assessed programming tasks and questions. When the results of the redesigned course were compared to earlier, traditional instance of the course, we found out, that the pass rates increased significantly, while the average grade remained the same. The results are even more remarkable since the exam in the technologically enhanced course was more complicated than in the earlier instance. Hence, we can conclude that engaging students into active and collaborative learning process has highly positive effect on pass rates, although individual factors cannot be isolated with this many changes in the course design.

Keywords: Programming courses, Introductory programming, Educational technology, Learning environments, Technology adaptation, Student performance

1 Introduction

The educators and researchers in computer science are constantly trying to come up with better means for teaching programming. There have been several studies conducted (see e.g. McCracken et al., 2001, Lahtinen et al., 2005) about the state of programming learning, and in general they seem to come up with worrisome results: the students seem to lack motivation, and the high dropout rates and poor results seem to indicate that there is a lot to do to improve the teaching. Still, limited teacher resources as well as the limited time reserved in curriculum make the course improvement challenging.

In education, active learning is generally considered as a valid method for engaging students and for improving motivation and results (Freeman et al., 2014). According to constructivist learning theories (see e.g. Papert, 1980, Moons et al., 2013), the knowledge can be constructed by actively participating in the learning process. In programming education this generally means that writing programs and taking other suitable assignments is highly useful in programming education. However, the teachers’ workload for assessing several programming assignment in crowded courses can be too heavy.

Educational technology can be used to move the workload away from the course personnel. Automatic assessment and immediate feedback (see e.g. Laakso, 2010) can be effectively used to utilize actively engaging tasks, such as programming assignments. Instead of providing feedback from a few programming assignments in a traditional course, it is possible to offer dozens of automatically assessed tasks by utilizing a novel approach. This means, that the students can be engaged into active learning effectively throughout the course, which presumably means better learning results.

In this paper, we present a redesign of a typical programming course. The change took place between instances of 2011 and 2012. In the redesign the focus was on changing the focus from passive listening into active participation by utilizing educational technology and collaboration. The factors concerning the redesign are discussed as well as the methodology used. Then the performance of two instances of the courses, one right before the redesign and one after, is discussed in the scope of pass rates and course averages.

2 Related Work

As stated in a multinational, multi-institutional study by McCracken et al. (2001), novice programmers lack both motivation and sufficient skills for basic programming after introductory courses. According to Tan et al (2009), the lack of understanding the basic concepts reduces novice programmers’ interests for further exploration and self-experimentation in programming. They also state,
that novices prefer examples and “drill-practice method”, while conventional lectures lead to decreased interest in subject. Lahtinen et al. (2005) surveyed more than 500 students about their difficulties in learning, and found out, that the novice programmers found example programs as most helpful material, and working on exercises most helpful study method for learning to program.

Caspersen and Bennedsen (2007) present a proposition of designing an introductory programming course based on cognitive science and educational psychology. They argue that the cognitive load theory and cognitive skill acquisition play an important part in emphasizing a pattern-based approach to learning. The authors present guidelines in instructional design that they have successfully utilized to redesign the course. Hall et al. (2013) utilized tutorial based learning in the CS course for three weeks, and concluded, that both, tutorials and lectures, should be combined in the course.

Crescenzi and Nocentini (2007) present a two year experiment of utilizing educational technology – namely an algorithm visualization tool – in a programming course. The feedback from students was mainly positive. Still, as reported by Saunders & Kelmming (2003), when technology is integrated into programming course, the students may actually find the module harder, though the performance is improved. According to Rajaravivarma (2005), a games-based approach can be used to emphasize problem solving and logical thinking. In general, engaging students into active learning seems to have a positive effect on motivation and performance.

Utilizing educational technology in a programming course might solve several problems concerning student performance and motivation. There are various learning environments that can be utilized in courses. First, there are the course management systems, such as Moodle (see e.g. Cole et al. 2008) or Blackboard (Bradford et al. 2007). Still, these are traditionally used to manage courses and materials, and in lesser extent to engage students with exercises. Typical examples of exercise-based tools are various visualization tools developed over the recent years. With these tools the users can illustrate the execution of algorithms (see e.g. Grissom et al. 2003, Hundhausen et al. 2007, Malmi et al. 2004) or programs (see e.g. Kannusmäki et al. 2004, Kölling et al. 2003, Oechsle et al. 2002). The visualization is often accompanied with tasks to perform as well.

3 ViLLE

ViLLE is a collaborative learning environment, with focus on exercise-based learning. It supports a variety of exercise types designed for computer science, mathematics, languages and for other topics. All exercises and courses created in ViLLE can be shared with all other teachers registered to system. For CS education, ViLLE supports a variety of programming languages, including for example Java, Python, C++ and C#.

ViLLE supports collaboration in two ways: first, it enables students to work together with one computer, solving the exercises in collaboration. This method
utilizes the best practices of pair programming (see e.g. 
McDowell et al., 2002, Beck & Andres, 2004.), but can 
be utilized with other types of exercises as well. Second, 
all resources (courses, exercises and tutorials) created in 
ViLLE can be shared with other teachers easily. This 
means, that it can be used for distributing best practices 
with other educators.

The exercise types found most suitable for the course 
redesign are

- **Coding exercise:** an exercise where a student is 
supposed to write a program or a missing part of 
the program code in given programming 
language. The solution is tested against model 
solution provided by the teacher, and the test 
cases can be randomly parameterized.

- **Robot exercise:** a special version of coding 
exercise, where a student needs to write a 
program that controls a robot crane. The goal is 
to move a number of boxes into their target positions (Figure 1).

- **Visualization exercise:** an exercise where the 
program code is executed one step at a time, and 
the execution is visualized with various 
components – including variable values, object 
states and call stack. The execution is 
accompanied with multiple choice questions, 
open questions and graphical array questions.

- **Simulation exercise:** an exercise where student 
needs to simulate the state of the program one 
step at a time by creating variables and objects, 
changing their values and references and 
handling the methods in the call stack.

- **Code sorting:** also known as Parson’s puzzles 
(Parsons et al. 2006). A student needs to 
organize the shuffled program code lines into the 
correct order according to given task. The 
solution can be visualized after the sorting, if 
there are no errors in the program.

- **General sorting:** an exercise where a student 
needs to sort or connect objects as required. For 
example, connecting result values with 
expressions, or value ranges with object types.

- **Quiz:** contains multiple choice questions and 
open questions.

We have previously researched the usage of ViLLE in 
various studies with promising results. As shown in Kaila 
et al. (2009), ViLLE can be used effectively to enhance 
learning in various different setups and with different 
methods. The effect achieved on controlled setups was 
transferred into course-long usage in Kaila et al. (2010) 
and Kaila et al. (2014), where we demonstrated, that 
student performance can be significantly improved if 
ViLLE is integrated holistically into the course.

The complete description of the environment as well 
as more use cases can be found in the ViLLE system 
paper (Laakso et al, 2014), and at ViLLE home page 

4 Course redesign

*Introduction to algorithms and programming* is a 
compulsory programming course for first year CS majors 
at University of Turku. The course contains fundamental 
programming concepts – such as variables, conditional 
statements, repetition, methods and arrays – in Java. In 
addition to CS majors, several other students from the 
faculty take the course as mandatory part of their minor 
studies. For most students, the course is the first actual 
programming course, though some very basic concepts of 
programming in Python are covered in an introductory 
course before that. Course lasts for eight weeks, and 5 
ECTS are awarded for passing it. The course 
methodology was thoroughly redesigned between 
instances of 2011 and 2012 (from now on C2011 and 
C2012). In this section, the differences between instances 
are presented.

4.1 Facilitating active learning with tutorials

The first, and probably the most important, step was to 
introduce a concept of more active learning by using 
tutorials. In the 2011 instance of the course, there were 
two 2-hour lectures each week. In C2012, one of the 
lectures each week was replaced with a tutorial-based 
active learning session. The tutorials were created in 
ViLLE, and consisted of different types of assignments 
combined with related learning material such as text, 
tables and images. Hence, each week consisted of a two-
hour lecture about the topic in hand and a two-hour 
tutorial session, where the topics presented at the lecture 
were rehearsed. In total, seven tutorials were prepared:

1. Course introduction, advancing from Python to 
Java
2. Variables, Strings and conditional statements
3. Loops
4. Methods
5. Arrays
6. Using existing classes and modules
7. Summary about all topics

The tutorial sessions were organized in a lecture hall, 
where students brought their own computers. The 
tutorials were taken in collaborative mode, where two 
students worked on the same computer. Both students 
were awarded points from each solution. The controller – 
i.e. the student using mouse and keyboard – was switched 
every fifteen minutes to ensure active participation of 
both students. Active discussion was encouraged, and at 
least four members of course personnel were present in 
each session to assist the students with their possible 
problems.

Each tutorial consisted of nine to thirteen ViLLE 
assignments accompanied with learning material, adapted 
from the lecture slides. Roughly half of the assignments 
were coding exercises, while the other part consisted of 
visualization, code sorting, simulation and quizzes. An 
example of tutorial view is displayed in Figure 2.

Each tutorial was open for one week, but the 
collaborative mode was disabled after the two-hour 
session. Minimum of 50 % of maximum points as well as 
participation in at least five of the seven tutorial sessions 
were made mandatory to pass the course.
4.2 Underlining the importance of lectures with ViLLE exercises and surveys

Around three to four simple ViLLE exercises were prepared to accompany each week’s lecture. The exercises consisted of a quiz about the topics covered in lecture, a simple simulation or coding exercise, and a survey. The same three questions were included in each survey:

1. What did you learn from this week’s lecture?
2. What things remain unclear after this week’s lecture?
3. How would you develop this week’s lecture?

The data was analyzed each week before the next lecture, and the results were facilitated instantly: for example, the issues listed as unclear were summarized at the beginning of the next lecture. Also, several small technical problems were fixed based on student feedback.

Each of the exercises were scored with maximum of 5 to 10 points (surveys giving automatically full five points if answered), and the students were required to gain at least 50 % of total maximum points to participate in the final exam. In addition, ViLLE was used to automatically record the student attendances in lectures by using RFID readers in lecture halls and RFID tags given to each student. Though the participation in lectures was not mandatory, some bonus points were awarded if a student participated in all of them.

4.3 Redefining testing with electronic exam

In C2011 the final exam of the course was answered traditionally with pen and paper. Typically the exam consisted of three questions: two programming tasks (done in paper), and a theoretical question, such as an essay. In C2012 the exam was transformed into electronic form by using ViLLE. There are several benefits in using the electronic exam in a programming course:

1. An electronic exam can be automatically assessed, meaning less work for the teacher and quicker access to results for the students.
2. Programming exercises can be done by actually typing, testing and debugging the programs instead of writing them on paper.
3. More heterogeneous exercise types can be used, including for example simulation, visualization and code sorting exercises.
4. Even if manually assessed questions are to be used, they are easier to type and edit with a computer; also, the answers are easier to read and assess compared to those answered in pen and paper.

To make sure that the new instance of the course was comparable – or at least not easier – than the old one, the new electronic version of the exam was created as more challenging. A typical version of the exam in C2012 was used to show the students the types of questions they would be expected to answer.
consists of seven programming tasks – one being a robot task, a quiz measuring theoretical knowledge, and a sorting or simulation exercise. The comparison of exams is displayed in Table 1.

<table>
<thead>
<tr>
<th>C2011: Exam with pen and paper</th>
<th>C2012: Electronic exam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually assessed by teacher and course assistant(s)</td>
<td>Fully automatically assessed</td>
</tr>
<tr>
<td>Two programming tasks</td>
<td>Seven programming tasks</td>
</tr>
<tr>
<td>One theoretical question</td>
<td>One quiz of 10 MCQ / open questions and one code sorting or simulation exercise</td>
</tr>
<tr>
<td>Duration: four hours</td>
<td>Duration: three hours</td>
</tr>
</tbody>
</table>

Table 1: Comparison of exams in C2011 and C2012

The exams in C2012 were evaluated in the same scale than in C2011: minimum of 50 % of points was required to pass – i.e. to get grade 1. After that the subsequent grades of 2…5 were awarded in linear scale. The exam instances were evaluated by four individual researchers and/or teachers not affiliated with this paper, and they all agreed that the new instance is at least as difficult as the earlier instance, and very likely even more challenging.

The electronic exam was organized in one lecture hall and two computer labs at the same time. In the lecture hall the students used their own laptops, while the department computers were utilized in the computer labs. All internet traffic went through a firewall, and the only sites allowed during the exam were ViLLE and Java API. There were practically no technical difficulties during the exam, probably because the students had been familiarized with the setup during the tutorial sessions.

4.4 Other components in the course

Other changes in the course were somewhat minor. For example, C2012 contained the same number of demonstrations than C2011. In demonstrations, the students present their solutions to the programming tasks they are given a week before. In both instances at least 50 % of demonstration score needed to be achieved to attend the final exam. Only technical change in latter instance was that ViLLE was used to record the demonstration points by using aforementioned RFID readers and tags.

Also, the lectures were given in the same traditional form in both instances. However, as there was only half the number of lectures in C2012 – as half of the lecture times were used for tutorials – and the same topics needed to be covered, the lecture content needed to be compacted. Lecture content and slides were modified slightly after C2012 for the following years, based on the student feedback collected via surveys.

5 Course performance

Course performance was studied in one instance (C2011) of the traditional course as well as one instance (C2012) of the redesigned course. The instances are displayed in Table 2.

Table 2: Course instance properties

As seen on the table, the number of students starting the course was similar in both instances. However, as is typical for any programming course, not all of the students made it to the exam. The requirements to qualify for the exam are listed in Table 3.

Table 3: Requirements to qualify for course exam

The number of students who completed the required parts of the course to qualify for the exam and participated in at least one of the exams are displayed in Table 4.

Table 4: Percentage of students qualified to final exam

Notably there were more students qualified to take the final exam in the latter instance though there were more requirements to qualify.

In both courses, there were three possibilities to take an exam. A student could take the exam more than once, regardless of whether (s)he had passed the earlier exams. Combined final results in both instances are displayed in Table 5.

Table 5: Grade distribution in course instances

The distribution is visualized in Figure 3.
As seen in the table and the figure, the most significant difference in distribution among instances seems to be at the highest and the lowest grades. This is also the explanation for the grade average remaining the same; it is likely, that the active learning methods helped a lot of “worst” students to pass the course in the new instance.

The combined results for both instances are displayed at Table 6.

Table 6 Course performance results

<table>
<thead>
<tr>
<th></th>
<th>C2011</th>
<th>C2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total N</td>
<td>210</td>
<td>193</td>
</tr>
<tr>
<td>Qualify to take exam</td>
<td>70.95 %</td>
<td>86.53 %</td>
</tr>
<tr>
<td>% passed exam (of qualified)</td>
<td>75.17 %</td>
<td>93.41 %</td>
</tr>
<tr>
<td>% passed course</td>
<td>53.33 %</td>
<td>80.82 %</td>
</tr>
<tr>
<td>Grade mean (of passed)</td>
<td>3.63</td>
<td>3.57</td>
</tr>
<tr>
<td>Grade std. dev.</td>
<td>1.53</td>
<td>1.41</td>
</tr>
</tbody>
</table>

As seen on the table, all pass rates in C2012 were significantly higher than in the earlier instance. Still, the grade average remained almost the same between instances.

To confirm the difference, the grade distribution was analysed against a null hypothesis “the distribution of grades is the same across two groups”. With significance level of 0.05, we were able to reject the null hypothesis with both, Mann-Whitney U Test ($p=0.004$) and Kolmogorov-Smirnov test ($p=0.011$).

6 Discussion

Based on the student performance on the course, it seems that the redesign was quite successful. There was a significant raise in the pass rate as well as in the number of students who qualified to- and passed the final exam, respectively. Curiously, the grade average remained almost the same between the instances. It hence seems that though more students qualified for exam and passed the course, the increase in pass rate was not achieved at the cost of the performance in the final exam. Remarkably, the final exam in C2012 was likely more complex than the one on C2011: instead of two programming assignments, there were now seven. The assignments were at the same difficulty level, in fact, some of the programming tasks from C2011 were used in the exam at C2012.

What reasons may have affected the increased performance? First, the main reason is probably the introduction of active learning methods. As seen before in various studies (see e.g. Laakso, 2010) learning is more efficient when students are actively engaged into the process instead of passively following a lecture. The tutorial sessions seemed to work even better than what we have hoped for: the student feedback collected each week was mainly positive – only concerns being some technical aspects, such as network errors. The students also discussed the topic very actively during the sessions. This seems to be in line with our earlier observations (see Rajala et al. 2009, Rajala et al. 2010): we have previously shown that visualization has a more significant effect on learning when used in collaboration with another student, and that when students engage into using visualizations in collaboration, almost all discussion concerns the topic at hand.

Still, even after the redesign, half of the lectures were kept in the curriculum. The concept behind the redesign was to connect the theory and the practice by offering one lecture and one tutorial session each week. Whether transforming all lectures into active learning sessions would have had similar – or even better – effect remains unknown in the scope of this research. Still, it is definitely a concept worth testing in the future. To underline the significance of certain topics at lectures, a few ViLLE exercises were introduced after each lecture. The quiz about the introduced topics, as well as a simple coding or simulation task, was meant for summarizing the lecture. The survey about the concepts learned and improvement suggestions were also meant for students’ self-reflection: it is likely, that analysing and structuring the concepts right after the lecture can have a positive effect on learning.

Automatic assessment was a key factor in course redesign. Without the obvious benefits of automatically assessing programming assignments, the usage of exercises to this extent would have been virtually impossible. Though tutorials were primarily solved in the dedicated tutorial sessions, most of the students needed to complete some of the assignments outside the class room. Automatically assessed programming assignments also provided students a chance to redo tasks later for practice. Also, using ViLLE to try out simple Java programs is easier than starting an IDE or using compiler in command line.

Another important factor in the redesign was immediate feedback provided in ViLLE. When doing the assignments the students got feedback right after clicking the submit button. This also meant that when doing programming tasks at tutorials or weekly exercises, they could compare their results against the model solution results right after submitting, and keep on modifying their program until the results matched. As previously shown in Laakso (2010), automatic assessment and immediate feedback are the key factors when using educational technology effectively. In the earlier instance the only feedback students received from their programs was during the demonstrations. A student got to present his/her solution probably once or twice during the whole course; when compared to more than hundred automatically assessed tasks done in the latter instance,
with unlimited number of submissions, this difference can probably be seen as the most significant reason for the performance differences.

Immediate feedback was not provided in the course exam. Still, the students could see the compiler and runtime errors to bring the programming process closer to actual programming, testing and debugging. The students also had access to Java API. Moreover, the students got a subtle visual feedback if the answer was 100 percent correct: the background colour of the coding area changed to light green. Actually, this feature was left originally in exam mode as a mistake, and as such the students were not notified of it beforehand. Still, at least some of the students reported it as a nice feature in the final exam, since it helped them to confirm that their solution was correct. All programming assignments were randomly parameterized, and the test cases always checked for null and empty values and overflows, meaning that regardless of the visual feedback, the students could not test random solutions for full score. Moreover, as only automatic assessment was utilized, the students did not score any points on submissions that could not be compiled.

The student feedback on the novel features was highly positive. According to weekly surveys, the students seemed to value the tutorial based learning over all other forms of teaching. Moreover, a short survey was conducted after the course exam: according to results, the students faced no technical problems, thought that ViLLE as an exam platform was easy to use, and would recommend ViLLE usage to other students. When asked whether they would rather take the exam in paper, only 5 % of the students answered yes.

To conclude, the effect of the redesign seems to be highly positive. Still, there are various factors not considered in the scope of this paper. Most importantly, we can’t isolate the effects of individual changes in the new design. Although the change should be observed as holistic, it would be interesting to try to isolate the factors that have the best effect on learning. Also, the student feedback is not comprehensively analysed in this research, as the focus is on performance effects after the redesign. These, to name a few, are definitely factors we will observe closer in the future studies. In the future, we also plan to utilize tutorial-based learning in other CS courses, starting from the introductory course to computer science and algorithms. The method is also going to be tested at other universities, including for example RMIT at Melbourne, Australia.

7 References
Kannusmäki, O., Moreno, A., Myller, N. and Sutinen, E. 2004. What a Novice Wants: Students Using Program Visualization in Distance Programming Course. In Proceedings of the Third Program Visualization Workshop (PVW’04), Warwick, UK


