Automatically assessed electronic exams in programming courses

Teemu Rajala, Erkki Kaila, Rolf Lindén, Einari Kurvinen,
Erno Lokkila, Mikko-Jussi Laakso, Tapio Salakoski
Department of Information technology & University of Turku Graduate School (UTUGS)
University of Turku
20014 Turun yliopisto, Finland
{temira, ertaka, rolind, emakur, eolokk, milaak, sala} @utu.fi

ABSTRACT

Educational technology is nowadays utilized frequently in programming courses. Still, the final exams are mostly done using “traditional” pen-and-paper approach. In this paper, we present the adaptation of automatically assessed electronic exams in two programming courses. The first course was an introductory programming course taught using Java and the second one an advanced course about object-oriented programming. The usage of electronic exams offers several potential benefits for students, including, for example, the possibility to compile, test and debug the program code. To study the adaptation of electronic exams, we observed two instances of the courses mentioned above. Individual scores, submission counts and time spent on each task were analyzed. This data enabled us to classify the exercises in exams according to their difficulty level. This information can be used to further design exams to measure students’ knowledge and skills adequately. The analyzed data and the student feedback seem to confirm that electronic exams are an excellent tool for evaluating students in programming courses, and can be recommended to other educators as well.

Keywords

Electronic exams, Programming, Exercise difficulty, Automatic assessment.

1. INTRODUCTION

Although teachers are using more and more educational technology and new methods in their programming courses, the final exams are still mostly taken in traditional pen-and-paper form. Various studies have been conducted to improve the overall quality of programming exams (see for example Simon et al. 2015). Still, pen and paper as a medium is arguably not the best solution for testing students’ programming skills. Writing program code with pen and paper is slow and doesn’t allow testing the programs for syntax or logical errors.

ACSW '16 Multiconference, February 02-05, 2016, Canberra, Australia
© 2016 ACM. ISBN 978-1-4503-4042-7/16/02…$15.00
DOI: http://dx.doi.org/10.1145/2843043.2843062

The utilization of electronic exams offers several benefits in the programming courses. First, the students likely are used to writing code using keyboard and a proper editor that takes care of code indentation and color coding automatically. Second, if proper exam tools are selected, the solutions can be compiled, executed, tested and refactored after submission. After all, when testing for programming skills, the situation should resemble “real” programming as closely as possible. Still, most of the tools and platforms used in electronic exams are not designed to support programming.

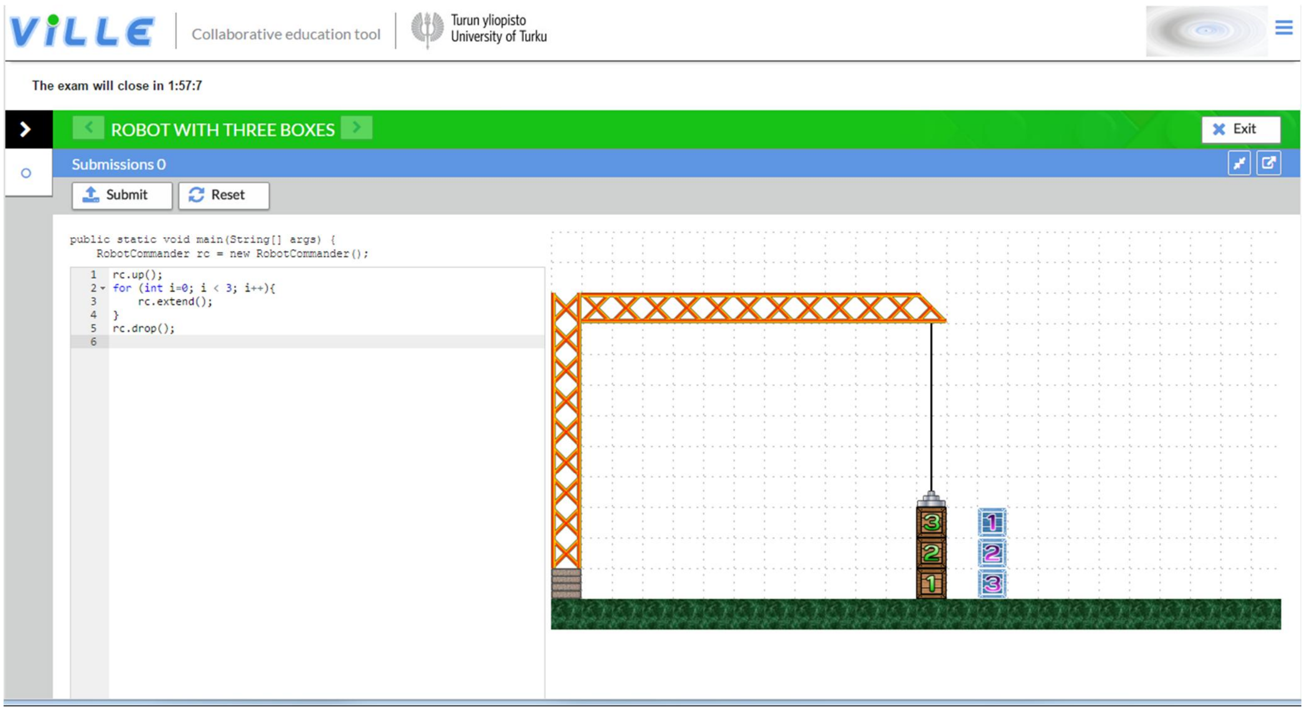
With this in mind, we have designed and implemented a collaborative education tool called ViLLE, with full support for automatically assessed programming exercises. The exercises can be used in electronic exams as well. The only difference in exams is that the immediate feedback is hidden. In this paper, we report the usage of automatically assessed exams in total of four instances of two programming courses: one introductory course and one course about object oriented programming. The scores, submission counts and the time usage are observed, and the data collected is used to identify the difficulty level of different kinds of exercises used in the exams.

The paper is structured as follows: first, the existing literature about programming education and electronic exams is reviewed. Then, ViLLE and the electronic exams are presented, followed by the research setup and course descriptions. After that, we present the results and experiences collected from conducting the exams. Finally, the results are discussed and some future possibilities for research are presented.

2. RELATED WORK

There is some existing literature about designing and conducting exams in programming courses. Daly & Waldron (2004) state, that the current methodology for assessing programming skills after the introductory courses is not sufficient. Though the study is more than ten years old, the arguments provided are still somewhat solid: pen and paper exams are not the best way to measure programming knowledge. Regardless of the tools used, designing questions at proper difficulty level is very important (see for example Harland et al. 2003). Hence, the work done for classification of the exam questions by Sheard et al. (2011) can be extremely useful when designing the exams.

Various educational tools to be used in programming courses (among others) have been developed in recent decades. Some of the



The exam will close in 1:57:7

ROBOT WITH THREE BOXES

Submissions 0

```
public static void main(String[] args) {
    RobotCommander rc = new RobotCommander();
rc.up();
for (int i=0; i < 3; i++){
    rc.extend();
}
rc.drop();
```

Figure 1. Example of ViLLE's student view in exam mode

most well-known general environments are Blackboard (Liaw, 2008) and Moodle (Dougiamas & Taylor, 2003). Typical for such environments is that they can be used for various course needs, such as delivering materials, answering exercises and for communication (such as discussion forums and news). Examples of more specific tools include different exercise systems designed for programming courses. Ihantola et al. (2010) present various tools for automatically assessing programming assignments. Visualization tools (see for example Kaila et al. 2009, Korhonen & Malmi, 2000) are typical examples of exercise-based tools designed especially for programming courses. However, the tools used in programming courses are rarely designed to be used in exams.

There are luckily some experiments done on using electronic exams to assess programming skills. Jacobson (2000) reported an experiment where the assessment was changed from traditional exams into on-computer exams. According to the author, the new assessment system measured students' programming skills more accurately. Barros et al. (2003) confirm this approach: they argue, that though the lab exams are more demanding than "traditional" assessment, the students find them fairer than, for example, group assignments. Navrat & Tvarozek (2013) replaced the whole exam with summative assessment of data collected by interactive online learning environment, and found out that the data collected can be used to predict grades accurately enough. Kuikka et al. (2014) compared Moodle, Optima, ViLLE, Soft Tutor and Tenttis as e-exam platforms in a university of applied sciences in Finland. The article lists extensively the challenges that were faced during the comparison.

3. ViLLE – COLLABORATIVE EDUCATION TOOL

ViLLE is a collaborative education tool, developed at Department of Information Technology, University of Turku (see Figure 1). The tool works as a web-based application with separate views for teachers and students. ViLLE contains several automatically assessed exercise types for programming, mathematics and for general subjects. All exercises provide immediate feedback about submission, with the possibility to make unlimited number of re-submissions. Collaboration is encouraged for both students and teachers: the registered teachers can share all of their resources with other teachers in ViLLE and the students can answer the exercises and tutorials in collaboration with other students. Moreover, other means of student interaction (such as peer reviewing other students' submissions) are supported as well.

4. ELECTRONIC EXAMS WITH AUTOMATIC ASSESSMENT

Electronic exams in ViLLE can consist of automatically assessed or manually graded assignments. In the courses included in this research, only automatically assessed tasks were decided to be used. The main reason for this was the number of students taking the exams: since there were around ten exercises and around one hundred students in each exam, the number of submissions would have been too laborious to assess manually. Moreover, automatic assessment and real-time compilation of student code enables students to fix compilation or runtime errors after submitting their code, and hence focus on creating the algorithmically correct solution instead of worrying about syntax errors. Apart from compilation and runtime errors and the program output, no feedback was provided during the exam.

Number of submissions is not limited in ViLLE's electronic exams. Instead, there is a time limit for completing the exam; in the courses described in this study, we decided to set the limit as three hours. The exam was done supervised in a lecture hall or computer lab, and the internet connection was restricted via firewall to only allow access to ViLLE and into Java API. Three exams in total were conducted for each course, and the students could participate in as many as they wanted.

Since ViLLE was originally designed to be used solely in computer science courses, it supports variety of programming languages (including for example Java, Python, C, C++ and C#). Moreover, several exercise types are designed specifically for programming and computer science courses. The exercise types selected to be used in the exams were the following:

- **Quiz** consists of multiple choice questions and short open questions. The questions can be answered in any order, and the students were able to change their answers later. The questions were mostly designed to measure code tracing skills (for example by providing a piece of program code and asking the student to predict the output) or general knowledge about programming and Java (for example by asking to provide the instruction that is used to terminate the execution of a loop). Typically, one quiz was included in each exam.
- **Coding exercise** requires students to write a program (or a missing part of a program) according to given description. The code can be compiled and executed by clicking a single button, after which ViLLE provides either compilation or runtime errors or the program output if it was executed successfully. The correctness is decided by comparing the student program output to that of the model solution. Though output comparison may seem like a simple tool, it was actually found to be powerful enough to test rather complicated tasks when used creatively.
- **Robot exercise** is a specific type of coding exercise, where the students need to write a Java program that controls a crane to move boxes to their goal positions. The crane (or a "robot") is controlled via robot object that offers methods for moving the crane's arm and picking up or dropping the boxes. The teacher can set limits for the number of statements and the steps the robot can move, and exceeding these limits subtracts the score gained. This means that instead of listing the method calls one after another the students need to utilize loops and own methods to complete the task in more clever way.
- In **Code shuffling exercise** the students need to order the shuffled code lines according to the given task. The exercise, also known as Parsons Puzzle (Parsons et al. 2006), is designed to measure the students' ability to understand the program flow and algorithm execution. Again, in the exam mode no feedback was provided, meaning that the students needed to be able to understand the execution of the program by looking at the code. Hence, the tasks were designed to be quite simple.
- **Connect the items** is a simple exercise type where the students need to drag the items in the right column in the correct locations in relation to the items in the left column. In the exams, the exercise type is usually used to test the knowledge in some general issues, such as combining the privacy definition (such as "public static void calculate()") with the correct definition (such as "public class method").

Since the ability to write programs is the most important outcome of the course, most of the exercises in the exams were decided to be coding exercises. Still, at least one quiz and one other exercise type were included in each exam. The difficulty level of tasks was designed to increase towards the latter exercises, and the exercises were planned to measure the students' skills in all topics taught throughout the course.

5. RESEARCH SETUP

The exams were utilized in two programming courses during two years, totaling to four course instances. Both courses are mandatory for all computer science majors, and are typically taken during the first academic year.

5.1 Courses and materials

The Basic Course of Programming and Algorithms (from now on Course 1) is the first programming course in the curriculum (although some fundamental issues are covered in the introductory computer science course using Python). The course covers basic programming topics in imperative paradigm using Java, including for example variables, conditional statements, repetition, methods and arrays. As a learning outcome, the students should be able to design and implement simple algorithms in Java.

The exam was structured as described in Table 1.

Table 1. The exam structure at Course 1

Exercise #	Type	Description
1	Quiz	Questions about code tracing and general questions about programming and Java
2	Coding	Task about using conditional statements, such as calculating the speeding ticket based on speed
3	Coding	Easy array question, such as finding the minimum or maximum value in an array
4	Code shuffle	Typically string operations, such as ordering charAt and substring operations in correct order to form the given word
5	Robot	ViLLE robot exercise where five boxes need to be moved into their target positions
6	Coding	Medium array question, where for example the range of items or the minimum and maximum item need to be found and returned as a new array
7	Coding	Task about string operations (such as changing the first character of each word in a sentence into upper case) OR a task about using the error handling with try-catch-finally structure
8	Coding	Difficult array question, for example finding the most frequent item in an array.
9	Coding	Question about two-dimensional arrays, for example flipping the matrix OR reversing the order of some items in the middle of an array

As seen on the table, most of the exercises were selected to be coding exercises. The first exercises are the easiest, and the latter ones probably the most difficult. 50 % of the total score of 90 (ten points per exercise) was required to pass the exam.

The Fundamentals of Object Oriented Programming (from now on Course 2) is an advanced course, taken right after Course 1. The concepts of objects and classes and writing own classes, constructors and class members are taught. Additionally, some more advanced OO concepts such as inheritance and polymorphism are also discussed. As a learning outcome, the students should be able to write their own classes and know how to utilize inheritance and interfaces when designing and writing their programs.

The exam was structured as described in Table 2.

Table 2. The exam structure at Course 2

Exercise #	**Type**	**Description**
1	Quiz	Questions about code tracing and general questions about object oriented programming techniques.
2	Coding	Task about writing constructor(s) and getter and setter methods for a class with pre-defined attributes
3	Connect the items	Task about connecting the variable and method definitions with different privacy declarations to correct descriptions.
4	Coding	Task about inheritance: a base class is given, and the student needs to write a class inheriting this base class with some new attributes and methods.
5	Coding	Task about writing a new exception class.
6	Coding	Task about writing a class that implements a given interface. For example, an interface modeling a shape might be given, and the student needs to write a class that models a cube.
7	Coding	Task about defining an enum type with given set of values. Typically something like defining an enum containing all Nordic countries.
8	Coding	Task about using try-catch mechanism to detect and (possibly) re-throw errors.
9	Coding	Task where a class definition is given and the student needs to implement either the equals or the compareTo method.
10	Coding	Task about polymorphism. Typically, a list (or a vector) containing different kinds of objects with same parent class given, and the objects need to be handled based on their type.

In both courses, two instances of the exam are observed, totaling four instances. The instances were taught during academic years of 2013-14 and 2014-15.

5.2 Utilizing ViLLE in the courses

Both courses were redesigned to utilize ViLLE for various aspects.

Enhancing active learning: For both courses, one weekly lecture was transformed into a tutorial session. In these sessions, the students answered a tutorial (a combination of automatically assessed programming exercises and learning material, such as text, images and tables) in collaboration with another student. The sessions were organized in a lecture hall, with some older students and course staff available to help the students with problems they might have encountered answering the tutorial questions.

Facilitating communication: Two weekly surveys were conducted each week in both courses: one after the lecture and one after the tutorial. In these surveys, the students were asked to describe the things they learned, things that remained unclear after the session, and ways to improve the lecture or the tutorial. The answers to surveys were briefly analyzed before the next session, and the consecutive lectures and tutorials were improved based on the feedback.

Electronic exam difficulty: The exam instances were evaluated by two individual researchers who are not affiliated with this paper. They both agreed that the exams in ViLLE were at least as difficult as earlier instances, and likely even more challenging than the previous exams,

The changes made to the courses significantly improved the pass rate and the average grade for both courses. A detailed description of the redesign can be found at Lokkila et al. (2015) and Kaila et al. (2015).

5.3 Participants

The students taking the exam were mostly first year computer science majors at the Department of Information Technology, University of Turku. A total of 478 students participated in the four exam instances. The instances are described at Table 3.

Table 3. Exam instances in the study

Exam	Course	Year	N
Exam 1.1	Course 1	2013	133
Exam 1.2	Course 1	2014	149
Exam 2.1	Course 2	2014	85
Exam 2.2	Course 2	2015	111

Though there were three exams arranged each year at each course, only the first instance per year was selected for this study. The reason for this was the low number of participants in the resit exams: almost all students had passed the course after the first exam, regardless of the course or the year.

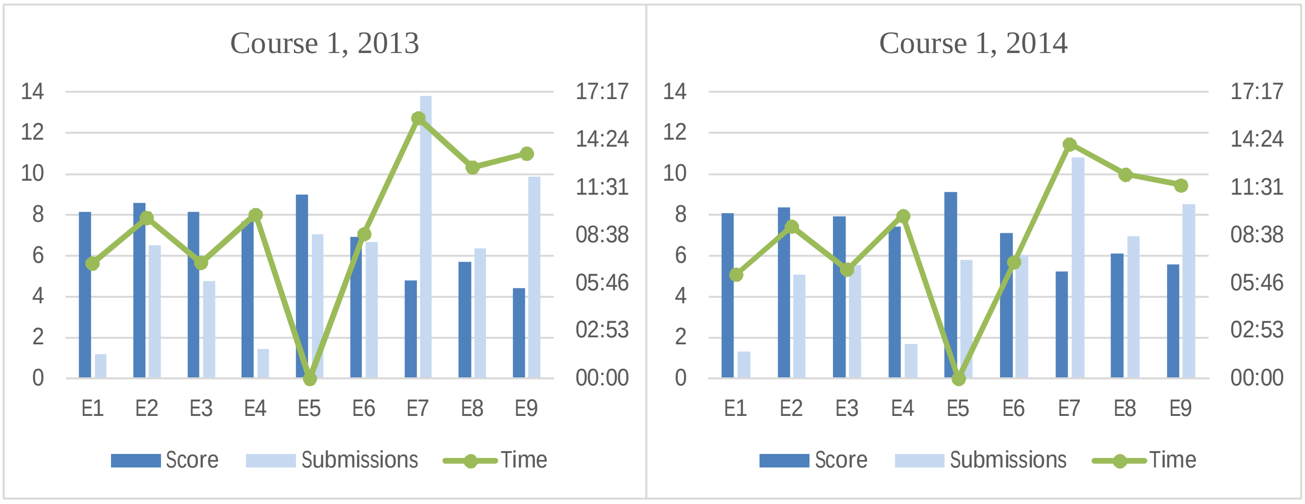


Figure 2. Score, submission and time averages of Course 1 instances

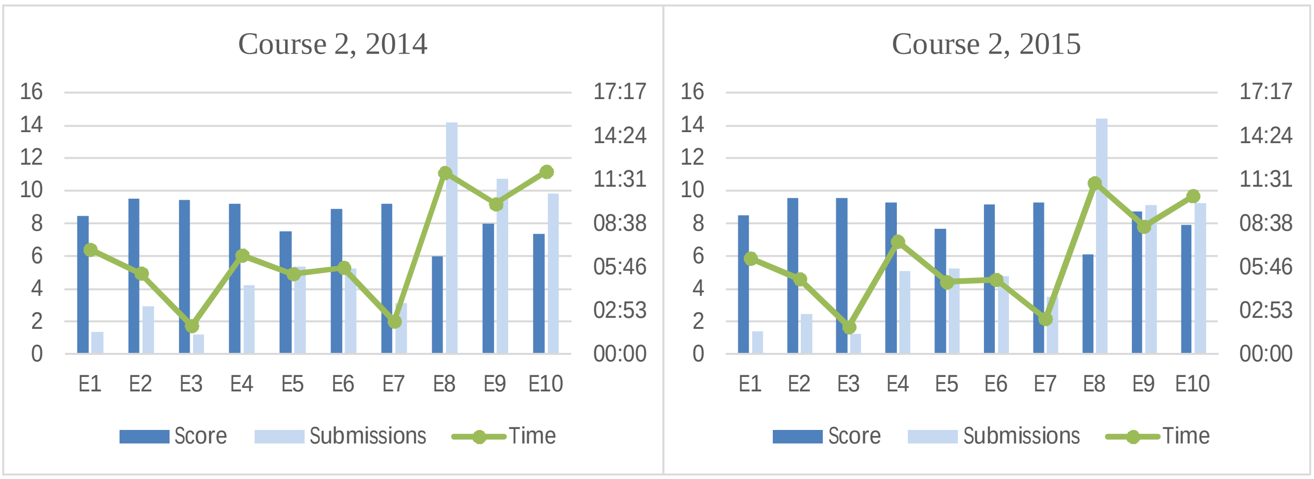


Figure 3 Score, submission and time averages of Course 2 instances

6. RESULTS

In order to estimate the difficulty level of the exercises we gathered the scores, the submission counts and the answer times for each student for each exercise in the exams. Figure 2 shows the averages of these values in both instances of Course 1.

As maximum score for each exercise in the exam was 10, the figure clearly shows that the average scores for the first six exercises were high. Unfortunately the learning platform at the time didn't record the answer times for the robot exercise (E5). Otherwise the answering times of E1-E6 are quite close to each other. The figure also shows that there is a large difference in submission counts of E1 and E4 when compared to the other exercises. All the other exercises were program code writing tasks and the students could debug the answers by submitting them, so naturally there is more submissions to those than to quiz exercises (E1) and code shuffle exercises (E4), which provided no feedback on submit.

From exercise 7 onwards, the students spent more time answering to the exercises, got lower scores and submitted their answers more. This is a clear indication that the last three questions were more difficult. Exercise 7 seems to be the most difficult one, as it has the lowest average score, the highest answer time and the most submissions of all of the exercises. The exercises in both instances were similar or as in most cases, the same, which is indicated by the similar figures in both course instances.

Figure 3 shows the same average values for the Course 2 instances. Exercise scores were scaled to a maximum of 10 points in both course instances. Similarly to Course 1 instances, the E1 and E3 submission counts are lower because they weren't program code writing tasks. Here also the last three exercises show the highest answering times and the highest submissions counts, although the score averages are quite high in E9 and E10 in 2015 instance. Thus, based on the data, the last three exercises are more difficult than E1-E7, while E8 is clearly the most difficult one.

In both courses the corresponding exercises in the consecutive instances were similar, and in most cases the same. Figures 2 and 3 also illustrate that the students performed and behaved similarly in both instances. Based on this we decided to combine the instance data to help discussing the results. Figure 4 shows the combined data for both courses.

We also gathered student feedback about taking electronic exams from both courses. Table 4 shows the results from the feedback

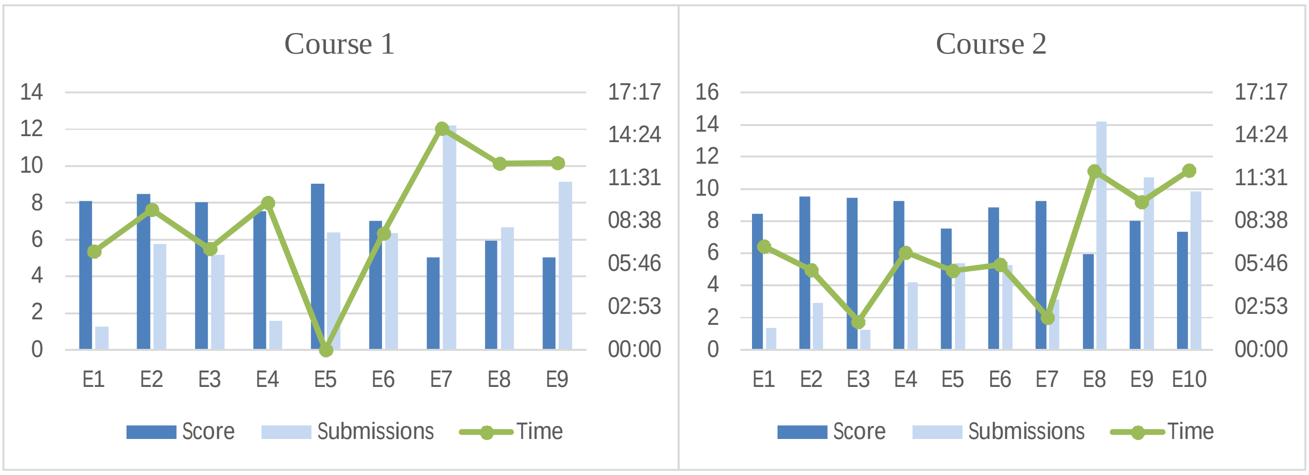


Figure 4 Exam data combined

survey. The detailed description of the survey results can be found in Rajala et al. (2015)

Table 4. Student feedback from the course exams. All questions were answered on a Likert scale of 1 to 5 (1 = totally disagree, 5 = totally agree), unless otherwise noted.

Statement (1 – strongly disagree, 5 – strongly agree)	**Avg. Course 1, 2014 (N = 130)**	**Avg. Course 2, 2015 (N = 111)**
There was enough time to take to the exam	4.45	4.86
Answering to the exam was easy	3.59	4.17
The exam application was clear to use	3.98	4.24
I would prefer to use pen-and-paper over ViLLE	1.42	1.39
ViLLE suits very well for the exam of this study module	4.46	4.64
If possible, I would like to take the exam of this study module at home	3.91	3.68
I would recommend ViLLE to other students	4.11	4.31
Technically ViLLE is an excellent solution	3.73	3.89
How would you grade ViLLE as an exam platform	3.90	4.12
I got enough guidance on how to use ViLLE before the exam	4.65	4.20
eThe difficulty of the exam content (1 – low, 5 – high)	2.75	3.27

7. DISCUSSION

It seems that the exams we designed for both courses, respectively can be used quite reliably to measure students' programming skills in the tasks given. There were practically no differences between the scores, the submissions or on the time spent on the tasks between the two instances of the same course. This also seems to indicate that although the exam structure was quite similar between the two instances, no major cases of plagiarism occurred. It is quite typical (and in some cases even encouraged) in our university that the students have a public archive for exam questions. However, questions in electronic exams are more difficult to share, especially since the students could not access the questions after the exam was closed. Sharing was also made more difficult by parameterizing some of the questions.

When two instances of the courses were combined (see Figure 4), we could observe the difficulty level of the questions. The questions can be roughly divided into three categories:

1) Easier coding tasks: the coding exercises where higher scores could be obtained with less effort (either with fewer submissions or in shorter time). In Course 1 exercises E2 and E3 fall under this category. Similarly, in Course 2 exercises E2, E4 and E7 can be classified as easier tasks.
2) More difficult coding tasks: the coding exercises where a higher number of submissions (or more time spent answering) was needed to obtain the score. Most of the coding exercises in both courses fall under this category.
3) Non-coding tasks: as seen on Figure 4, significantly lower number of submissions were made to all other exercise types. The average number of submissions done to exercises E1 (quiz) and E4 (code shuffling exercises) in Course 1 was just above one. The same applies to exercises E1 (quiz) and E3 (connect the items) in Course 2.

Hence, from the students' point of view, the ability to test and resubmit code in the coding exercises is probably the most beneficial factor of the electronic exam. The automatically assessed quizzes and the sorting exercises definitely decrease the teacher's workload drastically, but do not offer similar advantages to the students.

When observing the difficulty level of individual exercises, we can see that E7 seemed to be the most difficult one in Course 1. The exercise was about combining different string operations (such as capitalizing the first letter of each word in a sentence) or about handling errors with the try-catch structure. Somewhat surprisingly, the average number of submissions done to E7 was a lot higher than that of exercises 8 and 9.There are likely two possible explanations for this. First, it is possible that some of the weaker students got stuck in E7 and never reached the final exercises. While there is no way to tell if the students tried to solve the exercises E8 or E9 without submitting them, we did find out that roughly two tenths of the students who answered to E7 did not have any submissions for E9, and half of those students had no submissions to E8 either. Second, E7 might have been too demanding, as it required of the students to apply multiple learned skills in the same exercise. This claim is backed up by a qualitative analysis on the types of errors that were present in the assignment submissions for assignment E7 (omitted). In most of the cases, errors were due to misconceptions about variable typing, arrays and variable initialization. For instance, the students tried to test the equality of strings and integers, or refer to an array element simply by referring to the index variable. Similar rudimentary errors were absent from Course 2. The qualitative analysis also revealed few odd cases where the student displayed iterative behavior, or tried unsuccessfully to fool the automatic checking mechanism by inserting the desired answer directly to the output stream. The difficulty level of individual tasks can be observed on Table 5, where time on task and the number of submissions are shown related to score obtained.

Table 5. Course 1 score per minute and score per submission count values

	Score per minute	Score per submission count
E1	1.23	6.46
E2	0.90	1.47
E3	1.18	1.55
E4	0.76	4.77
E5		1.41
E6	0.90	1.10
E7	0.34	0.41
E8	0.47	0.89
E9	0.40	0.55
Total	0.80	1.18

According to the data collected in Course 2 exam, the most difficult exercise was E8. The relation between time on task and the number of submissions to score obtained is displayed on Table 6. The exercise was (again) about using the try-catch mechanism to catch errors when reading data from a stream. Based on the student submissions, the ones who didn't manage to solve the exercise had problems understanding on how to read from a stream of data. The usual mistake was to use a for-loop to iterate through the stream based on the length of the data. Additionally, in some instances the student had correctly used the try – catch mechanism, but didn't handle the exception in the catch block. When evaluating the path leading to a correct submission, the students usually started with a solution that didn't include any exception handling, then added the catch mechanism, and finally tried to handle the exception. Again, it is possible that some students got stuck in the eighth exercise and never advanced to latter ones. Still, in both courses, the error handling can be identified as the most difficult topic. Since the ability to utilize the error handling mechanism is essential in Java, more time and effort should definitely be addressed in teaching it. Especially the combination of exception handling mechanisms and loops should be covered with more detailed examples.

Table 6. Course 2 score per minute and score per submission count values

	Score per minute	Score per submission count
E1	2.44	12.37
E2	1.78	3.24
E3	2.49	3.92
E4	1.41	2.19
E5	0.71	0.70
E6	1.55	1.68
E7	2.11	1.48
E8	0.50	0.42
E9	0.81	0.75
E10	0.61	0.75
Total	1.20	1.40

The feedback collected from the students was mainly positive. The students seemed to prefer the electronic exam over the pen and paper version and they thought that ViLLE was very well adjusted for a programming course exam. Notably, the students seemed to think that there was plenty of time to take the exam (the average for Course 2 being as high as 4.86 out of 5). This is remarkable, since in the older instances of the course a pen and paper version of the exam only contained two or three programming tasks, and the time reserved was an hour more.

The electronic exam format works well at least for the subject of introductory programming. In addition to some generally used automatically assessed exercise types including quizzes and drag and drop sorting and matching, the possibility for providing more authentic programming environment to answer to the programming tasks is a clear advantage. The programming assignments in the introductory programming course exams are often quite short and simple, so writing test cases to automatically assess the answers is usually straightforward. Although preparing the exam takes more time, the time saved in the assessment phase is manifold. For example, nearly 5000 answers were assessed in the exams presented in this paper, and this number includes only the final answers to the assignments. Solving a programming problem usually includes several tries and one of the major benefits of the electronic platform is that all submissions leading to the final submission can be stored. This offers vast possibilities for studying how the problem solving process works and in more general how the understanding of algorithmic thinking evolves.

However, some critique for the setup can be given: to get a more comprehensive understanding about the difficulty level of the individual exercises, students' success in the course before the exam could be observed. In the future, we plan to collect more data from similar programming courses (and from more instances of the courses discussed in this study), and use that data to develop a model for properly categorizing the exam exercises based on their difficulty level. When combined with the existing research about exam question difficulty levels, (see for example Harland et al.

2003, Sheard et al. 2011) the model can be used by educators to develop automatically assessed exams that reliably and effectively test students' programming knowledge and skills.

8. REFERENCES

[1] Barros, J. P., Estevens, L., Dias, R., Pais, R. & Soeiro, E. 2003. Using lab exams to ensure programming practice in an introductory programming course. *SIGCSE Bull.* 35, 3 (June 2003), 16-20

[2] Daly, C. & Waldron, J. 2004. Assessing the assessment of programming ability. *SIGCSE Bull.* 36, 1 (March 2004)

[3] Dougiamas, M., & Taylor, P. (2003). Moodle: Using learning communities to create an open source course management system. In World conference on educational multimedia, hypermedia and telecommunications (Vol. 2003, No. 1, pp. 171-178).

[4] Harland, J., D'Souza, D, & Hamilton, M. 2013. A comparative analysis of results on programming exams. In *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136* (ACE '13)

[5] Jacobson, N. 2000. Using on-computer exams to ensure beginning students' programming competency. *SIGCSE Bull.* 32,

[6] Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010, October). Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research (pp. 86-93). ACM

[7] Kaila, E., Rajala, T., Laakso, M.-J. & Salakoski, T. 2009. Effects, Experiences and Feedback from Studies of a Program Visualization Tool. Informatics in Education, 8, 1, 17-34.

[8] Kaila, E., Kurvinen, E., Lokkila, E., Laakso, M.-J., Salakoski, T. (2015). Redesigning an Object-Oriented Programming Course. Submitted to ACM Transactions on Computing Education.

[9] Korhonen, A. & Malmi, L. 2000. Algorithm simulation with automatic assessment. ACM SIGCSE Bulletin 32.3 (2000): 160-163.

[10] Liaw, S. S. (2008). Investigating students' perceived satisfaction, behavioral intention, and effectiveness of e-learning: A case study of the Blackboard system. Computers & Education, 51(2), 864-873.

[11] Lokkila, E., Kaila, E., Karavirta, V., Salakoski, T. & Laakso, M.-J. (2015) Redesigning Introductory Computer Science Courses to Use Tutorial-Based Learning. ICEE 2015 – International Conference on Engineering Education.

[12] Navrat, P. & Tvarozek, J. 2014. Online programming exercises for summative assessment in university courses. In *Proceedings of the 15th International Conference on Computer Systems and Technologies* (CompSysTech '14)

[13] Parsons, D. & Haden, P. 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*. Australian Computer Society, Inc., 2006.

[14] Rajala, T., Lokkila, E., Lindén, R. & Laakso, M.-J. 2015. Student Feedback about Electronic Exams in Introductory Programming Courses. Proceedings of EDULEARN15 – 7th International Conference on Education and New Learning Technologies. IATED Academy.

[15] Sheard, J., Simon, Carbone, A., Chinn, D., Laakso, M.-J., Clear, T., de Raadt, M., D'Souza, D., Harland, J., Lister, R., Philpott, A &Warburton, G. 2011. Exploring programming assessment instruments: a classification scheme for examination questions. In *Proceedings of the seventh international workshop on Computing education research* (ICER '11). ACM, New York, NY, USA, 33-38

[16] Simon, Sheard, J., D'Souza, D., Lopez, M., Luxton-Reilly, A., Putro, I. H., Robbins, P., Teague, D., & Whalley, J. 2015. How (not) to write an introductory programming exam. In *Proceedings of the 17th Australasian Computing Education Conference (ACE 2015)* (Vol. 27, p. 30).