TUCS

Erkki Kaila

Utilizing Educational Technology in Computer Science and Programming Courses

Theory and Practice

Turku Centre *for* Computer Science

TUCS Dissertations
No 230, February 2018

Utilizing Educational Technology in Computer Science and Programming Courses

Theory and Practice

Erkki Kaila

To be presented, with the permission of the Faculty of Science and Engineering of the University of Turku, for public criticism in Auditorium XXI on February 17th, 2018, at 12 noon.

University of Turku
Department of Future Technologies
Vesilinnantie 5, 20500 Turku, Finland

2018

SUPERVISORS

Professor Tapio Salakoski, Ph.D.
Department of Future Technologies
University of Turku
Finland

Adjunct Professor Mikko-Jussi Laakso, Ph.D.
Department of Future Technologies
University of Turku
Finland

REVIEWERS

Associate Professor Judithe Sheard, Ph.D.
Faculty of Information Technology
Monash University
Australia

Adjunct Professor Jarkko Suhonen, Ph.D.
School of Computing
University of Eastern Finland
Finland

OPPONENT

Professor Mike Joy
Department of Computer Science
University of Warwick
United Kingdom

ISBN 978-952-12-3677-8
ISSN 1239-1883

To my Loved Ones
and
One Love

ABSTRACT

There is one thing the Computer Science Education researchers seem to agree: programming is a difficult skill to learn. Educational technology can potentially solve a number of difficulties associated with programming and computer science education by automating assessment, providing immediate feedback and by gamifying the learning process. Still, there are two very important issues to solve regarding the use of technology: what tools to use, and how to apply them?

In this thesis, I present a model for successfully adapting educational technology to computer science and programming courses. The model is based on several years of studies conducted while developing and utilizing an exercise-based educational tool in various courses. The focus of the model is in improving student performance, measured by two easily quantifiable factors: the pass rate of the course and the average grade obtained from the course.

The final model consists of five features that need to be considered in order to adapt technology effectively into a computer science course: active learning and continuous assessment, heterogeneous exercise types, electronic examination, tutorial-based learning, and continuous feedback cycle. Additionally, I recommend that student mentoring is provided and cognitive load of adapting the tools considered when applying the model. The features are classified as core components, supportive components or evaluation components based on their role in the complete model.

Based on the results, it seems that adapting the complete model can increase the pass rate statistically significantly and provide higher grades when compared with a “traditional” programming course. The results also indicate that although adapting the model partially can create some improvements to the performance, all features are required for the full effect to take place.

Naturally, there are some limits in the model. First, I do not consider it as the only possible model for adapting educational technology into programming or computer science courses. Second, there are various other factors in addition to students' performance for creating a satisfying learning experience that need to be considered when refactoring courses. Still, the model presented can provide significantly better results, and as such, it works as a base for future improvements in computer science education.

TIIVISTELMÄ

Ohjelmoinnin oppimisen vaikeus on yksi harvoja asioita, joista lähes kaikki tietojenkäsittelyn opetuksen tutkijat ovat jokseenkin yksimielisiä. Opetusteknologian avulla on mahdollista ratkaista useita ohjelmoinnin oppimiseen liittyviä ongelmia esimerkiksi hyödyntämällä automaattista arviointia, välitöntä palautetta ja pelillisyyttä. Teknologiaan liittyy kuitenkin kaksi olennaista kysymystä: mitä työkaluja käyttää ja miten ottaa ne kursseilla tehokkaasti käyttöön?

Tässä väitöskirjassa esitellään malli opetusteknologian tehokkaaseen hyödyntämiseen tietojenkäsittelyn ja ohjelmoinnin kursseilla. Malli perustuu tehtäväpohjaisen oppimisjärjestelmän runsaan vuosikymmenen pituiseen kehitys- ja tutkimusprosessiin. Mallin painopiste on opiskelijoiden suoriutumisen parantamisessa. Tätä arvioidaan kahdella kvantitatiivisella mittarilla: kurssin läpäisyprosentilla ja arvosanojen keskiarvolla.

Malli koostuu viidestä tekijästä, jotka on otettava huomioon tuotaessa opetusteknologiaa ohjelmoinnin kursseille. Näitä ovat aktiivinen oppiminen ja jatkuva arviointi, heterogeeniset tehtävätyypit, sähköinen tentti, tutoriaalipohjainen oppiminen sekä jatkuva palautesykli. Lisäksi opiskelijamentoroinnin järjestäminen kursseilla ja järjestelmän käyttöönottoon liittyvän kognitiivisen kuorman arviointi tukevat mallin käyttöä. Malliin liittyvät tekijät on tässä työssä lajiteltu kolmeen kategoriaan: ydinkomponentteihin, tukikomponentteihin ja arviontiin liittyviin komponentteihin.

Tulosten perusteella vaikuttaa siltä, että mallin käyttöönotto parantaa kurssien läpäisyprosenttia tilastollisesti merkittävästi ja nostaa arvosanojen keskiarvoa ”perinteiseen” kurssimalliin verrattuna. Vaikka mallin yksittäistenkin ominaisuuksien käyttöönotto voi sinällään parantaa kurssin tuloksia, väitöskirjaan kuuluvien tutkimusten perusteella näyttää siltä, että parhaat tulokset saavutetaan ottamalla malli käyttöön kokonaisuudessaan.

On selvää, että malli ei ratkaise kaikkia opetusteknologian käyttöönottoon liittyviä kysymyksiä. Ensinnäkään esitetyn mallin ei ole tarkoituskaan olla ainoa mahdollinen tapa hyödyntää opetusteknologiaa ohjelmoinnin ja tietojenkäsittelyn kursseilla. Toiseksi tyydyttävään oppimiskokemukseen liittyy opiskelijoiden suoriutumisen lisäksi paljon muitakin tekijöitä, jotka tulee huomioida kurssien uudelleensuunnittelussa. Esitetty malli mahdollistaa kuitenkin merkittävästi parempien tulosten saavuttamisen kursseilla ja tarjoaa sellaisena perustan entistä parempaan opetukseen.

ACKNOWLEDGMENTS

It has been my privilege to be supervised by two very capable researchers. Both Tapio and Mikko have given me excellent guidance when needed, and known when to step aside and let me learn from my failures or wins on my own, when necessary. This thesis is just a short trip in a long journey, and I know I can rely on mentoring of my supervisors in the future as well. I also like to thank my pre-examiners Judithe Sheard and Jarkko Suhonen for their expertise in reviewing this thesis and for their deeply analytic and extremely helpful comments on how to complete the work. I like to thank Mike Joy for being my opponent in the public defense..

Teemu Rajala has been my brother in arms since we started the studies. We co-created ViLLE together, started our PHD studies together and have continued to work together for almost a decade in our department. I have known Teemu for almost twenty years, and he has always been the most trustworthy and technically high-skilled colleague, and a dear friend. The same can be said of Ilkka Sillanpää, who helped me and Teemu to create the first version of ViLLE before leaving for other duties.

Einari Kurvinen is one of the most enthusiastic and pedagogically skilled researchers I have worked with. Due to sharing similar views to teaching and research (and lately, due to ever-growing number of commitments outside the office, a number of noisy train compartments, shady hotel rooms and a fair amount of pints), Einari has become the colleague I seem to be calling out the most. I have worked with many other excellent researchers as well during the last decade. Rolf Lindén, Erno Lokkila, Ashok Kumar, Ville Karavirta, Johannes Holvitie, Ari Korhonen, Lauri Malmi, Jani Turtiainen and Nikke Leskelä are just a few examples of these highly skilled people.

This thesis would not have been possible without all the lovely people in the ViLLE Team throughout these years. I have been lucky enough to form a team with Rolf, Erno, Markus Kitola, Peter Larsson, Eero Suvanto and Tommy Johansson. Riku Haavisto, Riku Kajasilta, Tommi Tapaninaho, Ali Leino, Niko Hellgren and numerous others have been continuing to develop the system tirelessly for it to become what it is today. Essi Tamminen, Marika Parviainen, Aleksi Hermonen, Säde Kankare and Eeva Laatikainen have shown their creative talents in creating excellent content for learning.

I have been lucky to work in the department where I can have collaborated with so many true experts, whether doing research, teaching or counseling students. I would like to thank Antti Hakkala, Jari Björne, Sami Hyrynsalmi, Jorma Boberg, Erkki Sutinen, Olli Nevalainen, Filip Ginter, Jukka Heikkonen and Martti Tolvanen, to name a few. I would also like to thank Maria Prusila, Sami Nuuttila, Marko Lahti, Heli Vilhonen, Pia Lehmussaari, Eija Nordlund and others for making the daily life in the department so much easier.

I like to thank my family for believing that this thesis will be completed someday (even when I myself had some doubts). My deepest gratitude and appreciation go to my mother, Eija-Liisa and father, Esa, my sister, Elina and her family, Mats, Anton and Amanda, and to my brother, Marko and his family, Aino, Heikki, Esko and Antti. I also like to thank Timo, Jussi, Markku and all my other friends, relatives and co-musicians for everything.

Finally, I want to thank Petra for making everything better every day. I love you.

LIST OF ORIGINAL PUBLICATIONS

P1. Kaila, E., Rajala, T., Laakso, M.-J. and Salakoski, T. 2009. *Effects, Experiences and Feedback from Studies of a Program Visualization Tool.* Informatics in Education. 8 (1), 17 – 34. Vilnius University.

P2. Kaila, E., Rajala, T., Laakso, M.J. and Salakoski, T., 2010. *Effects of Course-Long Use of a Program Visualization Tool.* In Proceedings of the Twelfth Australasian Conference on Computing Education (ACE 2010) - Volume 103, 97 – 106. Australian Computing Society, Inc.

P3. Kaila, E., Rajala, T., Laakso, M.J., Lindén, R., Kurvinen, E. and Salakoski, T. 2014. *Utilizing an Exercise-Based Learning Tool Effectively in Computer Science Courses*. Olympiads in Informatics, *8*, 93 – 109. Vilnius University.

P4. Kaila, E., Rajala, T., Laakso, M.J., Lindén, R., Kurvinen, E., Karavirta, V. and Salakoski, T. 2015. *Comparing Student Performance Between Traditional and Technologically Enhanced Programming Course.* In Proceedings of the Seventeenth Australasian Computing Education Conference (ACE 2015). CRPIT, 160, 147 – 154. Australian Computing Society, Inc.

P5. Kaila, E., Kurvinen, E., Lokkila, E. and Laakso, M.J. 2016. *Redesigning an Object-Oriented Programming Course*. ACM Transactions on Computing Education (TOCE), 16 (4), Article no. 18. ACM New York, NY, USA.

CONTRIBUTIONS OF THE AUTHOR

In papers **P1** and **P2** the data was collected and the setup of the studies designed by me, Teemu Rajala and Mikko-Jussi Laakso, and the analysis and reporting were done collaboratively by all authors.

In paper **P3** the setup was designed and the data collected and analyzed mainly by me, and the reporting was done collaboratively by all authors.

In paper **P4** the setup was designed and the data collected by me, Teemu Rajala and Mikko-Jussi Laakso, and the analysis and the reporting were done collaboratively by all authors.

In paper **P5** the setup was designed and the data collected by me and Mikko-Jussi Laakso, and the results were analyzed and reported with help from the other authors.

LIST OF CO-AUTHORED ORIGINAL PUBLICATIONS NOT INCLUDED IN THIS THESIS

- Laakso, M.-J., Kaila, E. and Rajala, T. 2018. *ViLLE – Collaborative Ecucation Tool: Designing and Adapting a Learning Environment for Collecting and Analyzing Educational Data.* Accepted for publication in Education and Information Technologies.

- Lokkila, E., Kaila, E., Lindén, R., Laakso, M.-J. and Sutinen, E. 2017. *Refactoring a CS0 Course for Engineering Students to Use Active Learning.* Interactive Technology and Smart Education 14 (3).

- Rajala, T., Kaila, E., Lindén, R., Kurvinen, E., Lokkila, E., Laakso, M.-J. & Salakoski, T. 2016. *Automatically Assessed Electronic Exams in Programming Courses.* In proceedings of the Eighteenth Australasian Computing Education Conference (ACE2016), Canberra, Australia.

- Kaila, E., Lindén, R., Rajala, T., Hellgren, N. & Laakso, M.-J. 2016. *Redesigning Methodology for Student Counseling in the First Year IT Education.* In proceedings of EDULEARN 2016 Conference, Barcelona, Spain.

- Veerasamy, A. K., D'Souza D., Lindén R., Kaila E., Laakso M.-J. & Salakoski T. 2016. *The Impact of Lecture Attendance on Exams for Novice Programming Students.* International Journal of Modern Education and Computer Science (IJMECS), 05/2016, Volume 8, Issue 5.

- Kurvinen, E., Hellgren, N., Kaila, E., Laakso, M.-J. & Salakoski, T. 2016. *Programming Misconceptions in an Introductory Level Programming Course Exam.* In proceedings of the 21th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2016), Arequipa, Peru.

- Wilman S., Lindén R., Kaila E., Rajala T., Laakso M.-J. and Salakoski T. 2015. *On Study Habits on an Introductory Course on Programming.* Computer Science Education, 25(3), 276-291.

- Kaila, E., Kurvinen, E., Lokkila, E., Laakso, M.-J. & Salakoski, T. 2015. *Enhancing Student-Teacher Communication in Programming Courses: a Case Study Using Weekly Surveys.* In proceedings of ICEE 2015 - International Conference on Engineering Education

- Leskelä, N., Kaila, E., Kurvinen, E., Rajala, T. & Laakso, M.-J. 2015. *Teaching Programming in Elementary School - a Case Study Comparing Play-Based Learning to Visual Programming.* In proceedings of EDULEARN15 - 7th International Conference on Education and New Learning Technologies.

- Holvitie, J., Haavisto, R., Rajala, T., Kaila, E., Laakso, M.-J. & Salakoski, T. 2012. *A Robot exercise for learning programming concepts.* In proceedings of ICEE 2012 - International Conference on Engineering Education, July 30th - August 3rd, 2012, Turku, Finland.

- Holvitie, J., Haavisto, R., Kaila, E., Rajala, T., Laakso, M.-J. & Salakoski, T. 2012. *Electronic exams with automatically assessed exercises.* Appeared in ICEE 2012 - International Conference on Engineering Education, July 30th - August 3rd, Turku, Finland.

- Kaila, E., Rajala, T., Laakso, M.-J. & Salakoski, T. 2011. *Important features in program visualization.* Appeared in ICEE : An International Conference on Engineering Education, 21-26 August 2011, Belfast, Northern Ireland, UK

- Rajala, T., Salakoski, T., Kaila, E. & Laakso, M-J. 2010. *How Does Collaboration Affect Algorithm Learning? A Case Study Using TRAKLA2 Algorithm Visualization Tool.* In Proceedings of 2010 International

Conference on Education Technology and Computer (ICETC 2010), Jun 2010.

- Kaila, E., Laakso, M.-J., Rajala, T. & Salakoski, T. 2009. *Evaluation of Learner Engagement in Program Visualization.* Appeared in 12th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2009), November 22 - 24, 2009, St. Thomas, US Virgin Islands.

- Laakso, M.-J., Rajala, T., Kaila, E. and Salakoski, T. 2008. *The Impact of Prior Experience in Using a Visualization Tool on Learning to Program.* Proceedings of CELDA 2008, Freiburg, Germany, 129-136.

- Rajala, T., Laakso, M.-J., Kaila, E. and Salakoski, T. 2008. *Effectiveness of Program Visualization: A Case Study with the ViLLE Tool.* Journal of Information Technology Education: Innovations in Practice, 7, 15-32.

CONTENTS

1 Introduction

Computers and programs can be found everywhere. In addition to our laptops, mobile phones and tablets, for example, cars, household items and various other things around us rely on microchips and programs running on them. Hence, writing programs is even more essential skill than it used to be during previous decades. Still, as shown in various studies (see for example McCracken et al. 2001, Lahtinen et al. 2005, Lister et al. 2004), programming is a very difficult skill to learn, and even more difficult skill to master. After introductory courses, various students typically still have difficulties in reading the program code and in writing simple programs. Moreover, the dropout rates in introductory programming courses are typically quite high. Without a solid base in basic programming concepts, mastering the advanced topics presented later in the curriculum becomes close to impossible.

It would be easy to blame the teachers, but there is more to the issue. As programming is considered an essential skill (not only in computer science, but in other topics as well), introductory programming courses, especially at university level, are usually crowded with students. This means, that the course staff has very little time to spend on individually instructing the students and on assessing the assignments. As programming is generally considered a skill where active learning is essential (see for example Jenkins 2002), the lack of time leads to problems in learning. In general, you cannot learn to program unless you write a plenty of programs yourself, and writing programs without proper feedback and guidance might be an impossible task for beginners.

Educational technology provides potential solution to the problem. Though technology can be useful in teaching any topics, it can provide particular benefits when utilized in programming or computer science education. Assessing programming assignments and other tasks can be automatized, and immediate feedback (Laakso 2010) can be provided on the correctness (and to some extent, the quality) of the solutions. This enables practicing the topics anywhere, anytime, and an unlimited number of iterations, if the tasks are cleverly designed. Educational technology can potentially provide other benefits as well, such as easy (and transparent) recording of assignment scores and other tasks, gamification (Deterding et al. 2011) and continuous assessment where completing tasks throughout the course becomes meaningful.

Like any tools, educational tools are the most (or only) useful if utilized correctly. There are several questions to consider when adapting the tools into the courses: what tools would best suit the requirements, which features should be utilized, how to reduce cognitive load (Sweller 1994) and how to motivate and reward the students, to name a few. In other words, how to get the best benefits out of the educational technology and still keep the adaptation feasible in terms of the costs and the workload. There are several educational tools designed especially for programming and computer science education (see Sections 2.2 and 2.3 for some examples). However, it is quite rare to see studies where the adaptation of the tools is studied extensively after the introductory results.

In this thesis, I present the scientific background, the design and the development of an educational tool called ViLLE, and discuss the effectiveness of the tool by observing the results of the controlled tests and the adaptation cases in different scenarios utilizing different methods. Hence, the focus is both on the technology used, but even more on the ways the technology is utilized. Based on the results, I form a model about utilizing educational technology in programming and computer science courses. As a goal, I expect that such model can be used to increase the quality of the education in the field of information technology.

1.1 Research Questions

The goal of this thesis is to find out how educational technology can be adapted successfully into programming and computer science courses. For this goal, studies conducted with an educational tool called ViLLE during recent years are observed and the results evaluated. The thesis attempts to answer the following research questions:

RQ1. What kind of features are useful when adapting educational technology into computer science and programming courses?

RQ2. If such features are found, can they be combined into a model of a methodology for utilizing educational technology?

RQ3. What kind of requirements and limitations there are to consider when such model is adapted in computer science or programming courses?

Hence, the goal of the first research question is to observe the individual features that should be considered when an educational tool is utilized. The second question is about combining the features that are proven to be successful into a complete model that can be utilized by anyone interested in utilizing educational technology in programming or computer science courses. The third question is about the adaptation of such model, and the requirements and possible limitations concerning the technology, course contents and people that need to be addressed when the adaptation is done.

1.2 Methodology

The research reported in this thesis and the development of ViLLE is based on the design research methodology. The framework used is loosely based on one described by March & Smith (1995), but excluding the natural science research components. Hence, it consists of two research outputs, Constructs and Models, and two research activities, Build and Evaluate. The framework is displayed in Figure 1.

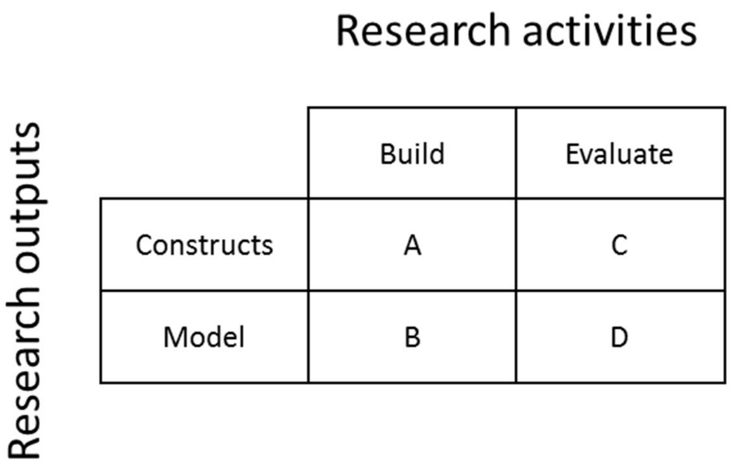


Figure 1 The research methodology utilized. Based on work of March and Smith (1995).

There are a total of four components in the model, described below:

- *Constructs* form the vocabulary of the domain. In this thesis, the vocabulary contains features such as assessment, round, assignment and tutorial. It is essential to understand their meaning *in this context* to understand the problem and the solution.
- *Model* expresses the joined constructs and their relationship. According to March & Smith (1995), the model is a description of "how things are". In addition to model described in Section 5, the cases described in Section 4 are sub models according to this definition.
- *Build* is an action of preparing an artifact. The artifact can be an individual construct or a whole model.
- The artifacts need to be *evaluated* based on how well they work. The evaluation requires metrics that measure the effectiveness of the artifacts. In this thesis, student performance (measured with pass rate and grade average) is the main target of evaluation.

The matrix itself consists of four cells, labeled A to D in Figure 1. The cells are described below.

A. In this stage, a construct is defined or build. Defining and implementing individual features later utilized in the model belong to this category.
B. After the features are defined, they can be joined into a model. The final outcome of this thesis is the proposed model, but sub models are defined and studied earlier.
C. Individual constructs are evaluated when possible. However, as described in Section 5.1, it is not feasible or ethically sustainable to test for the effects of all individual features in the model.
D. The model (or sub model) needs to be evaluated to find out if it is effective in the task it was designed for. This thesis is concluded with the evaluation of the complete model, but the sub models are evaluated in papers P1, P2 and P3, and in Section 4.

It is to be noted, that the design and implementation process of this research has been cyclic: new features have been designed, implemented, tested and reworked gradually to complete the whole model. Detailed description of the development process of ViLLE and the studies conducted is provided in Sections 3 and 4.

1.3 The Structure of this Thesis

This thesis is structured as follows. First, in the second section, the related literature by the research community is reviewed. The section starts with a review on programming learning and its difficulty, and continues with a discussion about program visualization, its potential effects and an overview about prominent program visualization tools. After that, other educational tools are discussed followed by a review about teaching methodologies and their effectiveness in the course redesign process. In the third section, the educational tool utilized in the studies of this thesis, ViLLE, is presented. As there are two, vastly different versions of the tool, they are presented separately along with a scientific background on which they were built upon, and an overview of the features.

In the fourth section, the studies about utilizing ViLLE in programming and computer science education are presented and discussed. The section starts with an overview of controlled tests performed before the tool was adapted into course-long usage. After that, five different use cases from courses with different content and student profiles are presented, along with the most important results gathered from the courses. The section is concluded with a summary of all presented results. In the fifth section, a model for adapting educational technology in computer science and programming courses is presented, based on the results discussed in the previous section. In addition to the model and the intervention features in it, adaptation and limitations of the model are discussed.

The thesis is based on five research papers, where new features to be included into the model are gradually presented. In paper P1, controlled tests and other experiments about using a visualization tool are presented. In paper P2, an early experiment about adapting the same tool into a computer science course is studied. In paper P3, we present the new version of the tool, and provide results of a study where the tool was adapted into a high school programming course and into a bioinformatics programming course, respectively. In papers P4 and P5 all features of the model to be formed are utilized when the tool is adapted into university-level programming courses.

2 About Educational Technology and Programming: Related Studies and Tools

In this section, work related to concepts of this thesis by other authors is reviewed. The section is divided into four subsections. First, the mechanisms and fundamental problems in programming learning are discussed, with focus on three questions: how programming is learned, why is programming difficult to learn, and how the difficulties are observable. In the second subsection, program visualization techniques and tools are observed. In the third subsection, development and evaluation of other types of tools representing different areas of educational technologies are presented, and finally theories behind effective education and effective educational technology are discussed. In that final subsection, a few examples of course redesign methodologies are also presented. Some theoretical background directly related into design of ViLLE is presented in Section 3.

2.1 Programming Learning: Why and how is it Difficult?

Programming is without a doubt one of the most important skills in the computer science and computing engineering curriculums. The skills needed to learn programming can be roughly divided into two subcategories: first, there are technical skills, such as the syntax of the programming language and the usage of the control structures (such as loops, functions or conditional statements). Second, to utilize programming as an effective tool, it is essential to learn algorithmic problem solving skills. DuBoulay (1989) divides these two categories further into five overlapping domains that are essential to master to learn to program: orientation, notional machine, notation, structures and pragmatics. As Havenga et al. (2013) note, students usually focus on writing programs instead of programming, meaning that they concentrate on the mechanical task instead of utilizing programming as a problem solving mechanism. Davies (1993) makes a similar distinction between programming knowledge and programming strategies.

In their systematic literature review about learning and teaching programming, Robins et al. (2003) list novice programmers' typical issues: they are limited to superficial knowledge, lack adequate mental models, use overly simplified problem solving strategies, and spend too little time in planning, testing and debugging programs. Novices also have poor code tracing skills and lack knowledge about the sequential nature of programs. Schemas (see for example Mayer 1981) are also something that distinguish experts from novices. Schemas are mental models representing solutions to algorithmic problems and sub problems in programming. Since experts have more developed schemas and better ability to apply them, problem solving process is much easier for them. As a concrete example, experts typically have highly developed schemas for sorting arrays or lists, and know which of them to apply in different cases, while novices may spend a lot of time solving the same problem and still come up with less suitable or ineffective solution.

There are various studies underlining the difficulties in programming learning. In their multi-national, multi-institutional study, McCracken et al. (2001) report that the programming skills of students are typically quite poor after the introductory courses. In fact, out of 110 points in the test used for the study, the students only gained an average of 23 points after taking the introductory courses. Although the report is already 15 years old, the results are likely still similar in many cases. Lahtinen et al. (2005) surveyed more than 500 students about their perceptions of difficulties in programming. Their results indicate, that the students find designing and building programs as well as debugging the most difficult issues, and error handling, recursion and references the most difficult concepts. The students also preferred practical sessions and coursework as the best learning methods, and example programs as the most useful learning materials.

Among several others, Bergin et al. (2005) underline the importance of motivation in learning to program. However, they also distinguish between internal and external motivation: the internal motivation derives from a desire to learn the concepts, while the external motivation comes from desire for a reward or to avoid a punishment. Gomez & Mendes (2014) state, that students typically lack intrinsic motivation in the programming courses due to their difficulty. One of the solutions they suggest, based on teacher interviews, is to attribute classifications based on the difficulty of tasks, for example by offering bonus points on more demanding exercises or assignments.

Bosse & Gerosa (2017) present a systematic literature review about programming difficulties, and present a "model, which may assist students in sharpening their focus, and teachers in preparing their lessons and teaching material, as well as researchers in employing methods and tools to support learning." Visual programming tools (see Section 2.2) may been seen as a potential aid, but as Tijani et al. (2017) state, the skills acquired with visual programming environment may have no correlation to procedural programming, meaning that such tools, although entertaining, still cannot provide any real help to the problem.

2.2 Program Visualization

Several methods and tools to assist programming learning have been developed and used. Programming skills can be roughly divided into two major areas: code writing, and code reading (typically referred as code tracing) skills. There are some controversies in the research community about whether the two are connected: for example, Lopez et al. (2008) found a strong correlation between students' code tracing and writing skills, but there are opposite findings as well: Winslow (1996) for example states, that according to studies, there is *"very little correspondence between the ability to write a program and the ability to read one."* Nevertheless, both are essential skills to master. Tracing skills are particularly important in adopting example programs and in debugging the program code.

Program visualization can potentially be used to enhance code reading skills. Nowadays, the definition of program visualization seems quite unambiguous for anyone in the research field. However, according to Hyrskykari (1993), terms visual programming and program visualization were used to describe anything that had something to do with computer graphics and programs, until Myers (1986) defined visual programming as a *"system that allows the user to specify a program in a two or more dimensional manner"* and program visualization as specifying program as text, but using graphics to illustrate some aspects of the program or its execution.

When discussing program visualization, it is nowadays usually common to refer to tools that illustrate the execution of programs with graphical or textual components. The goal of the program visualization is to illustrate *what* the program does, *why* it does it, *how* it works and *what* happens in the process (see for example Wiggins 1998). Typically the tools aim to visualize for the sequence and the order of statements, execution control, states of variables and objects, evaluation of expressions and the execution of procedures and functions. For these purposes, program visualization tools utilize for example code highlighting, call stacks, animated expression evaluation and verbal explanations about the executed code lines (see for example Rajala et al. 2007 or Moreno et al. 2004). Typically visualization tools also allow students to execute the programs step-by-step or in selectable speed.

There are several program visualizations tools developed over the years. Jeliot 3 (Moreno et al. 2004) is one of the most prominent ones. It illustrates the execution of example programs with various graphical and textual features: for example, the currently executed code line is highlighted, the execution of expressions is displayed animated in the "theatre area", methods are displayed in their own frames with their local variables, and program output is visible in the console frame. Čisar et al. (2011) studied the utilization of Jeliot 3 in two higher educate institutions with around 400 students in Serbia. According to their findings, the groups utilizing Jeliot 3 achieved better results than the students in the control group. However, the detailed (and hence often quite slow paced) evaluation animations in Jeliot 3 are not always favored by students (Kannusmäki et al. 2004).

VIP (a Visual Interpreter, Virtanen et al. 2005) is a program visualization tool aimed for learning C++. Utilizing a restricted subset of the programming language, VIP provides users some typical visualization features, such as code line highlighting, variable values, output console and an evaluation area. Isohanni & Knobelsdorf (2011) studied the long-term engagement of students with VIP and isolated four different phases describing the engagement. The authors state that high commitment with a visualization tool is crucial, because *"it distinguishes the use of a visualization tool from the use of a normal program development environment".* Examples of other visualization tools are jGRASP (Cross et al. 2004), which is a full development environment with possibility to visualize the programs, and Jype (Helminen & Malmi 2010), which provides quite comprehensive feature set for visualizing programs in Python, including for example the visualization of algorithms and data structures.

2.3 Other Educational Tools

In addition to visualization tools, there are several other educational tools developed to aid teachers and students. Dalsgaard (2006) makes a distinction between *integrated and separated tools*. Separated tools are individual tools, for example chat rooms, file sharing tools, shared whiteboards, e-portfolios and wiki platforms. Integrated tools combine several tools under a single platform. The platforms, called learning management systems (usually abbreviated as LMS), are a typical and very widespread use of educational technology. They are often used for administrative features and communications: sharing materials, receiving student submissions and for private and shared communications, but also for assignments, peer-reviews and assessment.

Very common examples of widely used LMS's are for example Moodle (Dougiamas & Taylor 2003) and Blackboard (Bradford et al. 2007). Coates et al. (2005) suggest reasons for universities' enthusiasm in adapting learning management systems: they provide means of increasing the efficiency of teaching and are associated with the promise of enriched student learning. Moreover, students' expectations towards learning technology are increasing, and there is a lot of competition among universities, which forces them to seek improvement on the quality of education. Finally, LMS's are proposed as a solution to higher demand for greater access to higher education, and as such, are part of a culture shift taking place in education.

In addition to LMS, term LCMS (learning content management system) is sometimes used. Although the terms are often used as interchangeable, there can be a semantic difference. According to Oakes (2002) LCMS's are focused on developing and delivering content in the form of *learning objects,* while LMS's are used for managing learners and activities. The concept of learning objects is used quite widely when discussing educational technology (although as Polsani (2006) points out, the term is often used without any critique which reduces its meaning). Still, learning objects can be defined as re-usable objects, based on one learning goal and being as cohesive and de-coupled as possible (Boyle 2003). Although most authors define learning objects as digital (see for example McGreal 2004), some also include objects that are not associated with or do not utilize educational technology (in fact, the IEEE standardization also includes non-digital objects, see Boyle 2003).

As stated before, LMS's can be used for delivering different kinds of content and learning objects to students. Wiki (an example of a typical learning content in almost any LMS) for example can enable collaborative creation and learning and support distant learning (see for example Wheeler et al. 2008 or Engstrom & Jewett 2005). Another common feature is an electronic (learning) portfolio, a collection of learning artifacts that offers students a possibility for self-reflection (Barret 2007). Electronic portfolios are typically used for assessing students as well. Computer-assisted grading rubrics can improve the quality of feedback and enable easier personalized comments about students' work (Auvinen et al. 2009, Anglin et al. 2008). Other tools that can either assist teachers in their daily tasks, or enable whole new approach to teaching are for example lecture polling tools (see for example Menon et al. 2004) and peer-review (see for example Gehringer 2001).

In addition, there are other kinds of tools designed especially for programming education. First, there are the tools for automatic assessment of program code. Examples of such tools are Web-CAT (Edwards & Perez-Quinones, 2008), Scheme-robo (Saikkonen et al. 2001) and BOSS (Joy et al. 2005). Features offered by such tools include for example the support for different programming languages, possibility for resubmissions, defining automated tests for submitted code, sandboxing for executing the submitted code securely, and a possibility for manual assessment (Ihantola et al. 2010). Other tools designed for programming education include for example the tools and IDEs for supporting programming process, such as BlueJ (Kölling et al. 2003), tools for program simulation (which can be seen as "reverse" visualization), such as UUHistle (Sorva & Sirkiä 2010) and to some extent, graphical programming tools such as Scratch (Resnick et al. 2009), Greenfoot (Henriksen & Gölling 2004) and Alice (Cooper et al. 2000).

2.4 Methods and Methodologies for Programming Education

Appropriate tools are still just a first step towards more effective and motivating computer science and programming courses. The study methods utilized are even more important. Hence, various studies about utilizing different methods have been conducted. Student collaboration, especially in the form of pair programming, is one example. For example, Lee et al. (2013) and Simon et al. (2010) found a positive correlation between pair-programming and student performance. Moreover, Porter et al. (2013) conclude, that pair-programming, peer instruction and media computation provide better results in introductory programming courses because they can make programming courses "easier" by making them less asocial, boring and competitive. Similarly, Nagappan et al. (2003) found out that students who utilized pair-programming were *"more self-sufficient, generally perform better on* *projects and exams and were more likely to complete the class with grade C or higher than their counterparts.".*

Flipped learning is an example of a methodology that is used quite often nowadays to try to enhance the classroom experience. In flipped learning, lectures are usually served as short videos, and the time in classroom is spent in doing exercises (see for example Sams & Bergmann 2013). However, the results from the flipped classroom experiments are somewhat mixed. For example, Amresh et al. (2013) report an experiment of utilizing flipped classroom in CS1 course. They found out, that the group utilizing the flipped methodology gained higher scores in the assignments, but on the other hand some students found the methodology *"overwhelming and intimidating at times".* Similarly, Bruff et al. (2013) report a study where a MOOC (massive open online course) was integrated into a machine learning course. The authors found out, that while some of the feedback from students was positive, there were also concerns about combining online materials with in-class components.

Although MOOCs do not really fit into the scope of this thesis (as they rely almost solely on distant and independent learning) they should still be discussed, since they rely heavily on educational technology and as such, provide excellent examples on the adaptation of such technology and tools. MOOCs have become extremely popular in recent years. According to Pappano (2012), Coursera – a popular MOOC platform – partners with many elite universities to offer distant learning to anyone, regardless of their base education or socioeconomic status. Probably at least partly because of the low threshold to enter the courses, the pass rates of students in MOOC courses are typically very low compared to traditional courses. For example, according to a survey conducted among professors utilizing MOOCs (Kolowich 2013) the pass percentage is only 7.5% (although the author states that the results of the survey are not "scientific" due to respondents being most likely *"the most enthusiastic of the MOOC professors"*).

Nevertheless, there are various educational tools and methodologies developed for and used in MOOC courses about programming or CS. For example, Vihavainen et al. (2012) report a development of introductory programming MOOC where they utilized a pedagogical method called Extreme Apprenticeship, which, according to the authors, is extremely effective in programming education. The MOOC in question is also used as an entrance exam into the formal computer science degree, which further extends the possibilities of MOOC courses. Warren et al. (2014) underline the importance of human interaction, even when taking an online course. They report the development and utilization of a programing environment CodeSkulptor (see also Tang et al. 2014) enabling code sharing with other students in a MOOC course they found to be successful. Other examples of technology utilized in programming MOOCs are reported for example by Nguyen et al. (2014), Miller et al. (2014), Pieterse (2013) and Yin et al. (2015).

There are examples of successful adaptation of new methodologies for teaching programming in traditional classroom setup. For example, Boyle et al. (2003) report an experiment where *blended learning* was utilized to try to improve student success in an introductory programming course. Changes made included for example an extensive usage of online environment. According to the authors, the changes resulted in better pass rates and the study indicated *"widespread use of the new online features"*. Similarly, Leutenegger & Edgington (2007) report an experiment where game programming was utilized in an introductory programming course to make the course more motivating for students. According to the authors, the approach used improved the understanding of all basic topics in the course.

Vihavainen et al. (2014) conducted a systematic review of different methodologies and on their success in teaching introductory programming. The authors isolated several intervention methods (including for example collaboration, content change, game-theme and peer-support) and evaluate the success based on improvements in pass rates and changes in the size of the student population. According to the results of analyzing 32 carefully selected articles, the most successful methods for improving the pass rates were the utilization of media computing (or relatable content, in other words), group work and the introduction of CS0 course before the first programming course. Hybrid approaches (approaches utilizing more than one of intervention methods studied) provided quite good results, but there was also a lot of variation between the results. This shows, that the methodology must be carefully designed and studied when adapting it.

3 Designing and implementing ViLLE

In this chapter, an educational tool utilized in this thesis is described. ViLLE (originally Visual Learning Tool, later Collaborative Education Tool) has been developed at the Department of Information Technology, University of Turku since early 2004. The lifespan of the tool can be divided into two separate versions. The first version – a program visualization tool – was developed starting from early 2004, first as a student project by me, Teemu Rajala and Ilkka Sillanpää, and later in the department by Teemu, me and Mikko-Jussi Laakso. The version is studied in papers P1, P2 and P3 in this thesis. The second version – a collaborative learning tool – was started in late 2008. The version was again designed by Teemu, me and Mikko. Later the development has been done by dedicated *ViLLE Team* research group. The latter version is studied in papers P3, P4 and P5 in this thesis.

Since the versions were developed one after another, the design process and scientific background are discussed separately in this section. Both versions were based on the existing research by the community and ourselves, and were designed to utilize the best practices to enhance learning as significantly as possible. Hence, for both versions, a brief history and a scientific background for design are presented, along with an overview of the most important features. In this thesis, the versions are labeled ViLLE 1 and ViLLE 2 when the distinction needs to be clarified.

3.1 ViLLE 1 – Visual Learning Tool

The first version of ViLLE was primarily a program visualization tool. The development started as a student project in early 2004, and the first version was published in 2005. The first version was developed until 2009, when the next generation was released for the first time. ViLLE 1 was developed as a standalone Java application, but was later extended with support for an external server side application to enable recording of scores and submission counts. From the very beginning, ViLLE supported the visualization of programs written by the user (alongside the examples provided in the package), and had a support for various programming languages. Latter versions added some features to encourage active learning, including for example multiple choice questions.

3.1.1 Background

Program visualization is a method of illustrating the execution of computer program or algorithm step by step. Typically in visualization, the program can be executed continuously or in steps controlled by the user. While the program is executed, the visualization tool depicts the changes in the state of the program graphically or textually. Hence, typical program visualization tools resemble the debugging features found in most advanced programming IDEs, with two differences: the visualizations are usually simplified in visualization systems, and it is common to add features to activate students, such as questions or other tasks to complete during the execution of the example program.

Some early studies (see for example Hyrskykari 1993) already suggested that program visualization can be an efficient tool in concretizing abstract concepts, such as variables or method calls. These concrete models can then be combined with existing knowledge to create schemas for programming. According to Naps et al. (2002), the educators do believe that visualization tools can aid learning, but still there is a "significant disconnect between the intuitive belief […] and the willingness to and ability of instructors to deploy visualizations in their classrooms". The authors state, that for teachers, the most significant reasons for not adapting visualization tools are the time required to search for good examples, the time it takes to learn to use the tool and the time it takes to develop pedagogically valid visualizations.

Naps et al. (2003) state that visualization systems can be roughly divided into two extremes: the systems that are meant for single purpose and single platform, are undocumented and do not engage students actively into learning, and the systems built for comprehensive learning experience with exercises that encourage active learning and that can be run in multiple platforms. The authors argue, that the systems that fall in the first category are only usable by their designers for a specific course and for a specific purpose, and could not be seen as generally usable. The support for active learning can be measured by *engagement taxonomy*, proposed by Naps et al. (2002). In this taxonomy, the relationship between the user and the visualization is divided into six levels according to the user's engagement into visualization. The levels are

1. No viewing – there is no visualization tool at use
2. Viewing – the visualization is viewed passively with no further engagement
3. Answering – the user is shown questions about visualization during the viewing
4. Changing – the user can change the algorithm or program visualized
5. Constructing – the user builds own visualizations
6. Presenting – the user presents his or her visualizations for peers to review

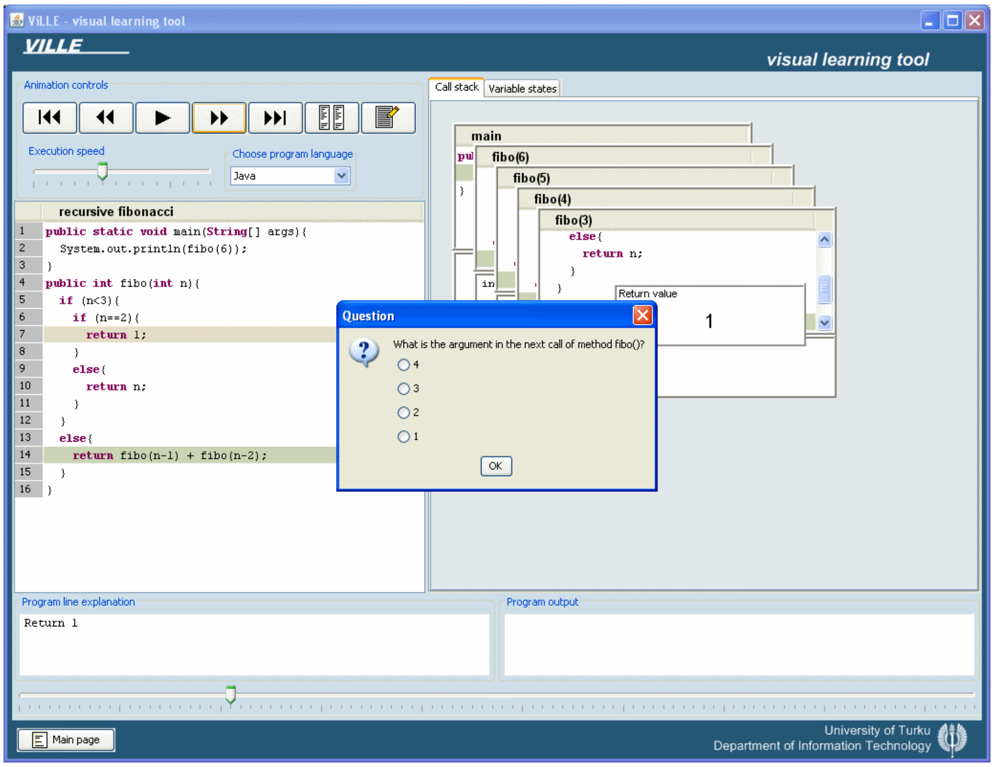
Important hypothesis about the taxonomy, proposed by Naps et al. (2003) is that visualizations are useful for learning only if used in level three or higher. This means that visualization tools should provide some form of active engagement for them to pedagogically useful.

One of the fundamental questions in programming education is the choice of the first programming language. Several studies have been conducted on the topic. For example, Grandell et al. (2006) argue, that Python provides many features that make it suitable as a first programming language in high school level, including easy-to-learn syntax and immediate feedback. In the extended study, Mannila et al. (2006) state, that if a language with simpler syntax (such as Python) is taught as the first language, the transition to more advanced language (such as Java) is smoother, as the students made less syntax or logic errors. However, the transition from a language to another should be made as effortless as possible, since the industry often requires the knowledge of advanced languages as well.

3.1.2 Features

Based on the requirements observed in the previous section, the design of ViLLE 1 was decided to be based on two major principles: first, the tool should activate learner in the upper levels of engagement, and second, it should support a variety of programming languages, making the transition between languages as easy as possible. The core feature of the first ViLLE was the visualization of programs: when a program is executed, the execution is illustrated with various graphical and textual components. The execution can be accompanied with multiple choice questions as well as graphical array questions. Moreover, the visualized programs can be displayed using any of the supported programming languages (including for example Python, Java, C, PHP and pseudo language). The features are described in more detail below, divided according to the design principles.

First, the visualization was designed to be simple, fluid and still detailed enough to provide adequate information about the program visualized. The program code visualized is displayed in the left frame, with the current and the previously executed code lines highlighted. The controls above the code frame allow the student to execute the program one code line at a time, or continuously with selected speed. The call stack on the right hand side of the window displays all subprogram calls in their own frames, with information about all variables (including parameters) created in those subprograms. Alternatively, the user can display a comprehensive view of all variables in the right frame. The program output is displayed below the call stack and a verbal explanation about the executed code line (such as "Initialize variable 'number' with value from expression 2 + 3 == 5") below the program code. The visualization view of ViLLE 1 is displayed in Figure 2.



recursive fibonacci

```
public static void main(String[] args){
  System.out.println(fibo(6));
}
public int fibo(int n){
  if (n<3){
    if (n==2){
      return 1;
    }
    else{
      return n;
    }
  }
  else{
    return fibo(n-1) + fibo(n-2);
  }
}
```

```
else{
  return n;
}
}
```

Return value: 1

Question

What is the argument in the next call of method fibo()?

- 4
- 3
- 2
- 1

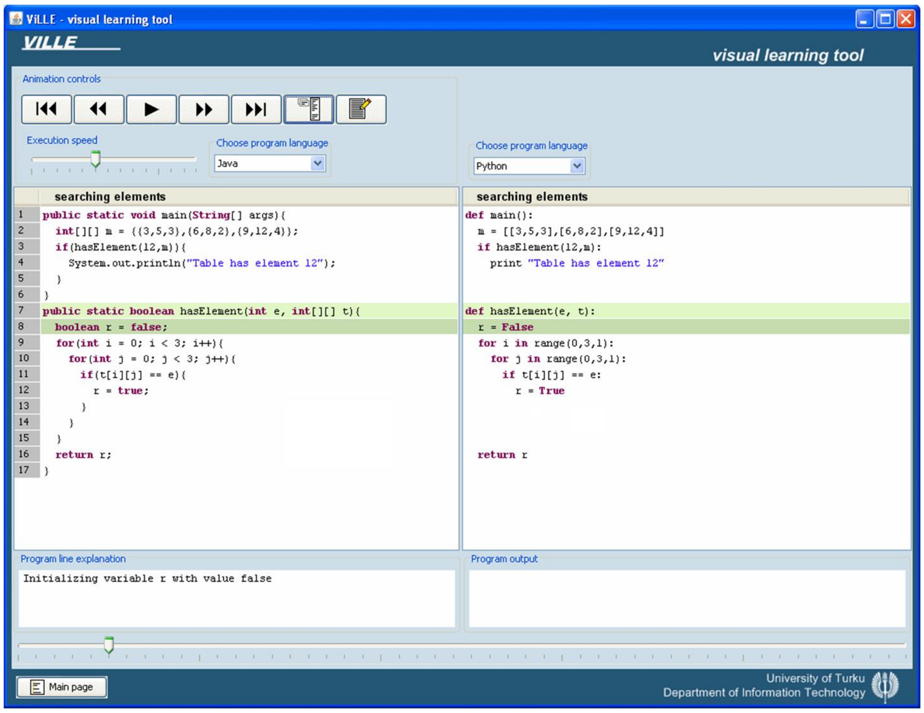
OK

Program line explanation: Return 1

Figure 2 Example program calculating the first six values in the Fibonacci series visualized.

Multiple choice questions and visual array questions can be included into desired points of the program execution. ViLLE 1 does not automatically generate questions. After experimenting with such feature, we found out that automatically generated questions do not provide enough pedagogical value to the visualization (as they tend to be mechanical and self-repeating), although they do make the generation of exercises easier for teachers. Still, in this case less is more: questions carefully designed and inserted into adequate points of the execution can engage the students deeper into visualization than repeated automated questions. The questions inserted into visualizations can be roughly divided into two categories: 1) questions about the program execution (such as "which is the next line executed" or "what will be the value of variable a after this code line is executed") and 2) general questions about programming (such as "What operator is used to calculate the remainder of division?"). Typically questions from both categories were contained in the example programs to try to come up with pedagogically valid and holistic tasks.

ViLLE 1 supports a variety of programming languages. All examples provided can be displayed in any programming language supported. Moreover, the tool has a parallel view (Figure 3), where the program execution can be displayed in two different programming languages at the same time.



Choose program language: Java

Choose program language: Python

searching elements

```
public static void main(String[] args){
  int[][] m = {{3,5,3},{6,8,2},{9,12,4}};
  if(hasElement(12,m)){
    System.out.println("Table has element 12");
  }
}
public static boolean hasElement(int e, int[][] t){
  boolean r = false;
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      if(t[i][j] == e){
        r = true;
      }
    }
  }
  return r;
}
```

searching elements

```
def main():
  m = [[3,5,3],[6,8,2],[9,12,4]]
  if hasElement(12,m):
    print "Table has element 12"


def hasElement(e, t):
  r = False
  for i in range(0,3,1):
    for j in range(0,3,1):
      if t[i][j] == e:
        r = True



  return r
```

Program line explanation

```
Initializing variable r with value false
```

Figure 3 Example program visualized in Java and Python at the same time.

The view was designed to help the transfer from one programming language to another. The feature is also meant for illustrating the similarities between imperative programming languages.

Though ViLLE 1 came pre-packaged with various example programs, it also allows teachers (and to some extent students) to create their own visualizations. The example programs are written in Java, and ViLLE automatically translates them into all other supported languages. It is also possible to modify the automatic translations, if necessary. When the translations are done, questions can be attached into the visualization by anchoring them to the execution steps.

The last version of ViLLE 1 provided some additional features. In addition to program visualization exercises, a new exercise type was added. In the *code shuffling exercise* the students need to order the shuffled code lines into correct order according to given task. The exercise, also known as Parsons puzzle (Parsons & Haden 2006), is intended for teaching basic syntactical and logical constructs of programming. Including code shuffling exercises among visualization exercises also provides some variation for students. The very final version of ViLLE 1 also introduced a simple coding exercise, where the students needed to complete the given task by writing a program or a missing part of it (however, the feature was never studied or used extensively). Another addition to the last version was the information about roles of variables (see Kuittinen & Sajaniemi 2004). Each variable in the program is assigned a role (such as Stepper, Follower, Gatherer or Temporary) to illustrate its role in the execution. Sajaniemi & Kuittinen (2005) had promising results about introducing the roles of variable to students. Unfortunately, in ViLLE the effect of adding the role information into visualizations was neither complete nor properly studied due to concentrating into other research topics. Hence, the role information was discarded when ViLLE 2 was introduced.

3.2 ViLLE 2 – Collaborative Education Tool

ViLLE 1 was deemed unfit for lot of areas in education by 2008 by the development team, and hence a design and implementation of a new version was started by Teemu Rajala, me and Mikko-Jussi Laakso. Though the original version worked well enough as a program visualization tool, it lacked a lot of features found necessary in the more comprehensive education tool. Most importantly, there was no support for teacher or student collaboration, the exercise selection was limited for computer science education (and more specifically, for programming education), and no support for other educational activities (such as assessing, lectures or assignments) was included. Moreover, it was executed as a Java applet, a technology that was found as obsolete and unsecure by many, and which required an external plugin to browser to be executed.

The second major version of ViLLE was designed with this in mind. The implementation was done on Vaadin framework (Grönroos 2011). Vaadin is a web-based server side framework, based on Google Web Toolkit (Johnson et al. 2007), enabling the client and server side programming in Java. The applications can be executed in all modern browsers without additional plugin requirements. Though ViLLE 1 could utilize an external server for delivering the exercises, the services offered by the server were inadequate for features designed for the next generation of ViLLE. Hence, a dedicated server environment was designed and implemented for the new version. The second version of ViLLE was first released in 2010, and the development has continued ever since.

3.2.1 Background

Student collaboration, especially in the form of pair programming, has been studied quite extensively. For example, in the quite comprehensive literature review about pair programming in CS education (Salleh et al. 2011) the authors found out, that pair programming can improve student satisfaction and students' grades on assignments. Moreover, they found a positive correlation between pair programming and on the time spent in programming. This is actually logical, as the time spent in assignments is likely to increase when the students discuss the assignment when working on it. In fact, our research group (see Rajala et al. 2011) measured the time the students spent on discussing the topic during the collaboration and found out, that over 90 % of the discussion is about the topic. Laakso et al. (2009) state similarly, that based on the analysis of collaboration and discussion during a session spent with algorithm visualization, it can be said that the students are "truly learning collaboratively".

Another important aspect of collaboration is the joint work of teachers. According to Briscoe et al. (1997) teacher collaboration can facilitate change as it “provides opportunities for teachers to learn both content and pedagogical knowledge from one another”. Moreover, the authors claim that collaboration can encourage teachers to implement new ideas and promote individual change in [science] teaching. Meiring et al. (2010) studied the relationship between teacher learning and collaboration in innovative teams, and found out, that collaboration can be mostly characterized as sharing. As a more practical result, they suggest that collaboration in professional teams (described as innovative, voluntary and, interestingly, temporary) could be a good direction for the professional development of teachers. The temporary team forming can be hence emphasized by creating a system where instead of having pre-defined teams, all teachers can collaborate in whatever forms they find the most useful.

Even before the redesigned version, active learning was already an important part of ViLLE. A definition by Freeman et al. (2014) illustrates the principle perfectly: active learning can be described as any activity wherein the student actively partakes in the process of forming a solution to a given problem. Active learning using educational technology, without connection to specific time or place, can be enabled with automatic assessment and immediate feedback (see for example the work of Laakso 2010). A clever utilization of aforementioned properties allows students to answer the exercises anywhere, anytime without compromising the quality or motivation of teaching. If exercises are initialized randomly (see Malmi et al. 2004 for an example), the reusability of them as well as a meaningful iteration of the same exercises increases drastically.

Gamification provides a lot of potential into educational technology. The definition of the term varies a bit, as stated by Deterding et al. (2011). However, based on the investigation on the historic use of the term, the authors propose a definition of gamification as “the use of game design elements in non-game contexts”. In educational technology, gamification has two obvious applications. First, game-like exercises can be included to enhance motivation and make solving the exercises more entertaining, which can result into better learning performance (see for example Giannakos 2013). In fact, in their literature survey of 40 studies about the effectiveness of educational games, Backlund et al. (2013) conclude that 29 of the studies reported positive effects. Second application of gamification is the utilization of virtual achievements, trophies and awards (Huang et al. 2013).

An educational tool enabling comprehensive usage throughout the course would also enable continuous assessment. Again, the concept varies depending on the definer: Nitko (1995) for example discusses the different views towards the definition, and continues to present a conceptual framework for organizing continuous assessment. Interestingly, in a survey conducted by Hernandez (2012), teachers reported providing feedback on student progress as the most important purpose of continuous assessment. Educational technology can potentially provide real-time data about students' progress in course to both, teacher and the students. There are however some requirements for this, including for example the comprehensive use of the tool in all aspects of the course, and the ability to provide meaningful feedback on the progress.

3.2.2 Features

New ViLLE (referred to as VILLE 2 when distinguished from the old version) is divided into two main views: the teacher view and the student view. Teachers can also access the student view to test their courses in the role of student, and to support cases where user has both roles (for example, assistant teachers can also be students in other courses). Both views can be accessed via web-based interface using any modern internet browser with no additional plugins required. ViLLE 2 uses a client-server architecture, where most of the data processing is done on the server side, and the communication via client, displayed as HTML and JavaScript in the browser. The data is stored into MySQL database or into flat files, depending on its type.

The model used in ViLLE relies on the concept of virtual courses. The courses are divided into rounds, which are primarily collections of assignments. Each round has its own properties, including for example the opening and closing times as well as score limits for virtual trophies. ViLLE currently supports four different types of rounds: normal rounds, exam rounds, conditional rounds and tutorials. The major distinction between normal rounds and exam rounds is the ability to limit the access and feedback in the latter. Tutorial rounds are combinations of materials (including for example text, images and videos) with exercises embedded into them. The structure of ViLLE's learning resources is displayed in Figure 4.

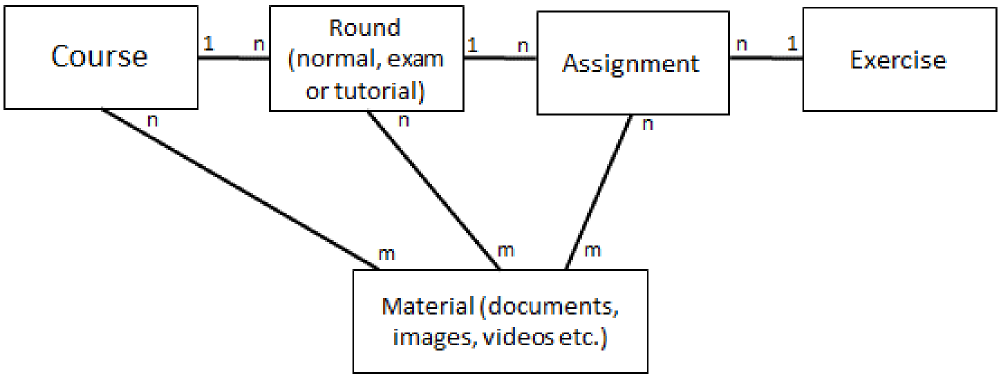

Course
Round (normal, exam or tutorial)
Assignment
Exercise
Material (documents, images, videos etc.)
1
n
1
n
n
1
n
n
n
m
m
m

Figure 4 The structure of ViLLE's learning resources. Courses consist of rounds, which consist of assignments. External materials can be attached to courses, rounds or assignments.

Currently there are more than 5,000 teachers and over 100,000 students using ViLLE. The platform is used in 15 different countries around the world.

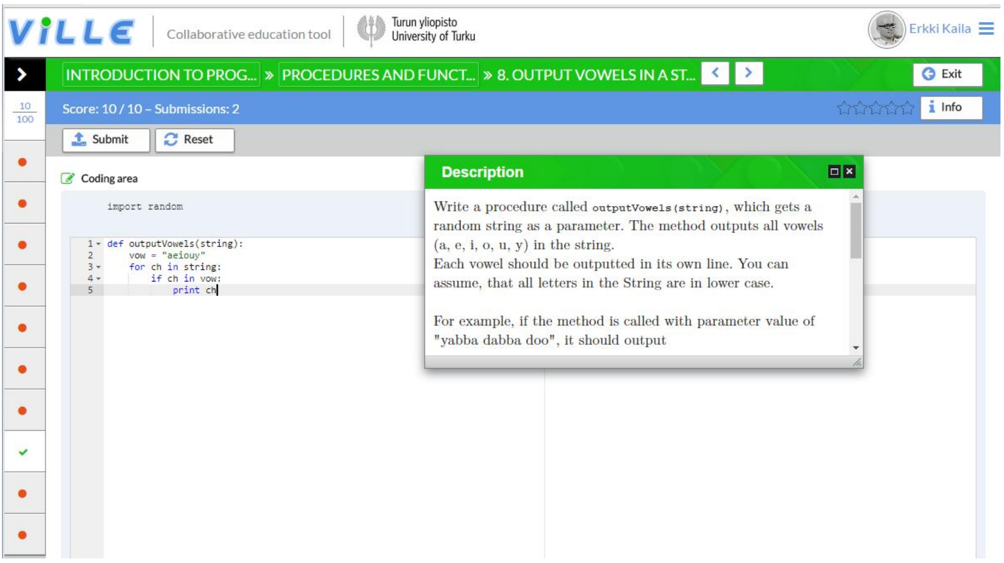
ViLLE has around one hundred different kinds of exercise types, divided into three main categories:

1. Computer science related exercises: exercises that are meant for teaching programming and computer science in general. Some examples are coding exercises, visualization exercises (similar to ones in ViLLE 1), Parsons Puzzles (see Parsons et al. 2006), conversion exercises and program simulation exercises.
2. Math exercises: ViLLE contains several exercise types designed for mathematics learning. Most of these exercises are designed for elementary school level (including several math drilling games), but there are exercises for university level mathematics as well.
3. General exercises: there are several exercise types that can (and are) used to teach any subjects. These include traditional quizzes and

surveys alongside with different kinds of categorization, sorting and selection exercises.

3.2.2.1 Student View

The student view of ViLLE 2 is divided into four separate sub views. In the *Profile view*, the students can see an overview of their courses, latest messages from teachers, the upcoming deadlines for rounds and the collected virtual achievements. In the *Course view* all rounds (excluding the ones hidden by the teacher) are displayed, and similarly the *Round view* displays all assignments in the round. The assignments are answered in the *Exercise view* (see Figure 5).


ViLLE
Collaborative education tool
Turun yliopisto
University of Turku
Erkki Kaila
INTRODUCTION TO PROG... » PROCEDURES AND FUNCT... » 8. OUTPUT VOWELS IN A ST...
Exit
10
100
Score: 10 / 10 – Submissions: 2
Info
Submit
Reset
Coding area
Description

```
import random

def outputVowels(string):
    vow = "aeiouy"
    for ch in string:
        if ch in vow:
            print ch
```

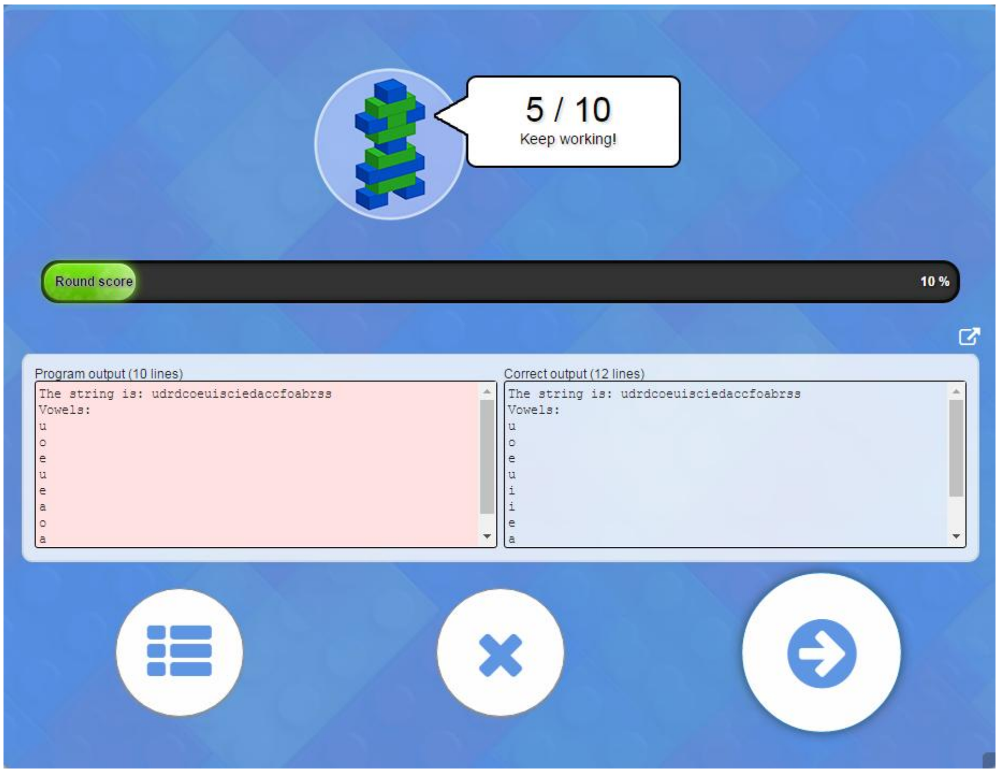
Write a procedure called `outputVowels(string)`, which gets a random string as a parameter. The method outputs all vowels (a, e, i, o, u, y) in the string.
Each vowel should be outputted in its own line. You can assume, that all letters in the String are in lower case.

For example, if the method is called with parameter value of "yabba dabba doo", it should output

Figure 5 ViLLE's Exercise View in the student mode. The exercise displayed is a coding exercise, where a student needs to write a program (or a missing part of a program) according to given specifications.

A majority of the view is reserved for the exercise to be solved. Additionally, the view displays the obtained score and number of submissions. The student can also view the exercise description and attached materials in their own windows. In exam round, most of the feedback (such as the score obtained) can be set as hidden by the teacher.

Most of the exercises in ViLLE are automatically assessed and provide immediate feedback when submitted. These allow students to immediately retake the assignment, if there were errors in it. To allow meaningful iteration, the exercises are randomly initialized when possible. An example of the immediate feedback is displayed in Figure 6.



Program output (10 lines)

```
The string is: udrdcoeuisciedaccfoabrss
Vowels:
u
o
e
u
e
a
o
a
```

Correct output (12 lines)

```
The string is: udrdcoeuisciedaccfoabrss
Vowels:
u
o
e
u
i
i
e
a
```

Figure 6 Immediate feedback provided by the coding exercise shows the student output along with the model answer's output. The exercise is initialized with random seed to provide different input values with each submit.

The amount and nature of feedback naturally varies between different kinds of exercises. Simplest form is to color the choices with different colors according to whether they're correct or incorrect, while the most complicated ones display for example the full solution to quadratic equation with graph and other information included.

3.2.2.2 Teacher View

The teacher view in ViLLE has three distinctive main functions: creating resources, browsing and utilizing existing resources and evaluating student performance. The resource creation is done with dedicated editors for courses, tutorials, exercises and materials. Since all content in ViLLE can be shared with all other teachers, there are separate content browsers for courses, exercises and materials. The exercise browser is displayed in Figure 7.

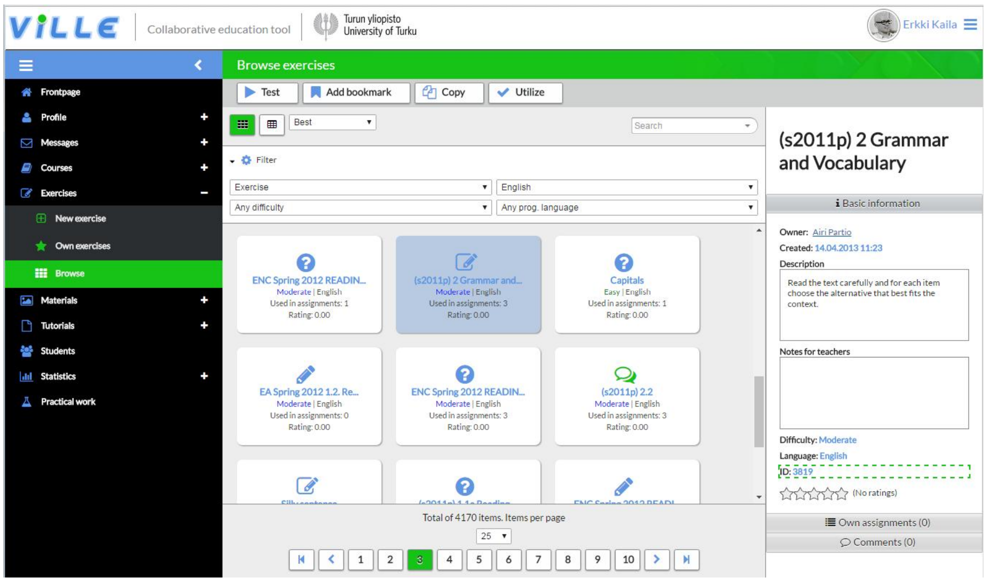


Figure 7 ViLLE's Teacher view in Exercise browser mode. In total, there are more than 40,000 public exercises. In the figure, the user is browsing for English exercises.

Teachers can browse and search exercises based on their name, description or tags attached. ViLLE utilizes features familiar from social media applications, such as commenting and rating, to enable easier browsing and interaction with other teachers.

Finally, there are dedicated views for analyzing and evaluating student performance and other activities. Statistics views provide detailed information about students' submissions and enable evaluation of manually graded tasks, such as essays or study journals. ViLLE also supports a number of other manual tasks, such as recording of attendances or demonstrations via RFID (Radio Frequency Identification) tags and readers, course projects, peer reviews and discussion forums.

3.2.2.3 Collaboration

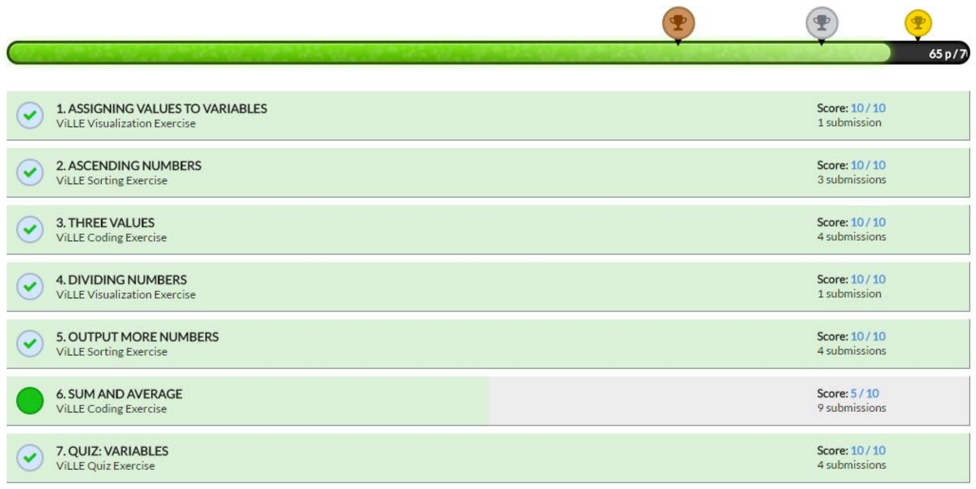
Collaboration was one of the major design principles behind the new ViLLE. Collaboration is enabled from two different perspectives. First, as described before, VILLE 2 supports teacher collaboration by enabling resource sharing, browsing, rating and commenting. From students' point of view, ViLLE supports collaboration specifically in tutorial rounds (although nowadays the support is extended to other rounds as well): two students can work together in answering the tutorial using the same computer. All submissions are recorded for both students logged into same session. This mechanism supports our research team's earlier results on the benefits of collaboration (see Rajala et al. 2010, Rajala et al. 2011), where the benefit of the collaboration comes largely from discussing the tutorial with another student.

3.2.2.4 Active Learning with ViLLE

Active learning, another important design principle, is considered all around ViLLE. Automatic assessment and immediate feedback (Laakso 2010) are utilized in most of the exercise types. As mentioned before, these enable active learning regardless of the time and the place. Additionally, most of the manually graded assignment types support constructive and active learning as well. For lecture-time activation, ViLLE has a built-in lecture poll tool (Kurvinen et al. 2016), which can be used to create polls that can be opened during the lecture and answered via computer or ViLLE Mobile application. The results can be visualized instantly after the question is answered.

3.2.2.5 Gamification and Continuous Assessment

ViLLE 2 supports gamification with various features. First, there are game-like exercises that can be used to make answering more motivating and fun. Second, ViLLE supports virtual trophies, medals with score levels that can be customized for individual students. Example of virtual trophies is displayed in Figure 8**.**


65 p / 7

✓	1. ASSIGNING VALUES TO VARIABLES ViLLE Visualization Exercise	Score: 10 / 10 1 submission
✓	2. ASCENDING NUMBERS ViLLE Sorting Exercise	Score: 10 / 10 3 submissions
✓	3. THREE VALUES ViLLE Coding Exercise	Score: 10 / 10 4 submissions
✓	4. DIVIDING NUMBERS ViLLE Visualization Exercise	Score: 10 / 10 1 submission
✓	5. OUTPUT MORE NUMBERS ViLLE Sorting Exercise	Score: 10 / 10 4 submissions
●	6. SUM AND AVERAGE ViLLE Coding Exercise	Score: 5 / 10 9 submissions
✓	7. QUIZ: VARIABLES ViLLE Quiz Exercise	Score: 10 / 10 4 submissions

Figure 8 Virtual trophies displayed in ViLLE's round view. The user has completed almost full score from all assignments.

The figure displays student's total score in a single round, with virtual trophies set to defined limits. Moreover, users can be awarded virtual achievements, either automatically (for example when a certain amount of rounds or assignments is completed with full scores) or manually by the teacher (for example when a student makes an important observation or a clever question in classroom).

3.3 Summary

The two versions of ViLLE are rather different. Still, the work and research done with the first version are almost completely utilized in the second version. A comparison of the two versions, based mostly on the features observed in this thesis, is displayed in Table 1.

Table 1 A comparison of ViLLE's versions 1 and 2.

Feature	ViLLE 1	ViLLE 2
Technical implementation	Java applet served via external server application	Java server with Vaadin framework, serving HTML and JavaScript
Viewer	Web browser with Java support	Web browser with no additional plugins required
Exercise types	Visualization, code sorting, coding	Around 100 public exercise types
Course management	None	Support for multiple courses per teacher
Resource sharing	No	All resources can be shared with all teachers registered into tool
Programming language support	Various	Various
Possible to create new program language syntaxes	Yes	No
Teachers can create own exercises	Yes	Yes
Exam rounds	No	Yes
Tutorial rounds	No, but exercises can be embedded into external web pages	Yes
Manually graded assignments	No	Yes
Lecture attendances and demonstrations	No	Yes, with RFID servers connected to main server, and with portable RFID readers connected to client
Surveys	No	Yes
Statistics	External server supports the displaying of student scores	Multifaceted, including learning analytics
Student collaboration	No	Yes
Automatic assessment	Yes	Yes
Immediate feedback	Yes	Yes

4 Utilizing ViLLE in Programming Education

Potentially, educational technology allows teachers to enhance their teaching, to improve student performance and motivation and to lower drop-out rates. Still, although an adequate tool is definitely a requirement for effective results, it is by no means a definitive answer. What to use is an important question, but how to use it might be even more important when it comes to educational technology. In this section, I review the usage of ViLLE in different programming and computer science courses. This is done by first introducing the features utilized in adapting the tool, and then showing their effectiveness in the light of the results obtained.

4.1 Laying the Foundation: Controlled Tests

Before adapting ViLLE into courses, the effect of several features were tested in controlled tests. The setup of the tests was always identical. The students were randomly assigned into two groups (control and treatment), and before the procedure, both groups answered a pre-test. The time reserved for pre-test was typically 15 minutes. After this, the treatment group utilized ViLLE while the control group either used a stripped-down version (to test specific features) or used an alternative form of education (such as programming tutorial in paper). Finally, the learning effects were measured with a post-test. The post-test always matched the questions in the pre-test (although with variating content or phrasing), but often also contained some additional questions. Hence, the time reserved for the post-test was usually a little longer, typically 30 minutes. The structure of controlled tests is displayed in Figure 9.

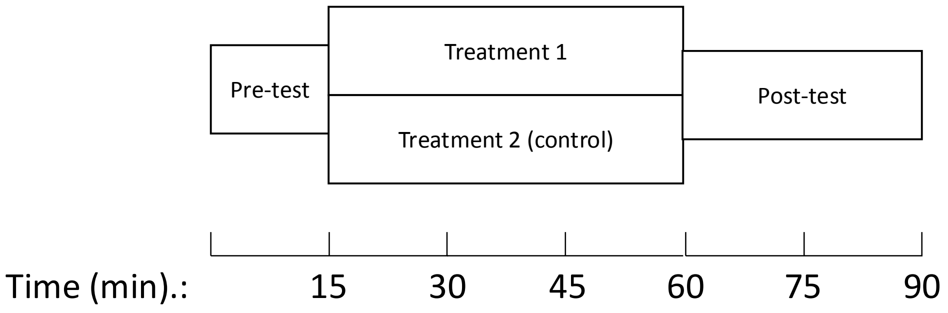


Figure 9 The structure of controlled tests; note, that sometimes more than two different treatments were used.

In the scope of this thesis, the controlled tests (reported in P1) were used to test the effectiveness of ViLLE, the role of engagement in the learning and the effect of cognitive load when using a learning tool. All tests are summarized in this Section.

4.1.1 Study 1: Effectiveness of ViLLE

The effectiveness was tested by utilizing a web-based programming tutorial with intention to teach fundamental programming concepts (see P1, section 3.1 for details). The tutorial consisted of text, images and code examples. The difference between groups was that the treatment group (N=32) was able to visualize the examples in the tutorial by using ViLLE 1, while the control group (N=40) used the tutorial without the tool. The study was conducted to find out if ViLLE can be effectively used to teach basic programming concepts, and whether previous programming experience has an effect on the learning performance.

The pre-test and post-test results are displayed in Table 2**.**

Table 2 The pre-test and post-test results from a controlled test measuring the effectiveness of ViLLE 1.

Scores	Control Group	Treatment Group
Pre-test	10.58	10.41
Post-test	17.55	18.13
Total difference	6.98	7.72
p-value	0.000	0.000

As seen in the table, both groups performed statistically significantly better (result obtained via two-tailed t-test, see Rajala et al. 2008) in the post-test than in the pre-test, which indicates, that tutorials (such as the one used) can be effectively used to teach programming with rather short exposure time. There is a small trend towards a better performance in both, the post-test score and in the total difference favoring the treatment group. However, the differences are not statistically significant.

The earlier programming experience was taken into account by further dividing both groups into two subgroups based on their earlier programming experience. As expected, there was a statistical difference between students with earlier experience in the pre-test in treatment and control groups. However, in the post-test the difference remained only in the control group. Hence, it seems, that accompanying the code examples in the tutorial with visualization exercises is especially useful for novice programmers. The comprehensive results, including an analysis of individual questions in the pre-test and post-test is given in section 3.1 of P1.

4.1.2 Study 2: Role of Engagement

Study 1 was later expanded to contain three groups using ViLLE 1 in different levels of engagement (Naps 2002). The goal was to find out whether there is a change in the learning performance if the visualization tool is used in the *viewing* or *responding* mode. The control group from the previous study was titled as *No-viewing group and* the treatment group as *Responding group* according to the corresponding level in the engagement taxonomy. Additionally, a third group, titled *Viewing group* (N=65), was included in Study 2. The group had access to ViLLE from the tutorials, but ViLLE was used in a lower level of engagement than in the Responding group: with the latter group, visualizations were accompanied with multiple choice questions to facilitate active learning, while the Viewing group could only passively follow the same visualizations.

As concluded in Study 1, there were no statistically significant differences in the post-test scores between groups. Again, when the previous programming experience was taken into account, the difference between experienced programmers and novices in the pre-test vanished only in the Responding group. Hence, it seems that ViLLE 1 is an effective tool for novice programmers, but only if used in the higher level of engagement. This seems to be in line with the hypothesis by Naps et al. (2002). To try to confirm the results, a post-hoc Student-Newman-Keuls test was used to form two homogeneous subsets. The results, displayed in tables 8 and 9 of P1 seem to show similar results than the previously mentioned analysis: the difference between experienced students and novices still disappeared only in the Responding group.

4.1.3 Study 3: Effect of Cognitive Load

The final controlled test reported in P1 was conducted to find out if the previous experience of using the educational tool has an effect into the results. The assumption was that cognitive load (see for example Sweller 1994) of adapting a new tool has a negative effect to learning performance. Study 3 was conducted in two instances of a high school programming course. The students who had experience about using ViLLE 1 before the study were chosen as a treatment group (N=7) while the students who used ViLLE for the first time acted as a control group (N=17). In the study, both groups used a similar tutorial, including the possibility to visualize the code examples with ViLLE.

The pre-test and post-test scores for both groups are displayed in Table 3**.**

Table 3 Pre-test and post-test scores for both groups; post-test scores are displayed for shared questions (i.e. questions common to pre-test and post-test) and for all questions in the post-test.

	Pre-test total	**Post-test total (shared)**	**Post-test total (all)**
Control Group (N=17)	7.12	12.59	16.94
Treatment Group (N=7)	9.43	19.57	26.43
p-value	0.515	0.047	0.046

As seen in the table, there is no statistical difference (obtained via two-tailed t-test, see Laakso et al. 2008) between groups in the pre-test. However, in the post-test the treatment group outperformed the control group statistically significantly. Hence, it seems that cognitive load has a significant effect to learning performance, and it would be beneficial to properly introduce the learning tool before utilizing it.

4.1.4 Summary of Controlled Tests

There are hence at least three important findings from the controlled tests that were utilized when ViLLE was later adapted in course-long use. First, the tool can be used to teach the basic programming concepts effectively (P1 and Rajala et al. 2008). Second, the visualizations are only useful if used in higher levels of engagement (P1 and Kaila et al. 2009, Laakso et al. 2009), and third, the students should be familiarized with the tool before utilizing it (P1 and Laakso et al. 2008). The results from the controlled tests were used as basis when ViLLE was adapted to computer science and programming courses later. The next two sections describe the design of the new course methodologies along with the results obtained.

4.2 Utilizing the Program Visualization Tool

In this section, the cases about utilizing the program visualization in computer science and programming courses are illustrated. The case numbering is continuous throughout this and the following section to enable easier comparison and summarizing of cases in Section 4.4.

4.2.1 Case 1: High school programming course

The first reported study (P2) about the adaptation of ViLLE 1 into course-long use was conducted in high school level. Naturally, the results from the controlled tests were utilized in adaptation: ViLLE was used in engagement level of responding, and the students were properly familiarized with the tool before it was used in the course. In addition to the previously mentioned design principles, course-long adaptation required new issues that needed to be addressed. Rewarding students about using the tool was one of them. As later shown in P3, if the exercises are served as non-mandatory or with no external reward (such as bonus points for the final exam), it is likely that the students do not spend as much time using the tool.

The study was conducted in three instances of a high school level programming course. The first two instances, serving as a control group (N=15), utilized ViLLE only in controlled test setup similar to ones reported in the previous section. The third instance, the treatment group (N=7), used ViLLE throughout the course. ViLLE was integrated into course by including a set of ViLLE exercises to be completed after each lecture. The exercises were automatically assessed, gave immediate feedback, included multiple choice questions and graphical array questions (and were hence in the higher engagement level) and were made mandatory. Hence, the course adaptation followed the principles found to be effective.

Since both groups participated in a controlled test, the pre-test of that setup can be used to measure their initial skills in programming. All groups also took a similar final exam at the end of the course. To measure the effects as validly as possible, all variables except the usage of ViLLE were kept unchanged when possible: all instances were taught by the same teacher, all materials were kept identical and the content of the final exam was similar. Since the control group had no access to ViLLE, similar code examples were provided instead. The results of the pre-test and the final exam results are combined into Table 4.

Table 4 The pre-test and the final exam scores for both groups.

	Pre-test total (max. 30)	**Final exam total (max. 60)**
Control group (N=15)	7.79	26.33
Treatment group (N=7)	10.83	39.79
p-value	0.430	0.030

As seen in the table, there are no statistical differences (analyzed via t-test) between the groups in the pre-test, but there is a significant difference in the final exam. The same difference can be found in separated code reading and writing sections of the final exam (see Table 3 in P2). The results seem to provide quite clear evidence that utilizing ViLLE exercises results into significantly better learning results. From the course methodology point of view, this indicates that active learning with engaging exercises is an effective method for education.

4.2.2 Case 2: Introductory Computer Science Course in University Level

Although the first experiment in adapting ViLLE into a programming course was successful, the study was still conducted with rather small groups and in high school level only. Hence, the next reported study (P3) was conducted in a university level computer science course. The course serves as an introduction to computer science fundamentals (such as data representation and the concept of an algorithm), and also introduces fundamental programming concepts to students. It is mandatory for all computer science majors and also for some other majors in the faculty. Three instances of the course, taught between 2007 and 2009, were selected for the study.

In the first instance, ViLLE 1 was introduced as an optional learning tool. A set of visualization exercises was prepared, and a link to them was offered for the students. In the two latter instances ViLLE exercises were made mandatory. The exercises were also integrated more tightly into the course curriculum: instead of offering all exercises in one collection, exercise rounds were opened after the corresponding topic was covered in the lecture. Mostly visualization exercises of ViLLE 1 were used in all instances, but some supplementary code sorting exercises were also included in some rounds. In the final instance, couple of coding exercises were introduced, but their completion was optional.

The course was evaluated based on the final exam. The accepted performances were graded in scale of 1 to 5. The pass rates and grade averages for all instances are displayed in Table 5.

Table 5 The pass rates and grade averages from all instances of the course. ViLLE exercises were optional in 2007 and mandatory in latter instances.

Instance	2007	2008	2009
N	131	134	181
Pass rate	80.92%	82.09%	85.08%
Grade avg. (max. 5)	3.13	3.48	3.31

As seen in the table, there is a trend towards better pass rate and better grade average in the latter instances, where ViLLE exercises were mandatory. The grade distributions between instances were also compared with a Chi-test (P3, Table 3), which shows that the distribution in the first instance is independent while the other two follow the same pattern. Hence, it seems that making the exercises mandatory has a positive effect on the learning performance.

The students were also surveyed for their perceptions of ViLLE 1 in the 2008 instance (P1, section 3.4). In the first section of the three-part questionnaire, the students were asked to evaluate statements about the tool in a Likert scale of 1 to 7. The students found ViLLE suitable for teaching programming (average 5.64) and quite easy to use (avg. 5.49). In the second section, the students were asked to evaluate the usefulness of ViLLE in understanding programming concepts (P1, Table 12). Based on the answers, ViLLE's visualization exercises are seen most useful for understanding loops and conditional statements, but are not deemed as useful for arrays. In the final section, the students evaluated the usefulness of ViLLE's individual features. Visualization of variable states and automatic assessment were listed as the most useful.

4.3 Utilizing the Collaborative Education Tool

After the aforementioned studies were conducted, ViLLE 2 was introduced. Hence, the final three cases, reported in this section, are conducted by using the latest major version of the tool. Regardless, the design principles that were proven effective by using ViLLE 1 were utilized in the adaptation of the new version as well.

4.3.1 Case 3: Introductory Programming Course for Bioinformatics Majors

The third observed case (P3) is an introductory programming course aimed for bioinformatics majors. The course is quite typical first programming course, taught in Python and following the imperative paradigm. The course does not contain bioinformatics specific content, but the target audience should be noted for another reason: most of the introductory programming courses (and most of the research done about them) are aimed for computer science majors or engineering students. A holistic methodology should be effective in all courses, regardless of the students' age or major topic.

A total of three instances of the course were observed between years 2010 and 2012. The new version of ViLLE was utilized in all three instances, but new exercise types and other features were introduced in the latter instances. In the first instance, a set of visualization exercises similar to one included in ViLLE 1 was utilized. In the second version, a heterogeneous set of different kinds of exercise types was introduced. In addition to visualization and code sorting exercises, puzzles, coding exercises and quizzes were utilized. The goal of variating exercise types was to make the set more motivating and to cover more aspects of programming. In addition, some of the new exercise types cover higher levels of engagement: for example, coding exercise is in the constructing level.

In the final observed instance, even more features were included. The final exam was replaced with an electronic ViLLE exam. An electronic exam has several benefits over traditional pen and paper exam: first, automatic assessment means that evaluating the answers is significantly faster and more neutral. Second, utilizing electronic exams provides students to test, debug and fix their answers before submitting them. This means, that the programs could be written in a near-authentic environment. In the 2012 instance ViLLE was also used to register lecture attendances and demonstration assignments instead of pen and paper method used earlier. Whenever electronic exams were utilized in the studies, their difficulty level was evaluated by at least three non-affiliated researchers and/or teachers to match the pen and paper exam used previously (in fact, the exam was only used after all the reviewers agreed that the electronic exam was at least as difficult as the pen and paper version).

The pass rates and grade averages for all observed instances are displayed in Table 6**.**

Table 6 Pass rates and grade averages of all observed instances of the course.

Instance	2010	2011	2012
N	23	16	25
Pass rate	95.65%	87.50%	96%
Average grade (max. 5)	3.52	3.75	4.04

As seen in the table, the pass rate has been quite high in all instances; the little drop in 2011 is because instead of one student, there were two that failed the course that year. There also seems to be at trend towards a higher grade average after heterogeneous exercise set was introduced in 2011. The grade average in all instances in nevertheless higher than in the instances of the course not utilizing ViLLE (average of 2006 to 2009 was 3.14). After introducing variating types of exercises, it also seems that the students completed more ViLLE exercises, as can be seen in P3, Table 6. Hence, utilizing different exercise types seems to have a positive effect to exercise completion rate and to student performance.

4.3.2 Case 4: Introductory Programming Course for CS Majors

Probably the biggest changes in the methodology were done between cases 3 and 4. Experiences collected from the controlled tests and previous adaptation cases were considered when an introductory programming course at University of Turku was thoroughly renewed between academic years of 2011 and 2012 (P4). The changes were made with three major steps: 1) facilitating active learning and collaboration, 2) underlining the importance of lectures and 3) redefining testing procedure with electronic exam. Other smaller changes were also introduced, including for example the utilization of older students as mentors.

Tutorial-based learning was already utilized in controlled tests (see Section 4.2 and paper P1). The idea behind the concept is the combination of automatically assessed exercises with learning materials, such as text, images or videos. As collaboration was one of the major design principles behind ViLLE 2 and the methodology, tutorials in the course were done in groups of two students. With the introduction of tutorials, the course structure was reformatted to contain two different sessions each week: instead of two two-hour lectures, one lecture and one two-hour tutorial session were conducted. The latter was reserved solely for active learning.

The importance of lectures was underlined with two changes. First, some easy ViLLE exercises were prepared to be answered right after each lecture. The goal of the exercises was to remind the students about the topic, and help them actively process the topics discussed in the lecture. Second, a continuous feedback cycle was introduced. After each lecture and tutorial the students answered to a short survey consisting of three questions: “What did you learn in this session?”, “What things were left unclear in the session?” and “How would you improve the session?” The answers were shortly evaluated after each session and the answers used to fix technical and content errors and to improve the quality of materials and teaching. A complete description about analyzing the surveys can be found in Kaila et al. (2015). As seen in Figure 10, the number of technical and mentoring issues reported got lower in the latter tutorials because analyzing the feedback helped the course staff to fix them early.

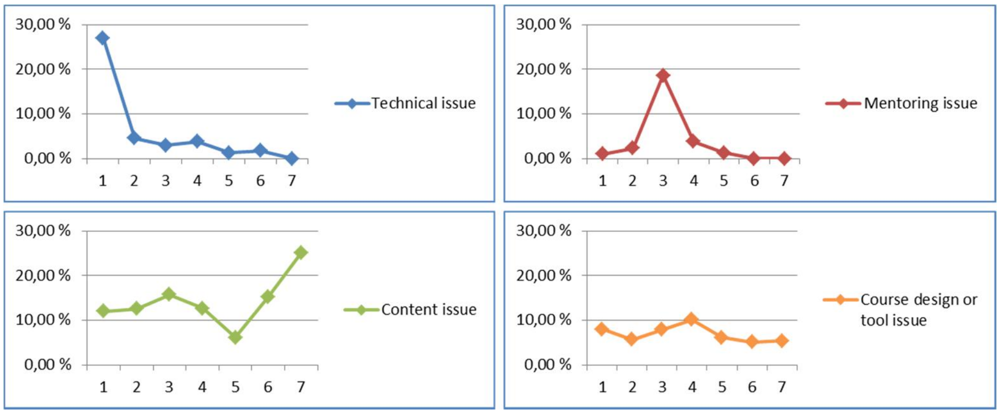


Figure 10 Issues reported by students in tutorials via continuous feedback cycle (Kaila et al. 2015).

Final major change was the utilization of electronic exam. The change was made largely due to the reasons listed in the Section 4.3.1 describing case 3: to provide more authentic programming environment and less biased evaluation method. Electronic exam also enables a larger amount of tasks in less time. The comparison of the electronic exam to the paper version can be seen in P4, Table 1. To ensure the comparability to the earlier instances, the electronic exam and the pen and paper version were again given for evaluation to four non-affiliated educators and/or teachers, who all agreed independently that the new version of exam is at least as difficult as the earlier one.

The redesigned instance of the course (2012) was compared to an earlier instance using the old methodology (2011). The changes in the methodology are combined into Table 7.

Table 7 The comparison of the methodologies used in old and redesigned instances of the course.

Component	2011	2012
Lectures	7 x 2 x 2h	7 x 2h
Tutorials	None	7 x 2h
Demonstrations	4 x 2h	4 x 2h
ViLLE exercises	None	3 to 4 each week in addition to tutorials
Feedback cycle	None	Survey after each lecture and tutorial
Exam	Pen and paper	Electronic

As with the previous cases, the comparison of the courses is done by observing the pass rates and the final grades. In both instances, the grade was affected by the final exam and the demonstrations, while in the latter instance exceeding the minimum limit for the tutorial or ViLLE exercises scores also awarded some bonus points. However, the bonus points were only awarded if the students passed the exam (i.e. received at least 50 % of the maximum points in the exam). The results are combined into Table 8**.**

Table 8 The pass rates and grade averages of all students in both instances of the course.

Instance	2011	2012
N	210	193
Pass rate	53.33%	80.82%
Grade average (max. 5)	3.63	3.57

As seen in the table, the pass rate increased statistically significantly (Mann-Whitney U-Test, p = 0.004 and Kolmogorov-Smirnov test, p = 0.011). Again, the phenomenon is similar to previous cases although the effect seems to be stronger in this case. Curiously, the grade average remained almost the same. However, the grade distribution was quite different (see P4, Figure 3): the number of highest grades as well as the lowest grades is higher in the latter instance.

4.3.3 Case 5: Object-Oriented Course at University Level

The final observed case (P5) is methodologically quite similar to case 4. However, content-wise the course is different to all other programming courses observed in this thesis. All other courses (although utilizing different programming languages and aimed for different target groups) are introductory programming courses. The course in the fifth case, however, is an advanced course at university level with focus on object-oriented programming. The course is mainly targeted for computer science majors and engineering students, but there are some students with other majors as well. All students in the course take the introductory course (case 4) before this course, unless they have completed a similar course somewhere else.

The main focus of the course is in utilizing objects and writing own classes. Some more advanced topics, such as inheritance, interfaces and polymorphism are also covered. The structure of the redesigned course is displayed in Table II, P5. As seen in the table, the structure is quite similar to introductory course presented as case 4. The only major difference is the final project included in case 5. Similarly, the changes in the methodology between old and redesigned instances are similar to previous case, as illustrated in Table III, P5. Again, the effectiveness of the methodology was evaluated by comparing the pass rates and final grades of the old and new instances. In the final case, a total of four instances – two using the new methodology and two the old one – were observed.

The pass rates and grade averages of all four instances are collected into Table 9.

Table 9 Pass rates and grade averages of all instances of the course

Instance	**2011**	**2012**	**2013**	**2014**
Methodology	Old	Old	New	New
N	186	201	191	158
Pass rate	41.94%	50.75%	76.96%	74.05%
Grade avg. (max 5)	3.27	3.37	3.55	4.06

Again, the pass rates are significantly higher (the Test of Equal or Given Proportions implemented in R, $p < 0.001$; see P5 Section 5 for details) in the latter instances using the redesigned methodology. This is in line with all previous cases: utilizing the technology-based methodology seems to increase pass rates, grade averages, or both in the courses. In the fifth case, there is also a trend towards better grade averages in the latter instances. In fact, grade distributions of all instances are statistically different to each other (P5, Table VI), but the difference is a lot smaller among instances sharing the same methodology.

4.4 Summary

All cases are summarized in Table 10.

Table 10 Cases summarized

Case #	Course / level	ViLLE Version	Methodology features introduced
1	High school programming	ViLLE 1	Active learning
2	University CS	ViLLE 1	Making exercises mandatory
3	University programming for non-CS majors	ViLLE 2	Heterogeneous exercise types, electronic exam
4	University programming for CS majors	ViLLE 2	Collaboration, feedback cycle
5	University OO programming for CS majors	ViLLE 2	Whole methodology revisited in object-oriented course

The changes in the methodology were made gradually. Hence, only the final two cases can be seen as a complete implementation. The features utilized in different cases are collected into Table 11.

Table 11 Features of the methodology utilized in different cases

Feature	**Case #**				
	1	**2**	**3**	**4**	**5**
Active learning	X	X	X	X	X
Exercises mandatory or reward given	X	X	X	X	X
Cognitive load considered	X	X	X	X	X
Heterogeneous exercises			X	X	X
Lectures and demonstrations registered electronically			X	X	X
Electronic exam			X	X	X
Tutorial-based learning				X	X
Student collaboration				X	X
Continuous feedback cycle				X	X

As seen in the table, the complete methodology was utilized in two last cases only. The individual features as part of the complete model are discussed in section 5.2. The changes in pass rates and grade averages throughout all cases (when available) are collected into Table 12**.**

Table 12 The pass rates and grade averages of all cases. Ctrl *denotes a control instance, usually an earlier instance of the course, where the methodology was not utilized, and* Treat *a treatment instance where the methodology was applied. If more than one control or treatment instance is observed in the case, the instances are combined.*

	Pass rate %			**Grade or score average**		
Case #	**Ctrl.**	**Treat.**	**Change**	**Ctrl.**	**Treat.**	**Change**
1	N/a	N/a	N/a	26.33	39.79	+51.12%
2	80.92	83.81	+3.57%	3.13	3.38	+8.06%
3	95.65	92.68	-3.1%	3.52	3.93	+11.56%
4	53.33	80.82	+51.55%	3.63	3.57	-1.65%
5	46.52	75.64	+62.60%	3.32	3.78	+13.82%

As seen in the table, the changes have been mostly positive in all cases after the methodology was introduced. There are some issues that need to be noted when observing the table. First, the negative change in pass rate in case 3 is a bit misleading, as the pass rate was very high in all instances. In fact, the difference is based on one of treatment instances having two failed students instead of one in all other instances. Moreover, the change in grade average in the same case would be higher (+20.38%) if all instances using ViLLE would be compared to previous instances of the course. Still, in concept of the whole methodology this is less relevant, as the case is included to illustrate changes based on the usage of heterogeneous exercises.

In addition to quantitative results, student feedback was collected in various instances throughout the cases. In addition to feedback about usefulness of ViLLE 1 and its features discussed in Section 4.2.2, students' perceptions were collected about electronic exams (P5, Table VII) and collaboration (Rajala et al. 2015). The results indicate, that students favor electronic exams over pen and paper and found ViLLE feasible as an exam tool. Moreover, the students seem to prefer collaboration over working alone, and see knowledge sharing, faster problem solving and improved learning as the greatest benefits of collaborative work in tutorials.

5 A Research-Based Model for Utilizing Educational Technology in CS Education

The results presented in the previous section enable a design of a functional model for teaching programming and computer science by utilizing educational technology. In this section, the model is presented. First, the results and the features are discussed to provide an adequate scientific background for the model. The suggested model and its adaptation are presented in detail after that, followed with requirements and possible limitations of the model and its utilization in the courses.

5.1 Results Revisited

The controlled tests were used as the basis for developing the model for the methodology. The first two cases reported (P1) are probably the most significant ones, as they provide evidence on the possibilities of active learning: it is possible to improve learning performance significantly in rather short exposure time. However, for the full effect, the students should be properly engaged with the tool. The cognitive load also seems to have a strong effect on the learning performance. To maximize the effect, the students should be familiarized with the tool before the treatment. When adapting a tool into a course, this should be considered, even though the students probably get accustomed to the tools during the course.

There are some things that need to be noted about the results. In the first two cases there are no statistically significant differences between the treatment and the control group (although there is a trend towards a better performance in the treatment group), if the previous programming experience is not taken into account. However, both groups improved their performance statistically significantly when the pre-test scores are compared to the post-test scores. This finding has relevance of its own: tutorials can be used to teach quite complicated concepts in relatively short time. This phenomenon is later utilized in the methodology (see P4 and P5).

Other controlled tests performed by our research group, but not affiliated with this thesis, provided valuable information that was also used as basis for designing the methodology. For example, the studies done about collaboration (Rajala et al. 2009, 2010, 2011, Laakso et al. 2009) provided essential information about the benefits of pair-work when working with exercises: the students' learning performance can be significantly improved if working in collaboration, and if appropriate learning material is combined with exercises, the students will spend the majority of the time in the session discussing the topic at hand. This information was later utilized when tutorial-based learning was designed as part of the final methodology, applied in cases 4 and 5 (P4 and P5).

The features were presented in the course-long adaptation cases gradually. The first two cases, utilizing the first version of ViLLE, provide evidence on the effectiveness of two major features in the methodology: active learning with engaging exercises, and making exercises mandatory. The third case introduced two new features, heterogeneous exercise types and an electronic exam. The final two cases utilize the full methodology, with tutorial-based learning (where collaboration is an important feature), continuous feedback cycle and extended mentoring. The final cases naturally utilize all features that were found effective in the controlled tests and in the previous cases.

Applying the methodology seems to have a positive effect in all cases. However, there is some variation on the nature and extent of the effect. The pass rate in courses has improved in cases 2, 4 and 5 (although the improvement is not significant in case 2). There was no data available about the pass rate in the first case, and in the third case the pass rate decreased slightly. The drop is not significant, and can be explained with the low number of participants and the high pass rates in all observed instances. Notably, the positive effect seems to be the highest in the last two cases, where the full methodology was applied. Hence, it seems that the features combined provide the effect in the full extent. The effect can also be seen in the number of participants of instances in case 5 (Table IV, P5): the final instance has notably lower number of participants, probably due to high pass rate in 2013 instance.

The changes in the grade average were positive in all cases, except for case 4. Again, the drop is quite small in the fourth case, and is not statistically significant. Still, the grade average is a little lower than in the control instance. The reason for this might be found in the grade distribution (Figure 3, P4): the grade distribution in the treatment instance centers more around the highest and the two lowest grades. The difference between grade averages in cases 4 and 5 is also something that needs to be noted: the methodology applied was similar, but the grade average improved with close to 14 percent in the latter. In my opinion, there are two possible explanations for this: first, the topic was different, and it is possible that object-oriented exercises provide different score distribution in the final exam. In fact, it seems that the grade distribution in the latter course (Table V, P5) is a little different than that in case 4. Also, the starting grade average in the case 4 was already higher than in the fifth case, so there was less room for improvement.

The second explanation might be the cumulative effect from the previous course. The student who took the course reported in case 5 usually take the introductory course reported in the previous case as well. Hence, in the treatment instances, a large number of students was already accustomed to methodology before starting the course. Still, this kind of cumulative effect is difficult to measure from an ethical point of view, as offering the highest quality courses possible should be the responsibility of every teacher. Hence, experimenting with different combinations of new and old methodology would not be ethically sustainable.

5.2 The Model

Based on the results and their significance, a research-based model for technology-enhanced education can be constructed. The model contains five features which are discussed below with the appropriate results linked.

1. **Active learning and continuous assessment**
   The first (and to some extent the most important) feature is active learning. Active learning, in this context, means utilizing exercises that engage the students in the engagement level (Naps et al. 2002) of responding or higher. Although the engagement taxonomy is a hypothesis about visualization tools, in my opinion, it can be (to some extent) applied to any educational tool. The benefits of active learning are justified in controlled tests in P1, but the feature is applied in all cases.

   Continuous assessment should be enabled by making exercises compulsory or giving a reward on completing them. As seen in case 2 (P3), if the exercises are made optional, the learning performance is worse than when they are required to complete the course. Making exercises mandatory can also increase their significance in students' perceptions. Optionally, a reward can be given for completing the exercises. This type of reward is external, and is typically offered as bonus points for final exam, bonus in course grade or exemption from other work at the course. The two cases can also be combined (like done in cases 4 and 5): a certain minimum amount of work is mandatory, and exceeding this limit gives students a reward. To enable this, the score collected should be visible for the student at all times – excluding the exam.

2. **Heterogeneous exercise types**
   Utilizing different kinds of exercises in the course has at least two possible benefits: the motivation for answering the exercises can be increased (as repeating same type of exercise can get boring), and the topics can be observed from different points of view. As seen in case

3, utilizing heterogeneous exercise types can have a positive effect on student performance, at least in the light of the grade averages.

3. **Electronic exam**

   In programming and computer science courses, electronic exam provides several benefits over a traditional exam done in pen and paper. The possibility to compile, execute, test and debug code before submitting the answer takes the whole process closer to actual programming. The process is also faster than when using pen and paper, which means that it is possible to test students' knowledge in wider scale in the same time. Another advantage is the possibility to use automatically assessed exercises, which both saves teachers' time significantly and makes the evaluation process less biased. All exercise types that are utilized in practicing the topic should be applicable in the exam as well. Electronic exam was utilized in cases 3, 4 and 5. Although there probably is no effect on the student performance (and to keep the evaluation of course instances valid, there should not be), the utilization makes the exam process a better fit into the whole methodology. Student feedback about the electronic exams has also been very positive (see Table 13).

Table 13 Students' perceptions of electronic exam in two programming courses. Although observed instances are different, the courses are the same (and use the same redesigned methodology) than the ones observed in P4 (Course 1) and P5 (Course 2). The comprehensive results are presented in Rajala et al. 2016.

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
The difficulty of the exam content (1 – low, 5 – high)	2.75	3.27

It is to be noted, that the evaluation of the course should not be limited to the final exam at the end of the course, but instead continuous assessment throughout the course should be utilized if possible. Utilizing tutorial based learning means that students can collect points from exercises and similar tasks (such as attendances and other assignments) starting from the beginning of the course, which gradually decreases the need for comprehensive exam at the end of the course.

4. **Tutorial-based learning**
   The utilization of tutorial-based learning (TBL) is probably one of the most important features in the methodology. Used in cases 4 and 5, TBL means replacing half of the lectures with tutorials, which in this context are combinations of learning material and exercises. The tutorials are answered in class room or computer lab in collaboration with another student. The emphasis is on the active learning, but collaboration brings another aspect to the feature: discussing the topic with a peer can significantly improve the learning performance.

5. **Continuous feedback cycle**
   The fifth feature in the methodology is the continuous collection of feedback from students throughout the course. In practice, after each lecture and tutorial the students are required to answer to the same three questions about what they learned, what remains unclear and how would they improve the session. Since the questions are kept simple, the answers can be evaluated right after the session, and the results utilized immediately to improve the quality of teaching, materials and technical implementation. The feature was utilized in the last two cases, and is observed in more detail in Kaila et al. (2015).

Additional Features: Mentoring and Considering cognitive Load
Although not really a part of the model, the utilization of older students as mentors is highly recommended when the model is applied. The mentors were utilized in cases 4 and 5 and have two important roles. First, they are present in the tutorial sessions along with course staff to aid students in whenever they have problems with the exercises. This way no one gets stuck in any of the exercises. The second usage is a dedicated mentoring session, organized every week. In the session, students can come by to ask for assistance in any of the course work, including for example tutorials or demonstrations. The mentoring session also advances the forming of social groups, which are highly useful for studies and later professional career (Kaila et al. 2016).

Another additional (but still important) feature is recognizing the effect of cognitive load by familiarizing the students with the tool before it is used. As seen in a controlled test (P1), the learning performance can be significantly better if the amount of cognitive load is kept to minimum. In course long usage, the familiarization can be done with dedicated session about using the tool, or for example by having a dedicated set of introductory exercises, that are designed for familiarizing students with the tool. The latter option was utilized in cases 3, 4 and 5, where an introductory round was included in the virtual course in ViLLE. In the first two cases, the tool was introduced to students by the teacher as part of the controlled tests setup.

The complete model with all features can be applied to programming or computer science courses to enhance student performance and the general quality of teaching, as seen in the two final cases (P4 and P5). However, all features in the model are not solely meant for improving the student performance. The intended roles of different features are illustrated in Figure 11.

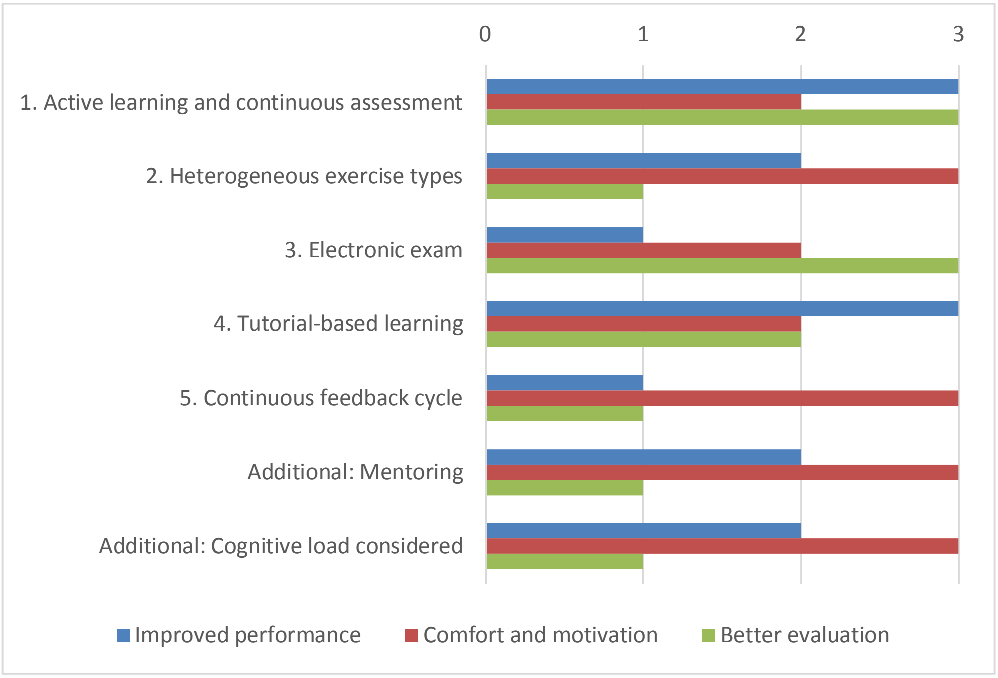
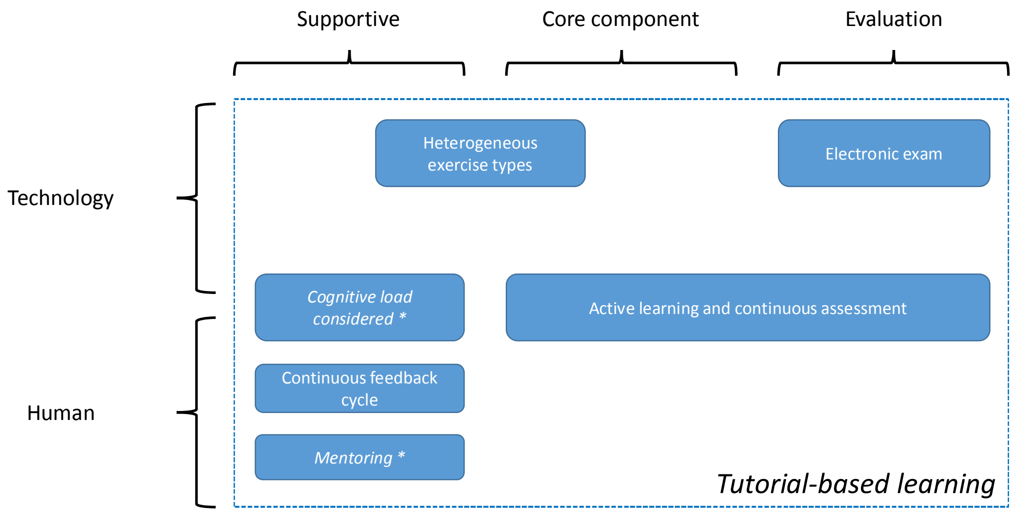


Figure 11 Features in the model categorized by their intended *effect for different outcomes of the course. The scale is based on the desired amount of effect the feature has on the defined category (1 = small effect, 2 = moderate effect, 3 = significant effect).*

In the figure, improved performance stands for better pass rates and better grade average, comfort and motivation denote better course quality from students' point of view, and better evaluation means easier, less biased and higher quality evaluation process. A few things should be noted: active learning has a high intended effect for evaluation, as it enables continuous and more transparent evaluation and a higher number of tasks that can be used as basis for evaluation. The electronic exam presumably has a low effect on the performance, but provides better experience for the students as well as less biased (from student's point of view) and more effortless (from teacher's point of view) evaluation.

The model with all features classified is displayed in Figure 12.



Supportive
Core component
Evaluation
Technology
Human
Heterogeneous exercise types
Electronic exam
Cognitive load considered *
Active learning and continuous assessment
Continuous feedback cycle
Mentoring *
Tutorial-based learning

Figure 12 The model with all features classified, including the additional features (denoted with an asterisk).

The figure displays the classification of all features in two categories: in the x-axis the components are classified based on their role in the whole model, divided into three categories: the core features of the model, the supportive features and the features intended for student evaluation. The further down the feature is in the y-axis, more human-based it is in comparison to technology-based features in the top. Tutorial based learning could also be positioned in the center of the image, but drawing it as an outline around the all other features emphasizes its central role in the model.

5.3 Applying the Model: Requirements and Limitations

The first requirement for applying the model is an adequate learning environment with suitable features. The cases all utilized either the first or the second version of ViLLE, and the complete model was tested by using the latest version. Still, as long as an educational tool fulfills certain requirements, the model should be applicable with any other platform as well. First, the platform should support automatic assessment, immediate feedback, and a number of heterogeneous exercises suitable for computer science and programming education. This enables active learning regardless of the time and the place. Moreover, to enable continuous assessment by making the exercises mandatory (or by offering a reward for completing them), the progress and the scores should be made transparent for students.

The tool should also support electronic exams, preferably by utilizing the same heterogeneous exercise types that are used to practice the topics. Hence, the exam should support constructive answering (such as writing program code or building algorithms) instead of multiple choice questions or short answers only. To enable easier utilization in larger course instances, the exam questions should likewise support automatic assessment. For the continuous feedback cycle, the tool should also support surveys and a practical view for evaluating the answers, either directly in the tool or by exporting them as a text file, spreadsheet or similar.

The utilization of tutorial-based learning also sets some requirements for the tool used. Naturally, the tool should support the combination of learning materials and automatically assessed exercises. For learning materials, at least basic components such as text and images should be supported. A natural approach is to support HTML5 for adding content alongside the exercises. Another requirement is the support for student collaboration. The collaboration should preferably happen with a shared computer, but collaboration over network should be fine as long as it enables discussion about the topic.

Conducting the tutorials also provides some requirements for the staff and the premises. First, there should be enough capacity provided for connecting all the computers into the network at the same time. In cases 4 and 5, the problem was easily solved by having local area network connectors in the lecture hall used for tutorial sessions. Since the group size in the tutorials was quite large, it is possible that using the wireless network would not have provided enough network capacity for everyone. Using LAN connectors instead of wireless network also provided another advantage, as all network traffic could be routed via single switch. This meant, that for example in the course exam, addresses besides ViLLE main server and Java API can be easily blocked.

Tutorials also require a sufficient number of course staff or mentors available during the sessions. For tutorials to be effective, the students should not get stuck into individual exercises. Hence, there should be enough aid available at any given time. In our experience, the students usually require more help at the early stages of the courses, and typically are more self-directed when the course advances a little more. In cases 4 and 5, there was typically one or two more complex tasks in each of the tutorials, and the students required assistance for these usually around the same time. The mentors are also essential for organizing the dedicated mentorin sessions mentioned as the final feature in the model.

All in all, adapting the model requires sufficient expertise on the course staff and the mentors. Preparing suitable content and exercises is essential. For this goal, ViLLE offers teachers a possibility to utilize, rate and comment materials created by other teachers. This enables easy sharing of best practices. Still, if no suitable resources can be found, the workload for constructing the courses that utilize the model can be quite big. However, as discussed before, the improved pass rates (and to some extent, better grades) should provide enough motivation for taking the steps. Luckily, the workload needed to run the courses becomes a lot smaller after the first instance. Utilizing automatic assessment actually reduces the time needed for maintaining the course, and enables teachers to dedicate their time more on personal guidance.

Some possible limitations for adapting the model should be noted. First, the model was constructed by presenting individual features and adapting them into the model when their adaptation provided better student performance. However, the changes made in the two final cases where quite extensive compared to the previous cases. Hence, the effectiveness of all individual features in the model was not tested, as testing all possible combinations of the features would be neither practical nor ethical. One could argue, that all the features in the final model are not required. However, all the features have a dedicated and intended role in the adaptation of the model, as described in section 5.2.

It should be noted, that the application of the model discussed in this thesis is done only in programming courses, although an early sub model was applied into an introductory computer science course (case 2). Based on the results, it is still possible, that a similar model would probably work at least nearly as well in other topics' courses, especially in the topics where automatic assessment can be reliably utilized. Examples include STEM subjects and language education. Still, in topics where the tasks are mostly essays or other similar constructivist writing assignments, automatic assessment is not as useful. This definitely makes adapting the model either difficult or impossible. Another examples of courses where adaptation of the model may be difficult are advanced courses, where the focus is more on a deeper analysis of concepts.

The teachers' effect on the results should also be discussed. Although the variables outside the adapted features (or the complete model) were tried to be kept as similar as possible, there were some cases (for example cases 4 and 5) where the treatment instances of the courses were taught by different teachers than the control instances. Actually, in some cases different teachers were the prerequisite for adapting a new methodology into the courses. Hence, one could argue that some of the effect is due to different course staff. Although it is possible that the teacher influenced the final results, the similar effects found throughout all cases with different teachers seem to predict that the effect of the teacher is not too significant. Another question about teachers' effect is whether the model could be applied to distant courses or MOOCs. Though this would require some major modifications (for example, the collaboration and mentoring should be done virtually), the model could be applicable to online courses as well.

Finally, the proposed model is definitely not the only effective model for utilizing educational technology effectively in programming and computer science courses. Still, there are enough experiments with different setups and target groups presented to prove the effectiveness of the model in education.

Though alternative ways to utilize technology exist, there are some features that probably need to be preserved in any similar model to enhance the performance as much as possible. These include at least the utilization of active learning, mandatory exercises and student collaboration. For other features, there could be alternative implementations that could work equally well. Regardless, the model proposed has been proven to be both, effective and relatively easy to implement. Having a skilled, personal tutor for each student available all the time would almost definitely provide better results than the model. Unfortunately, as educators we need to work in the real world with real world's limitations.

6 Conclusion and Future Work

Education and education technology are complex fields to study. It is often impossible (and by no means practical) to find comprehensive solutions to problems or unambiguous answers to questions. Rather, we need to come up with good enough practices that will, in time, give way to improved or totally new solutions. In this thesis, I tried to answer three research questions. The questions and the related results in this thesis are summarized below.

RQ1. *What kind of features are useful when adapting educational technology into computer science and programming courses?*

There are several useful features presented and discussed in this thesis. Papers P1, P2, P3 and P4 all present features that have been adapted into technology-enhanced learning process, and that have been found useful in the related studies. The list is by no means comprehensive (as no such list can or should ever be), but offers a good base for extension. Some of the most important features are summarized in Section 4.4, Table 10. The usage of features in different cases is described in Table 11 in the same section.

RQ2. *If such features are found, can they be combined into a model of a methodology for utilizing educational technology?*

The final model consisting of the most effective features is studied in papers P4 and P5, and fully presented in Section 5. In addition to five core features, two additional features are suggested. Based on the studies reported in P4 and P5, utilizing the model can provide significant results. Section 5.2 describes the features (including the additional ones) in the model. Figures 10 and 11 display the features classified by their intended effect and their role in the model, respectively.

RQ3. *What kind of requirements and limitations there are to consider when such model is adapted?*

The requirements and limitations of the model are discussed in P5 and, to full extent, in Section 5.3. To summarize, the requirements for adapting the tool are 1) appropriate educational tool, 2) sufficient technical infrastructure, 3) enough staff to prepare the materials and assist the students and 4) sufficient pedagogical and technical expertise on the course staff. The most notable limitations are the incomplete testing of individual features (for ethical reasons described in 5.1) and the focus on computer science courses.

The thesis opens up interesting possibilities for future research. First, the possibility to utilize the same model to teach other topics, for example mathematics or science topics, is something that should be experimented. Second, the possibilities of learning analytics when utilizing the model are very interesting. When the model is utilized, a vast amount of data is collected. This data can be potentially used for detecting students with learning difficulties and for assisting them in their problems. Our research team's first experiment in analyzing this type of data is published in Lindén et al. (2016). Finally, the possibilities for applying the model into different types of programming courses, aimed for different types of students, should be experimented even further. After all, the true goal of this work is to make the education of the future programmers a little easier.

References

Amresh, A., Carberry, A.R. and Femiani, J. 2013. Evaluating the effectiveness of flipped classrooms for teaching CS1. In 2013 IEEE Frontiers in Education Conference (FIE) (pp. 733-735). IEEE.

Anglin, L., Anglin, K., Schumann, P.L. and Kaliski, J.A. 2008. Improving the efficiency and effectiveness of grading through the use of computer-assisted grading rubrics. Decision Sciences Journal of Innovative Education, 6(1), 51 – 73

Auvinen, T., Karavirta, V. and Ahoniemi, T. 2009. Rubyric: an online assessment tool for effortless authoring of personalized feedback. In ACM SIGCSE Bulletin (Vol. 41, No. 3, pp. 377-377). ACM.

Backlund, P. and Hendrix, M., 2013. Educational games – are they worth the effort? A literature survey of the effectiveness of serious games. In 5th international conference on Games and Virtual Worlds for Serious Applications (VS-GAMES), (pp. 1-8). IEEE.

Barrett, H.C. 2007. Researching electronic portfolios and learner engagement: The REFLECT initiative. *Journal of Adolescent & Adult Literacy*, 50(6), 436 – 449

Bergin, S. and Reilly, R., 2005. The influence of motivation and comfort-level on learning to program. In proceedings of the 17th Workshop on Psychology of Programming, PPIG'05.

Bosse, Y. and Gerosa, M.A., 2017. Why is programming so difficult to learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage. ACM SIGSOFT Software Engineering Notes, 41(6), 1–6.

Boyle, T. 2003. Design principles for authoring dynamic, reusable learning objects. Australian Journal of Educational Technology, 19(1), 46 – 58.

Boyle, T., Bradley, C., Chalk, P., Jones, R. and Pickard, P. 2003. Using blended learning to improve student success rates in learning to program. Journal of Educational Media, 28(2-3), 165 – 178.

Bradford, P., Porciello, M., Balkon, N. and Backus, D., 2007. The Blackboard learning system: The be all and end all in educational instruction?. Journal of Educational Technology Systems, 35(3), pp.301-314.

Briscoe, C and Peters, J., 1997. Teacher collaboration across and within schools: Supporting individual change in elementary science teaching. Science Education, 81(1), pp.51-65.

Bruff, D.O., Fisher, D.H., McEwen, K.E. and Smith, B.E. 2013. Wrapping a MOOC: Student perceptions of an experiment in blended learning. Journal of Online Learning and Teaching, 9(2), 187

Čisar, S.M., Pinter, R. and Radosav, D. 2011. Effectiveness of program visualization in learning java: a case study with Jeliot 3. International Journal of Computers Communications & Control, 6(4), 668 – 680.

Coates, H., James, R. and Baldwin, G. 2005. A critical examination of the effects of learning management systems on university teaching and learning. Tertiary Education and Management, 11, 19 – 36

Cooper, S., Dann, W. and Pausch, R. 2000. Alice: a 3-D tool for introductory programming concepts. In Journal of Computing Sciences in Colleges (Vol. 15, No. 5, pp. 107-116). Consortium for Computing Sciences in Colleges.

Cross, J.H., Hendrix, D. and Umphress, D.A. 2004 jGRASP: an integrated development environment with visualizations for teaching java in CS1, CS2, and beyond. In Frontiers in Education, 2004. FIE 2004. 34th Annual (pp. 1466-1467). IEEE

Dalsgaard, C. 2006. Social software: E-learning beyond learning management systems. European Journal of Open, Distance and e-learning, 9(2).

Davies, S.P. 1993. Models and theories of programming strategy. International Journal of Man-Machine Studies, 39, 237–267.

Deterding, S., Dixon, D., Khaled, R. and Nacke, L. 2011. From game design elements to gamefulness: defining gamification. In Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments (pp. 9-15). ACM.

Dougiamas, M. and Taylor, P. 2003. Moodle: Using learning communities to create an open source course management system. MoodleResearch, http://research.moodle.net/33/

duBoulay, B. 1989. Some difficulties of learning to program. In Soloway, E. & Spohrer, C. (ed.). Studying the Novice Programmer. Hillsdale, NJ: Lawrence Elrbaum, 283 – 299

Engstrom, M.E. and Jewett, D. 2005. Collaborative learning the wiki way. TechTrends, 49(6), 12 – 15.

Freeman, S., Eddy, S., McDonough, M., Smith, M., Okoroafor, N., Jordt, H. and Wenderoth, M. P. 2014. Active learning increases student performance in science, engineering, and mathematics. Proceedings of the National Academy of Sciences, 201319030.

Gehringer, E.F. 2001. Electronic peer review and peer grading in computer-science courses. ACM SIGCSE Bulletin, 33(1), pp.139-143.

Giannakos, M.N., 2013. Enjoy and learn with educational games: Examining factors affecting learning performance. Computers & Education, 68, 429 – 439

Gomes, A. and Mendes, A. 2014. A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations. In 2014 IEEE Frontiers in Education Conference (FIE) Proceedings, IEEE, 1 – 8.

Grandell, L., Peltomäki, M., Back, R. J., and Salakoski, T. 2006. Why complicate things?: Introducing programming in high school using Python. In proceedings of the 8th Australasian Conference on Computing Education-Volume 52 (pp. 71-80). Australian Computer Society, Inc.

Grönroos, M. 2011. Book of Vaadin. Lulu.com.

Edwards, S.H. and Perez-Quinones, M.A. 2008. Web-CAT: automatically grading programming assignments. In ACM SIGCSE Bulletin (Vol. 40, No. 3, pp. 328-328). ACM.

Havenga, M., Breed, B., Mentz, E., Govender, D., Govender, I., Dignum, F. and Dignum, V. 2013. Metacognitive and problem-solving skills to promote self-directed learning in computer programming: Teachers' experiences. Sa-educ Journal, 10(2), pp.1-14

Helminen, J. and Malmi, L. 2010. Jype – a program visualization and programming exercise tool for Python. In proceedings of the 5th International Symposium on Software Visualization. ACM. 153 – 162.

Henriksen, P. and Kölling, M. 2004. Greenfoot: combining object visualisation with interaction. In Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (pp. 73-82). ACM.

Hernández, R., 2012. Does continuous assessment in higher education support student learning?. Higher Education, 64(4), pp.489-502.

Huang, W.H.Y. and Soman, D., 2013. Gamification of education. Research Report Series: Behavioural Economics in Action.

Hyrskykari, A. 1993. Development of program visualization systems. The 2nd Czech British Symposium of Visual Aspects of Man-Machine Systems, Praha, 1-21.

Ihantola, P., Ahoniemi, T., Karavirta, V. and Seppälä, O. 2010. 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.

Joy, M., Griffiths, N. and Boyatt, R. 2005. The BOSS online submission and assessment System. ACM Journal on Educational Resources in Computing. 5(3), article 2.

IEEE. (2002) Draft standard for learning object metadata. Available:http://ltsc.ieee.org/doc/wg12/LOM_WD6_4.pdf

Isohanni, E. and Knobelsdorf, M. 2011. Students' long-term engagement with the visualization tool VIP. In Proceedings of the 11th Koli Calling International Conference on Computing Education Research (pp. 33-38). ACM.

Jenkins, T., 2002. On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences (Vol. 4), 53 – 58.

Johnson, B. and Webber, J., 2007. Google web toolkit. Addison-Wesley.

Kaila, E., Kurvinen E., Lokkila E., Laakso M.-J., and Salakoski T. 2015. Enhancing student-teacher communication in programming courses: a case study using weekly surveys. In proceedings of ICEE 2015 - International Conference on Engineering Education, (2015).

Kaila, E., Laakso, M.-J., Rajala, T. and Salakoski, T. 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

Kaila, E., Lindén R., Rajala T., Hellgren N., and Laakso M.-J. 2016. Redesigning methodology for student counseling in the first year IT education. In proceedings of EDULEARN 2016, Barcelona, Spain.

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), 126 – 133.

Kolowich, S. 2013. The professors who make the MOOCs. The Chronicle of Higher Education, 18.

Kuittinen, M. and Sajaniemi, J. 2004. Teaching roles of variables in elementary programming courses. *SIGCSE Bull.* 36, 3 (June 2004), 57–61.

Kurvinen, E., Väätäjä J., Rajala, T. and Laakso, M.-J. 2016. Designing and utilizing a course poll tool to enhance learning activity. In proceedings of EDULEARN 2016 conference, Barcelona, Spain

Kölling, M., Quig, B., Patterson, A. and Rosenberg, J. 2003. The BlueJ system and its pedagogy. Computer Science Education, 13(4), 249 – 268.

Laakso, M.-J., Rajala, T., Kaila, E. and Salakoski, T. 2008. The Impact of prior experience in using a visualization tool on learning to program. In proceedings of CELDA 2008, Freiburg, Germany, 129—136.

Laakso, M.J., Myller, N. and Korhonen, A. 2009. Comparing learning performance of students using algorithm visualizations collaboratively on different engagement levels. Educational Technology & Society, 12(2), 267 – 282.

Laakso, M.J., Rajala, T., Kaila, E. and Salakoski, T., 2008. The impact of prior experience in using a visualization tool on learning to program. Appeared in Cognition and Exploratory Learning in Digital Age (CELDA 2008), 13-15.

Laakso, M.-J. 2010. Promoting programming learning. Engagement, automatic assessment with immediate feedback in visualizations. TUCS Dissertations no 131.

Lahtinen, E., Ala-Mutka, K. and Järvinen, H.M., 2005, June. A study of the difficulties of novice programmers. In ACM SIGCSE Bulletin (Vol. 37, No. 3, pp. 14-18). ACM.

Lee, C.B., Garcia, S. and Porter, L. 2013. Can peer instruction be effective in upper-division computer science courses? ACM Transactions on Computing Education (TOCE), 13(3), p.12.

Leutenegger, S. and Edgington, J. 2007 A games first approach to teaching introductory programming. In ACM SIGCSE Bulletin (Vol. 39, No. 1, pp. 115-118). ACM.

Lindén, R., Rajala T., Karavirta V., and Laakso M.-J. 2016. Utilizing learning analytics for real time identification of students-at-risk on an introductory programming course. In proceedings of EDULEARN 2016 Conference, Barcelona, Spain.

Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O. and Simon, B. 2004. A multi-national study of reading and tracing skills in novice programmers. In ACM SIGCSE Bulletin (Vol. 36, No. 4, pp. 119-150). ACM.

Lopez, M., Whalley, J., Robbins, P. and Lister, R, 2008. Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the Fourth International Workshop on Computing Education Research (pp. 101-112). ACM.

Mayer, R. 1981. The Psychology of how novices learn computer programming. Computing Surveys, 13 (1), 122 – 141.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O. and Silvasti, P., 2004. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. Informatics in education, 3(2), pp.267-288.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.D., Laxer, C., Thomas, L., Utting, I. and Wilusz, T., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. ACM SIGCSE Bulletin, 33(4), pp.125-180.

McGreal, R. 2004. Learning objects: A practical definition. International Journal of Instructional Technology and Distance Learning (IJITDL), 9(1).

March, S.T. and Smith, G.F., 1995. Design and natural science research on information technology. Decision Support Systems, 15(4), 251-266.

Meirink, J.A., Imants, J., Meijer, P.C. and Verloop, N. 2010. Teacher learning and collaboration in innovative teams. Cambridge Journal of Education, 40(2), pp.161-181

Menon, A.S., Moffett, S., Enriquez, M., Martinez, M.M., Dev, P. and Grappone, T. 2004. Audience response made easy: using personal digital assistants as a classroom polling tool. Journal of the American Medical Informatics Association, 11(3), 217 – 220

Miller, H., Haller, P., Rytz, L. and Odersky, M. 2014. Functional programming for all Scaling a MOOC for students and professionals alike. In companion proceedings of the 36th International Conference on Software Engineering (pp. 256-263). ACM

Moreno, A., Myller, N., Sutinen, E. and Ben-Ari, M. 2004. Visualizing programs with Jeliot 3. In proceedings of the Working Conference on Advanced visual interfaces (pp. 373-376). ACM.

Myers, B.A. 1986. Visual programming, programming by example, and program visualization: a taxonomy. In ACM SIGCHI Bulletin (Vol. 17, No. 4, pp. 59-66). ACM

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C. and Balik, S. 2003. Improving the CS1 experience with pair programming. ACM SIGCSE Bulletin, 35(1), pp.359-362.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Velázquez-Iturbide, J. Á. 2002. Exploring the Role of Visualization and Engagement in Computer Science Education. Working group reports from ITiCSE on Innovation and Technology in Computer Science Education ITiCSE-WGR 02, 35 (2), 131-152.

Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R. J., Anderson, J., Fleischer, R., Kuittinen, M. and McNally, M. 2003. Evaluating the Educational Impact of Visualization. Working group reports from ITiCSE on Innovation and Technology in Computer Science Education. ACM Press, 124–136

Nguyen, A., Piech, C., Huang, J. and Guibas, L. 2014. Codewebs: scalable homework search for massive open online programming courses. In Proceedings of the 23rd International Conference on World Wide Web. ACM, 491–502.

Nitko, A.J., 1995. Curriculum-based continuous assessment: a framework for concepts, procedures and policy. Assessment in Education, 2(3), pp.321-337.

Oakes, K. 2002. E-learning: LCMS, LMS — they're not just acronyms but powerful systems for learning. Training & Development, 56(3), 73 – 75.

Pappano, L. 2012. The Year of the MOOC. The New York Times, 2(12), p.2012.

Parsons, D. and 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 (ACE '06)

Pieterse, V. 2013. Automated assessment of programming assignments. In Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research (pp. 45-56). Open Universiteit, Heerlen.

Polsani, P.R. 2006. Use and abuse of reusable learning objects. Journal of Digital information, 3(4).

Porter, L., Guzdial, M., McDowell, C. and Simon, B. 2013. Success in introductory programming: What works? Communications of the ACM, 56(8), 34 – 36

Rajala, T., Laakso M.-J., Kaila E., and Salakoski T. 2007. VILLE - a language-independent program visualization tool. In proceedings of the Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007), Koli National Park, Finland, November 15-18, 2007, Volume 88, (2007)

Rajala, T., Laakso, M.-J., Kaila, E. and Salakoski, T. 2008. Effectiveness of program visualization: A case study with the ViLLE tool. Journal of Information Technology Education: Innovations in Practice, IIP715—32.

Rajala, T., Kaila E., Laakso M.-J., and Salakoski T. 2009. Effects of collaboration in program visualization. Appeared in Technology Enhanced Learning Conference 2009 (TELearn 2009), October 6 to 8, 2009, Academia Sinica, Taipei, Taiwan, (2009) Rajala, T., Kaila, E., Laakso, M.-J. and Salakoski, T. 2010. How does collaboration affect algorithm learning? A case study using TRAKLA2 algorithm visualization tool. In proceedings of 2010 International Conference on Education Technology and Computer (ICETC 2010)

Rajala, T., Kaila E., Holvitie J., Haavisto R., Laakso M.-J., and Salakoski T. 2011. Comparing the collaborative and independent viewing of program visualizations. In Frontiers in Education 2011 Conference, October 12-15, Rapid City, South Dakota, USA.

Rajala, T., Lokkila E., Lindén R., Laakso M.-J., and Salakoski T. 2015. Students' perceptions on collaborative work in introductory programming course. In ICEE 2015 - International Conference on Engineering Education.

Rajala, T., Kaila, E., Lindén, R., Kurvinen, E., Lokkila, E., Laakso, M.-J. and Salakoski, T. 2016. Automatically assessed electronic exams in programming courses. In proceedings of the Eighteenth Australasian Computing Education ConferenceACM.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. 2009. Scratch: programming for all. Communications of the ACM, 52(11), 60 – 67

Robins, A., Rountree, J. and Rountree, N. 2003. Learning and teaching programming: A review and discussion. Computer Science Education, 13(2), pp.137-172.

Saikkonen, R., Malmi, L. and Korhonen, A. 2001. Fully automatic assessment of programming exercises. ACM Sigcse Bulletin (Vol. 33, No. 3, pp. 133-136). ACM.

Sajaniemi, J. and Kuittinen, M. 2005. An experiment on using roles of variables in teaching introductory programming. Computer Science Education 15(1), 59-82.

Sams, A. and Bergmann, J. 2013. Flip your students' learning. Educational Leadership, 70(6), pp.16-20.

Salleh, N., Mendes, E. and Grundy, J. 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.

Simon, B., Kohanfars, M., Lee, J., Tamayo, K. and Cutts, Q. 2010. Experience report: peer instruction in introductory computing. In proceedings of the 41st ACM Technical Symposium on Computer Science Education (pp. 341-345). ACM.

Sorva, J. & Sirkiä, T. 2010. UUhistle: a software tool for visual program simulation. In proceedings of the 10th Koli Calling International Conference on Computing Education Research (pp. 49-54). ACM

Sweller, J. 1994. Cognitive load theory, learning difficulty, and instructional design. Learning and Instruction, 4(4), 295 – 312.

Tang, T., Rixner, S. and Warren, J. 2014. An environment for learning interactive programming. In proceedings of the 45th ACM technical symposium on Computer science education. ACM, 671-676.

Tijani, F. and Callaghan, R., 2017, March. Exploring college students' program comprehension skills from visual to procedural programming. In Society for Information Technology & Teacher Education International Conference (pp. 83-88). Association for the Advancement of Computing in Education (AACE).

Warren, J., Rixner, S., Greiner, J. and Wong, S. 2014. Facilitating human interaction in an online programming course. In proceedings of the 45th ACM technical symposium on Computer science education (pp. 665-670). ACM.

Wheeler, S., Yeomans, P. and Wheeler, D. 2008. The good, the bad and the wiki: Evaluating student-generated content for collaborative learning. British Journal of Educational Technology, 39(6), 987 – 995.

Vihavainen, A., Luukkainen, M. and Kurhila, J. 2012. Multi-faceted support for MOOC in programming. In proceedings of the 13th Annual Conference on Information Technology Education (pp. 171-176). ACM.

Vihavainen, A., Airaksinen, J. and Watson, C. 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 (pp. 19-26). ACM.

Winslow, L.E. 1996. Programming pedagogy—a psychological overview. ACM SIGCSE Bulletin, 28(3), pp.17-22.

Virtanen, A.T., Lahtinen, E. and Järvinen, H.M. 2005. VIP, a visual interpreter for learning introductory programming with C++. In proceedings of The Fifth Koli Calling Conference on Computer Science Education (pp. 125-130).

Yin, H., Moghadam, J. and Fox, A. 2015. Clustering student programming assignments to multiply instructor leverage. In proceedings of the Second (2015) ACM Conference on Learning@ Scale (pp. 367-372). ACM.

Paper 1

Kaila, E., Rajala, T., Laakso, M.-J. and Salakoski, T. 2009.

Effects, Experiences and Feedback from Studies of a Program Visualization Tool.

Informatics in Education. 8 (1), 17 – 34. Vilnius University.

Informatics in Education, 2009, Vol. 8, No. 1, 17–34
© 2009 *Institute of Mathematics and Informatics, Vilnius*

Effects, Experiences and Feedback from Studies of a Program Visualization Tool

Erkki KAILA, Teemu RAJALA, Mikko-Jussi LAAKSO, Tapio SALAKOSKI
TUCS, Turku Centre for Computer Science, University of Turku
Joukahaisenkatu 3–5 B, 6th floor, FI-20520 Turku, Finland
e-mail: {ertaka, temira, milaak, sala}@utu.fi

Received: December 2008

Abstract. Program visualization (PV) is potentially a useful method for teaching programming basics to novice programmers. However, there are very few studies on the effects of PV. We have developed a PV tool called ViLLE at the University of Turku. In this paper, multiple studies on the effects of the tool are presented. In addition, new qualitative data about students' feedback of using the tool is presented. Both, the results of our studies and the feedback indicate that ViLLE can be used effectively in teaching basic programming concepts to novice programmers.

Keywords: program visualization, programming, education, effects,experiences, student feedback.

1. Introduction

Programming is one of the main objectives in computer science studies. However, according to several studies (see, e.g., McCracken *et al.* (2001) and Ala-Mutka (2005)) students have significant problems in learning the very basics of programming. Because of scarce teaching resources and large group sizes students often get inadequate personal instruction. Thus, there is clearly a need for instructional tools that support students' independent learning.

Visualization systems use various graphical means in concretizing abstract programming and algorithmic problems. According to Wiggins (1998), the purpose of visualization is to help the user understand *what* a program does, *why* it does it, *how* it works, and *what* the end result is. Hence, visualization systems can supposedly help students to understand programs better, thus improving the learning results. There are several studies on the effects of different algorithm visualization systems (see, e.g., Hundhausen *et al.* (2002) and Laakso *et al.* (2008a)), but very few on program visualization.

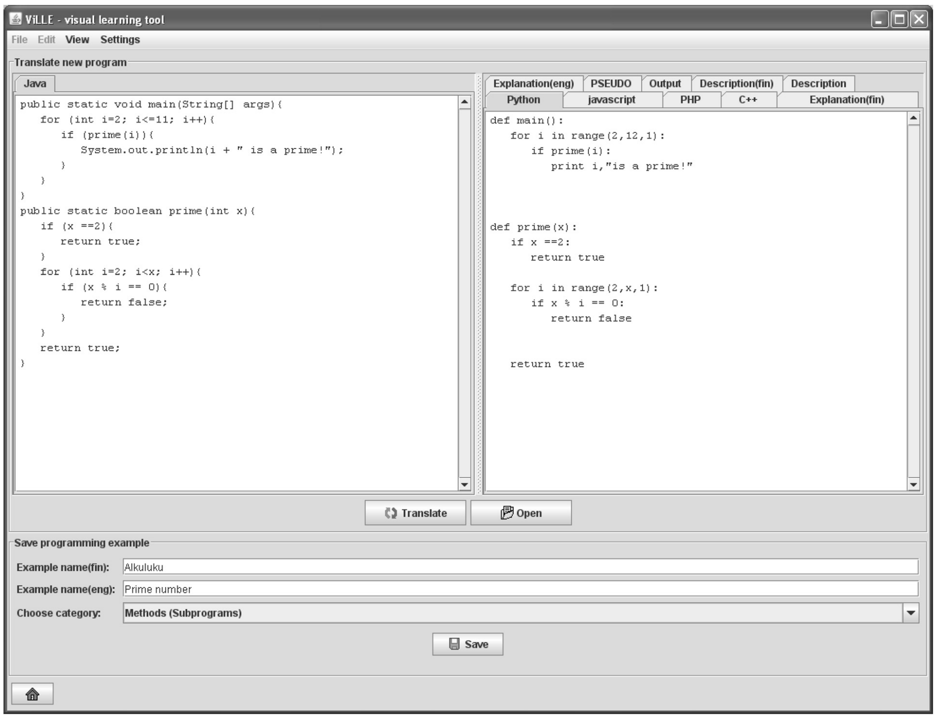
We have developed a program visualization tool called ViLLE at the University of Turku, Finland. The purpose of developing the tool was to find out if program visualization can indeed help students in learning to program. ViLLE has been tested with different kind of setups taking student competence levels and backgrounds into account. In this paper we describe the tool itself for both in teacher's and student's point of view and present the results of our studies on the effects of ViLLE so far. Moreover, we present new quantitative results, based on the student feedback about using the tool. Finally, the future of ViLLE is discussed.

2. ViLLE

ViLLE is a program visualization tool, developed at the University of Turku. Its main purpose is to illustrate the changes in the programs states during the execution with various graphical and textual means. ViLLE supports multiple programming languages and has built-in editors for defining and editing syntaxes, examples and questions. With the export function the defined examples can be published as an independent collection, distributable in web or other media. By using the TRAKLA server (see Malmi *et al.* (2004)), ViLLE's automatically assessed exercises can be easily integrated as a part of a programming course.

2.1. *Teacher's Point of View*

From a teacher's point of view, the suitability for different kinds of courses can be seen as the number one feature of ViLLE. Programming languages, examples and questions are all customizable with the built-in editors. Hence, it should be relatively easy to integrate



```
public static void main(String[] args){
   for (int i=2; i<=11; i++){
      if (prime(i)){
         System.out.println(i + " is a prime!");
      }
   }
}
public static boolean prime(int x){
   if (x ==2){
      return true;
   }
   for (int i=2; i<x; i++){
      if (x % i == 0){
         return false;
      }
   }
   return true;
}
```

```
def main():
   for i in range(2,12,1):
      if prime(i):
         print i,"is a prime!"



def prime(x):
   if x ==2:
      return true

   for i in range(2,x,1):
      if x % i == 0:
         return false


   return true
```

Example name(fin): Alkuluku
Example name(eng): Prime number
Choose category: Methods (Subprograms)

Fig. 1. Example editor in ViLLE.

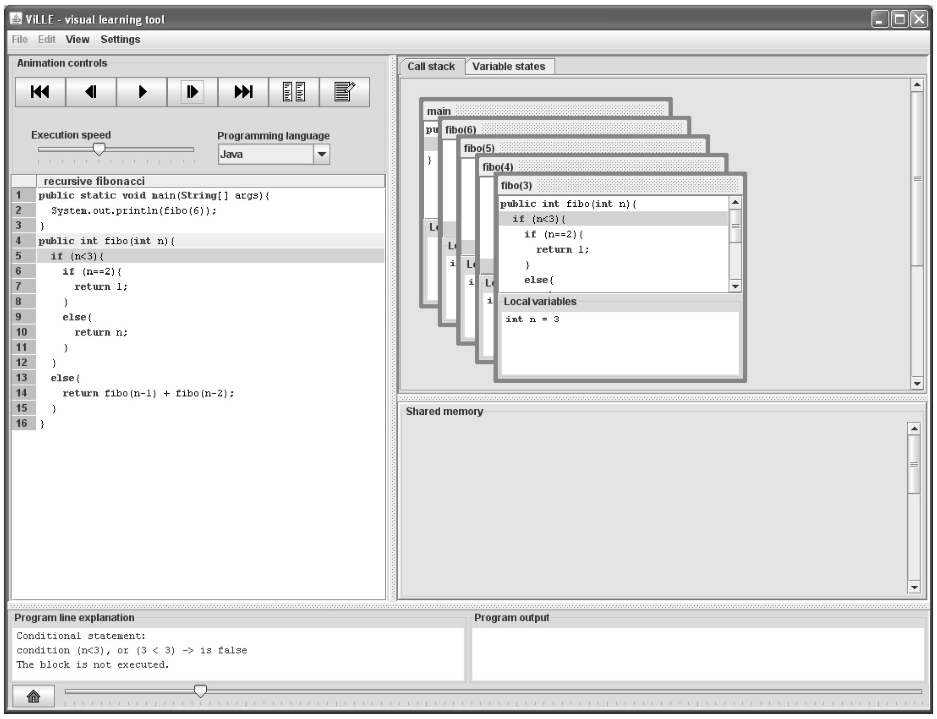
ViLLE into almost any programming course. Moreover, the flexibility gives teachers a chance to fulfil their own teaching philosophy, without a need to adjust to tool's limitations.

2.1.1. *Example Editor*

Examples are divided into categories. Teacher can create new examples and edit the existing ones by using the built-in example editor (see Fig. 1). The examples are written in Java, and ViLLE automatically translates the program code to all defined languages. Moreover, the visualizations of examples are automatically generated, as well as explanations on each program line. The list of supported Java features is limited, mainly, because all the programs should be easily translatable to other defined languages, and because the tool is directed for novice programmers. Furthermore, the visualization of more complex features (e.g., GUI components) can get quite tricky.

2.1.2. *Question Editor*

To further engage the learners to visualization, teacher can create questions about the example programs. Currently multiple choice questions and graphical array questions are supported. The questions are attached to program execution with a built-in question editor (see Fig. 2), and are automatically launched when desired point of program is reached.



```
public static void main(String[] args){
  System.out.println(fibo(6));
}
public int fibo(int n){
  if (n<3){
    if (n==2){
      return 1;
    }
    else{
      return n;
    }
  }
  else{
    return fibo(n-1) + fibo(n-2);
  }
}
```

```
int n = 3
```

```
Conditional statement:
condition (n<3), or (3 < 3) -> is false
The block is not executed.
```

Fig. 2. Question editor.

The TRAKLA2 server can be used with ViLLE to keep the score of answered questions and correct answers.

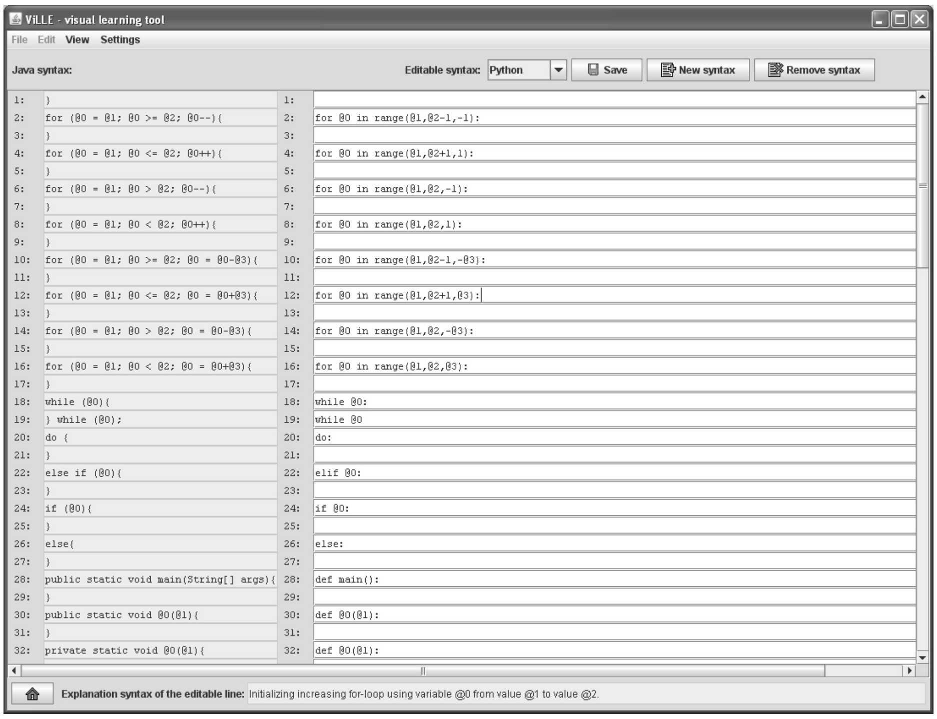
2.1.3. *Syntax Editor*

Teacher can add or edit syntaxes with the built-in syntax editor (Fig. 3). The editable syntax is presented on the right hand side of the window, and the correspondent Java syntax on the left hand side. Additionally, an explanation of the edited syntax line is presented in the bottom window. All defined syntaxes must have matching lines with the Java syntax to ensure the flow of execution and the consistency of questions.

2.2. *Student's Point of View*

ViLLE's key features from the student's point of view can be divided into following categories:

Visualizing the program execution: The execution of the example program is visualized line by line (see Fig. 4). Currently and previously executed lines are highlighted, and the variable values, program line explanation and the output of the program are presented in their own frames. All subprograms, and their return values, are presented in the call order in their own frames in the call stack. Moreover, all global variables (namely arrays) are presented graphically in their own area.



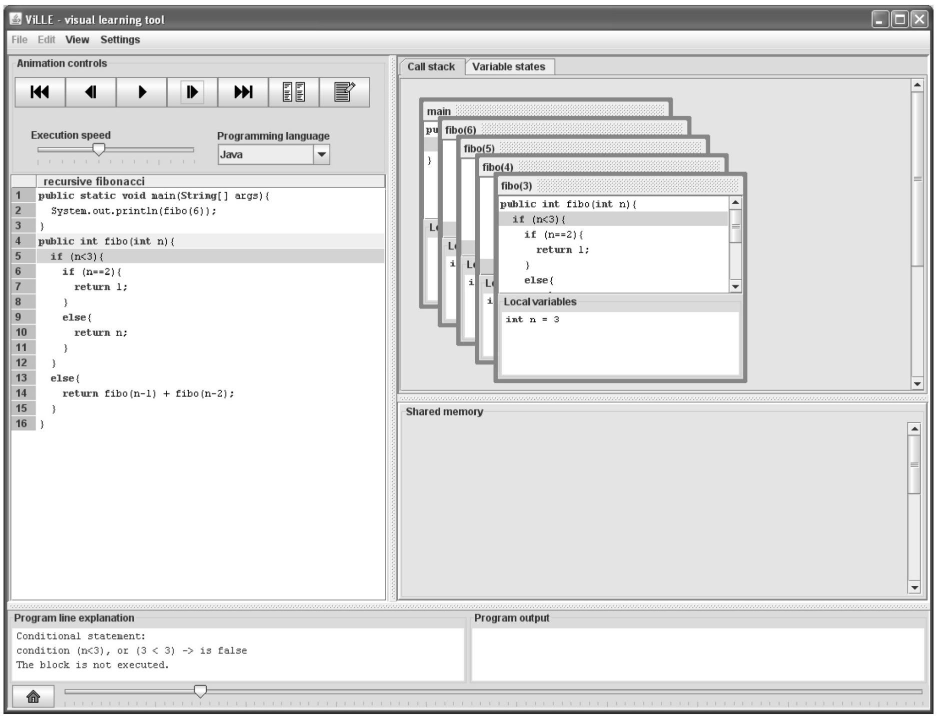
Java syntax:

Editable syntax: Python

	Java syntax		Python
1:	`}`	1:	
2:	`for (@0 = @1; @0 >= @2; @0--){`	2:	`for @0 in range(@1,@2-1,-1):`
3:	`}`	3:	
4:	`for (@0 = @1; @0 <= @2; @0++){`	4:	`for @0 in range(@1,@2+1,1):`
5:	`}`	5:	
6:	`for (@0 = @1; @0 > @2; @0--){`	6:	`for @0 in range(@1,@2,-1):`
7:	`}`	7:	
8:	`for (@0 = @1; @0 < @2; @0++){`	8:	`for @0 in range(@1,@2,1):`
9:	`}`	9:	
10:	`for (@0 = @1; @0 >= @2; @0 = @0-@3){`	10:	`for @0 in range(@1,@2-1,-@3):`
11:	`}`	11:	
12:	`for (@0 = @1; @0 <= @2; @0 = @0+@3){`	12:	`for @0 in range(@1,@2+1,@3):`
13:	`}`	13:	
14:	`for (@0 = @1; @0 > @2; @0 = @0-@3){`	14:	`for @0 in range(@1,@2,-@3):`
15:	`}`	15:	
16:	`for (@0 = @1; @0 < @2; @0 = @0+@3){`	16:	`for @0 in range(@1,@2,@3):`
17:	`}`	17:	
18:	`while (@0){`	18:	`while @0:`
19:	`} while (@0);`	19:	`while @0`
20:	`do {`	20:	`do:`
21:	`}`	21:	
22:	`else if (@0){`	22:	`elif @0:`
23:	`}`	23:	
24:	`if (@0){`	24:	`if @0:`
25:	`}`	25:	
26:	`else{`	26:	`else:`
27:	`}`	27:	
28:	`public static void main(String[] args){`	28:	`def main():`
29:	`}`	29:	
30:	`public static void @0(@1){`	30:	`def @0(@1):`
31:	`}`	31:	
32:	`private static void @0(@1){`	32:	`def @0(@1):`

Explanation syntax of the editable line: Initializing increasing for-loop using variable @0 from value @1 to value @2.

Fig. 3. Syntax editor.



recursive fibonacci

```
public static void main(String[] args){
  System.out.println(fibo(6));
}
public int fibo(int n){
  if (n<3){
    if (n==2){
      return 1;
    }
    else{
      return n;
    }
  }
  else{
    return fibo(n-1) + fibo(n-2);
  }
}
```

```
public int fibo(int n){
  if (n<3){
    if (n==2){
      return 1;
    }
    else{
```

```
int n = 3
```

```
Conditional statement:
condition (n<3), or (3 < 3) -> is false
The block is not executed.
```

Fig. 4. Visualization view.

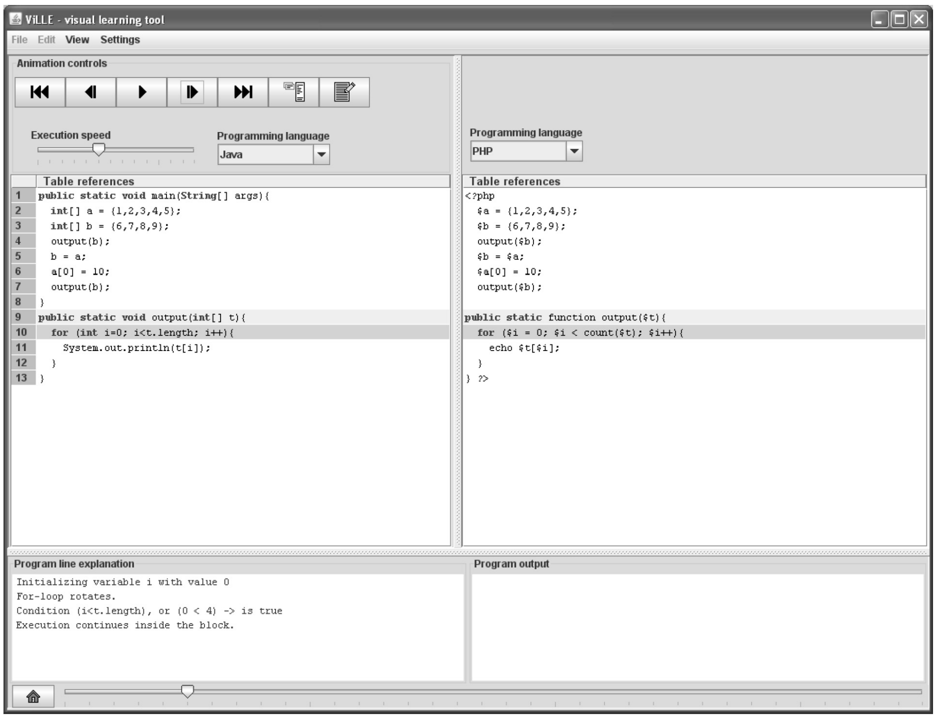
Language independency: The visualization of the execution works similarly, regardless of the chosen programming language. Programming language can be changed anytime during the visualization. In addition, ViLLE has a parallel mode (see Fig. 5) where execution can be viewed in two different languages simultaneously.

Visualization controls: Controls are flexible – the user can move one step at a time, both forwards and backwards in the execution of a program. In addition, the program can be executed continuously in adjustable speed. The execution slider at the bottom of the window can be used to move directly to a desired state of execution. The slider also has a secondary function: the number of steps can be used to determine the complexity of the program, and with suitable examples to compare the complexities of algorithms.

Interaction: Besides answering questions (see Fig. 6) the students can edit the program code in the visualization view. The changes in program code can be visualized instantly. However, since the editing must be done in Java, the feature can't naturally be utilized in all courses.

2.3. *Automatic Assessment of Exercises*

Examples made with ViLLE can be transferred to a collection in a web (see Fig. 7) by using the TRAKLA2 server. The server keeps score on students' logins and submissions, and makes it possible to set opening and expiration dates for example rounds. Individual



Execution speed

Programming language: Java

Programming language: PHP

```
public static void main(String[] args){
  int[] a = {1,2,3,4,5};
  int[] b = {6,7,8,9};
  output(b);
  b = a;
  a[0] = 10;
  output(b);
}
public static void output(int[] t){
  for (int i=0; i<t.length; i++){
    System.out.println(t[i]);
  }
}
```

```
<?php
  $a = {1,2,3,4,5};
  $b = {6,7,8,9};
  output($b);
  $b = $a;
  $a[0] = 10;
  output($b);

public static function output($t){
  for ($i = 0; $i < count($t); $i++){
    echo $t[$i];
  }
} ?>
```

```
Initializing variable i with value 0
For-loop rotates.
Condition (i<t.length), or (0 < 4) -> is true
Execution continues inside the block.
```

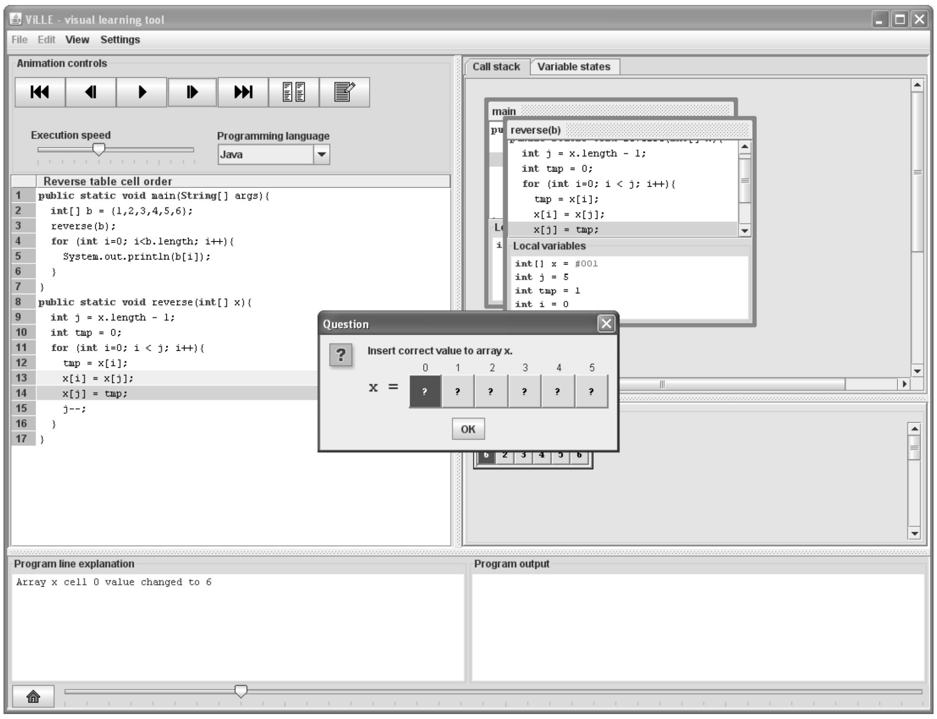
Fig. 5. Parallel view.

examples can be retaken an unlimited number of times. Additionally, the teacher can set the minimum number of points required for each round to pass the course. Web-based ViLLE exercises are nowadays in use in most universities in Finland, and the student feedback (see Subsection 3.4) has been mostly positive: it seems that ViLLE has a positive role in enhancing the reading and tracing skills of novice programmers.

3. The Studies on ViLLE

3.1. *The Effectiveness of ViLLE*

The effectiveness of ViLLE was studied at the University of Turku, Finland, in a course called ”Introduction to information technology”. There were 72 students participating in the study. We tried to find the answer to two research questions: 1) “Is ViLLE useful in learning basic programming concepts?” and 2) “Is there any difference in learning results when earlier programming experience is taken into account?”. The null-hypotheses were that ViLLE is not useful in learning to program, and that the effect is same for both novices and more experienced students. No programming was taught before the study, taking place in the third week of the course. However, a lecture was arranged before the study where the programming language and the tool used were introduced. A link



Reverse table cell order

```
public static void main(String[] args){
  int[] b = {1,2,3,4,5,6};
  reverse(b);
  for (int i=0; i<b.length; i++){
    System.out.println(b[i]);
  }
}
public static void reverse(int[] x){
  int j = x.length - 1;
  int tmp = 0;
  for (int i=0; i < j; i++){
    tmp = x[i];
    x[i] = x[j];
    x[j] = tmp;
    j--;
  }
}
```

Local variables

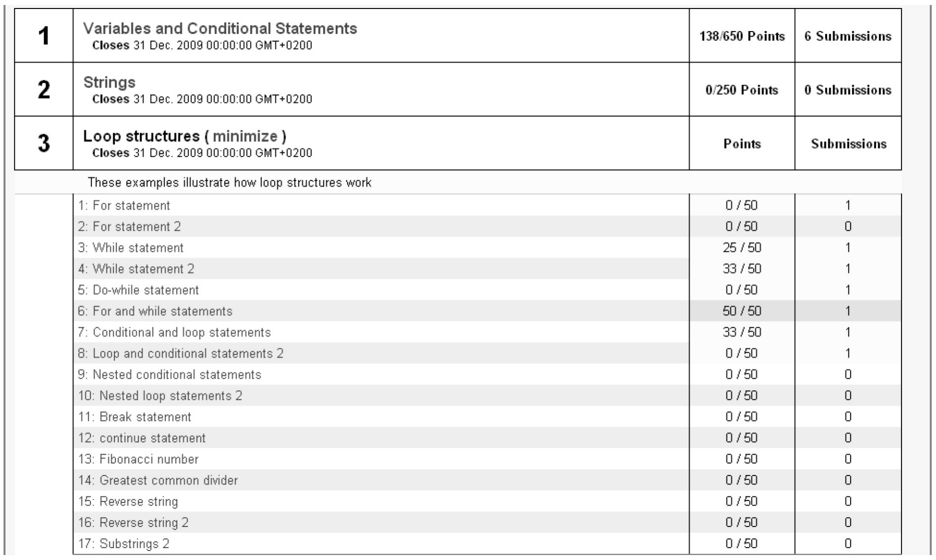
```
int[] x = #001
int j = 5
int tmp = 1
int i = 0
```

Insert correct value to array x.

	0	1	2	3	4	5
x =	?	?	?	?	?	?

```
Array x cell 0 value changed to 6
```

Fig. 6. Array question in visualization view.



1	Variables and Conditional Statements **Closes** 31 Dec. 2009 00:00:00 GMT+0200	138/650 Points	6 Submissions
2	Strings **Closes** 31 Dec. 2009 00:00:00 GMT+0200	0/250 Points	0 Submissions
3	**Loop structures** (minimize) **Closes** 31 Dec. 2009 00:00:00 GMT+0200	Points	Submissions
	These examples illustrate how loop structures work		
	1: For statement	0 / 50	1
	2: For statement 2	0 / 50	0
	3: While statement	25 / 50	1
	4: While statement 2	33 / 50	1
	5: Do-while statement	0 / 50	1
	6: For and while statements	50 / 50	1
	7: Conditional and loop statements	33 / 50	1
	8: Loop and conditional statements 2	0 / 50	1
	9: Nested conditional statements	0 / 50	0
	10: Nested loop statements 2	0 / 50	0
	11: Break statement	0 / 50	0
	12: continue statement	0 / 50	0
	13: Fibonacci number	0 / 50	0
	14: Greatest common divider	0 / 50	0
	15: Reverse string	0 / 50	0
	16: Reverse string 2	0 / 50	0
	17: Substrings 2	0 / 50	0

Fig. 7. ViLLE exercises in TRAKLA2 environment.

to ViLLE was added to course homepage in the second week so that the students could familiarize themselves with the tool before the study.

The students were randomly divided into two groups: there were 32 ($N = 32$) students in the *treatment* group and 40 ($N = 40$) in the *control* group. At the beginning of the two hour session students first took a pre-test that lasted for 15 minutes. The pre-test included three program reading exercises where students were asked to write down the output of each program. After the pre-test, students rehearsed programming concepts presented in the test with a web-based tutorial for 45 minutes. The treatment group could also visualize the program code examples presented in the tutorial with the ViLLE tool. After the tutorial session students had 30 minutes to answer to a post-test. The post-test had the same three questions as the pre-test with two more demanding questions, in which the students had to write a program implementing a given task, and write down the output of a recursive program.

Pre- and post-test exercises were analyzed in the scale of zero to ten. Thus, the maximum points of the pre- and post-test were 30 and 50. Table 1 presents the pre-test scores for both groups, including point averages, standard deviations (in parentheses) and p-values calculated with a two-tailed t-test.

No statistically significant differences were found in any of the questions. In absolute scale the treatment group performed better in questions Q2 and Q3, and the control group in question Q1.

The scores from the post-test are presented in Table 2. Corresponding questions from the pre-test are shown in square brackets beside the post-test question names. Total points averages are presented both for the shared questions (questions that were the same in the pre- and post-test) and for all questions. Additionally, the point difference between the shared questions in the pre- and post-test is shown.

The comparison between the shared questions shows that the control group performed better in question PQ1 and the treatment group in questions PQ2 and PQ3. One reason for the smaller difference in the scores of the treatment group in PQ1 is the relatively high scores they got from the pre-test. In any case, the differences are so small that the null-hypothesis can not be rejected.

Table 3 displays a comparison between the pre- and post-test scores inside the groups.

As seen in the table, both groups performed statistically significantly better in the post-test than in the pre-test (p-value < 0.01). Based on this, it seems that it is possible

Table 1
Pre-test scores

Question	Control Group ($N = 40$)	Treatment Group ($N = 32$)	p-value
Q1	5.20 (2.67)	6.19 (2.46)	0.111
Q2	2.70 (3.53)	2.13 (3.53)	0.494
Q3	2.68 (4.15)	2.09 (3.88)	0.546
Total	10.58 (8.64)	10.41 (7.18)	0.930

Table 2

Post-test scores

Question	Control group ($N = 40$)	Treatment Group ($N = 32$)	p-value
PQ1 [Q1]	6.30 (2.81)	6.13 (2.69)	0.790
PQ2 [Q2]	5.10 (4.35)	5.50 (4.50)	0.704
PQ3	6.28 (3.75)	5.88 (3.75)	0.654
PQ4 [Q3]	6.15 (4.56)	6.50 (4.42)	0.744
PQ5	7.05 (3.78)	6.69 (4.08)	0.698
Total (shared)	17.55 (9.08)	18.13 (8.81)	0.788
Total (all)	30.88 (15.20)	30.69 (15.08)	0.959
Difference PQ1 – Q1	1.10 (2.60)	– 0.06 (2.81)	0.073
Difference PQ2 – Q2	2.40 (3.30)	3.38 (4.02)	0.262
Difference PQ4 – Q3	3.48 (4.81)	4.41 (4.53)	0.405
Total difference	6.98 (6.81)	7.72 (6.76)	0.646

Table 3

Pre- and post-test scores

Scores	Control Group	Treatment Group
Pre-test	10.58	10.41
Post-test	17.55	18.13
Total difference	6.98	7.72
p-value	0.000	0.000

to efficiently study basic programming concepts independently in these kinds of tutorial sessions.

The other research problem was the effect of earlier programming experience on learning results. When the results were analyzed, both groups were divided into two: students with no earlier programming experience (NPE) and students with some earlier programming experience (SPE). The pre-test scores with earlier programming experience taken into account are presented in Table 4.

The figures in the table show that students with some earlier programming experience got statistically significantly better scores in both groups. The corresponding results from the post-test are presented in Table 5.

As we can see from the table, statistically significant difference between NPE and SPE remained in the post-test in the control group. However, the difference vanished in the treatment group (p-values 0.212 and 0.151). Thus, it seems that ViLLE is especially useful for novice programmers. Because of the relatively short exposure to the tool, the result can be seen as significant. Cronbach alpha-values calculated for pre- and post-test questions (pre-test $\alpha = 0.667$ and post-test $\alpha = 0.831$) indicate high reliability. This study is presented in more detail in (Rajala *et al.*, 2008).

Table 4
The effect of earlier programming experience on pre-test scores

Question	Control Group			Treatment Group		
	NPE ($N = 23$)	SPE ($N = 17$)	p-value	NPE ($N = 20$)	SPE ($N = 12$)	p-value
Q1	4.17 (2.33)	6.59 (2.53)	0.003	5.60 (2.11)	7.17 (2.76)	0.107
Q2	1.22 (1.78)	4.71 (4.31)	0.005	1.00 (2.22)	4.00 (4.51)	0.049
Q3	1.00 (2.86)	4.94 (4.62)	0.005	1.65 (3.62)	2.83 (4.34)	0.414
Total	6.39 (4.68)	16.24 (9.63)	0.001	8.25 (5.44)	14.00 (8.48)	0.051

Table 5
The effect of earlier programming experience on post-test scores

Question	Control Group			Treatment Group		
	NPE ($N = 23$)	SPE ($N = 17$)	p-value	NPE ($N = 20$)	SPE ($N = 12$)	p-value
PQ1 [Q1]	5.74 (2.78)	7.06 (2.75)	0.144	5.90 (2.86)	6.50 (2.43)	0.533
PQ2 [Q2]	3.39 (3.97)	7.41 (3.81)	0.003	4.70 (4.58)	6.83 (4.22)	0.199
PQ3	5.30 (4.06)	7.59 (2.90)	0.045	5.05 (3.65)	7.25 (3.65)	0.109
PQ4 [Q3]	5.22 (4.83)	7.41 (3.94)	0.122	6.00 (4.71)	7.33 (3.94)	0.418
PQ5	6.09 (4.09)	8.35 (2.96)	0.049	6.05 (4.20)	7.75 (3.82)	0.261
Total (shared)	14.35 (8.27)	21.88 (8.51)	0.008	16.60 (9.29)	20.67 (7.64)	0.212
Total (all)	25.74 (14.44)	37.82 (13.68)	0.011	27.70 (15.49)	35.67 (13.53)	0.151
Difference PQ1 – Q1	1.57 (2.48)	0.47 (2.70)	0.198	0.30 (2.62)	− 0.67 (3.11)	0.354
Difference PQ2 – Q2	2.17 (3.07)	2.71 (3.65)	0.620	3.70 (4.38)	2.83 (3.46)	0.564
Difference PQ4 – Q3	4.22 (4.73)	2.47 (4.87)	0.261	4.35 (4.73)	4.50 (4.38)	0.929
Total difference	7.96 (5.80)	5.65 (7.98)	0.295	8.35 (7.98)	6.67 (4.14)	0.439

3.2. *The Effect of Engagement on Learning*

Naps *et al.* (2002) presented a taxonomy of learner engagement with a visualization tool. The taxonomy defines six levels of engagement:

1. *No Viewing*: There is no visualization tool in use.
2. *Viewing*: User follows the visualization passively. Despite of its name, all forms of observation belong to this level. The user can control the flow of the visualization but is not allowed to actively participate in any way.
3. *Responding*: User answers questions presented during the visualization.
4. *Changing*: User changes the visualization, for example by modifying the program code or algorithm used in the visualization.
5. *Constructing*: User actively participates in the construction of visualization, for example by writing program code.
6. *Presenting*: User presents the visualization and evaluates it together with the audience.

The study presented in Subsection 3.1 was extended so that a third group was formed which used ViLLE in a lower level of the engagement taxonomy. Hence, there were three groups: no viewing ($N = 40$), viewing ($N = 65$) and responding ($N = 32$). The questions were removed from the version of ViLLE used by the viewing group. The purpose of the setup was to confirm the hypothesis of Naps *et al.* (2002), which states that visualizations can have an effect on learning only if they are used in engagement level three or higher.

Statistical differences between the groups in the pre- and post-test were calculated with one-way ANOVA-test, and are presented in Table 6.

As seen in the table, there were no statistically significant differences between the groups. Students in the viewing group also improved their results statistically significantly during the session (the p-value of pre- and post-test difference inside the group < 0.01).

The differences in learning results between the novices and more experienced programmers inside the groups were examined next. The results are shown in Table 7.

There was a statistically significant difference in all groups between the NPE and SPE. As discovered in the previous section, the difference in the responding group vanished in the post-test. However, the difference remained in the viewing group (p-value < 0.001). Thus we can conclude that for the ViLLE to be useful, it should be used in an engagement level higher than viewing. To confirm the result, a post-hoc Student-Newman-Keuls test

Table 6
Statistical differences between the groups

	No viewing ($N = 40$)	Viewing ($N = 65$)	Responding ($N = 32$)	p-value
Pre-test total	10.58 (8.64)	10.85 (8.89)	10.41 (7.18)	0.968
Post-test total (shared)	17.55 (9.08)	17.94 (9.53)	18.13 (8.81)	0.963
Total difference	6.97 (6.81)	7.09 (6.63)	7.72 (6.76)	0.881

Table 7
The effect of previous programming experience on pre- and post-test score

	No viewing			Viewing			Responding		
	NPE ($N = 23$)	SPE ($N = 17$)	p-value	NPE ($N = 36$)	SPE ($N = 29$)	p-value	NPE ($N = 20$)	SPE ($N = 12$)	p-value
Pre-test total	6.39	16.24	*0.001*	6.81	15.86	*0.000*	8.25	14.00	*0.051*
Post-test total (shared)	14.35	21.88	*0.008*	13.72	23.17	*0.000*	16.60	20.67	*0.212*
Total difference	7.96	5.65	*0.320*	6.92	7.31	*0.812*	8.35	6.67	*0.439*
Post-test total (all)	25.74	37.82	*0.011*	24.72	39.90	*0.000*	27.70	35.67	*0.151*

Table 8
Pre-test scores divided into homogenous subsets

Group	*N*	Subset for alpha = 0.05	
		1	2
No Viewing NPE	23	6.39	
Viewing NPE	36	6.81	
Responding NPE	20	8.25	
Responding SPE	12		14.00
Viewing SPE	29		15.86
No Viewing SPE	17		16.24

Table 9
Post-test scores divided into homogenous subsets

Group	*N*	Subset for alpha = 0.05	
		1	2
Viewing NPE	36	13.72	
No viewing NPE	23	14.35	
Responding NPE	20	16.60	16.60
Responding SPE	12		20.67
No viewing SPE	17		21.88
Viewing SPE	29		23.17

was used. The analysis forms two homogenous subsets from the pre-test scores so that the novices and more experience programmers belong to different subsets (Table 8).

When the post-test scores were analyzed similarly, the results show that the novices in the responding group caught up all the SPE groups (Table 9).

The analysis confirms the previous finding that ViLLE is especially useful for novices learning programming basics, but only if used in higher levels of engagement. Similar learning results are not achieved if visualizations are viewed passively.

3.3. *The Impact of Prior Experience on Learning*

We also wanted to study what kind of effects prior experience on using a visualization tool has on the learning results. Presumably, students who have familiarized themselves with the usage of a tool can focus better on the subject taught because the cognitive load of using the tool is lighter. The study was organized in two instances of a high school programming course. The only difference between the courses was that in the latter course students were familiarized with the use of ViLLE and its features. A session similar to the ones presented in previous sections was arranged, although the questions in pre- and post-test were partly modified (for example recursion was thought to be concept

Table 10

Math and CS grades (scale 4...10)

Group	Math	CS
Control Group	6.75 (1.60)	7.94 (1.09)
Treatment Group	7.67 (2.25)	8.57 (1.62)

Table 11

Pre- and post-test scores

	Pre-test total	Post-test total (shared)	Post-test total (all)
Control Group ($N = 17$)	7.12	12.59	16.94
Treatment Group ($N = 7$)	9.43	19.57	26.43
p-value	*0.515*	*0.047*	*0.046*

too complex for high school students). Students who hadn't used ViLLE before belonged to control group ($N = 17$) while students with prior experience on ViLLE formed the treatment group ($N = 7$). Though the groups were quite small, results were statistically significant.

To confirm the equality of the groups, participants' earlier math and CS grades were analyzed. The averages and standard deviations are presented in Table 10.

There were no statistically significant differences in grades between the groups. The absolute differences are less than one point.

Pre- and post-test scores are presented in Table 11.

There was no statistically significant difference between the groups in the pre-test scores, but there is a difference in the post-test. The statistically significant difference can be found both in the shared (same questions in pre- and post-test) questions and all questions. Based on the results we can conclude that prior knowledge of ViLLE clearly improves learning results. Furthermore the results support our earlier findings that ViLLE is useful in learning basic programming skills: both groups got statistically significantly better results from the post-test in comparison to the pre-test. The results are presented in more detail in (Laakso *et al.*, 2008b).

3.4. *Student Feedback*

In addition to quantitative tests we wanted to find out what students think about the tool. The opinions were gathered from students participating in a course called 'Introduction to Information Technology' at the University of Turku. ViLLE was an integral part of the course as all the assignments were done with it. 114 students answered to the questionnaire consisting of three parts: general questions about the tool, how useful the system is when learning new programming concepts, and opinions about the features of the tool.

In the first section six statements about system were presented, and students were asked to evaluate those in a scale of one to seven (1: completely disagree, 7: completely agree). Based on the answers, the students seem to think that the tool is quite suitable for teaching programming (average of all answers 5.64), that it is fairly easy to use (avg. 5.49) and that it helps in understanding the basic programming concepts (avg. 5.41).

In the second section the students were asked to evaluate the usefulness of ViLLE in different areas related to programming (see Table 12). Based on the answers, the students found ViLLE as an useful tool for teaching all the basic concepts – arrays were the only concept which got an average less than five (avg. 4.73). Based on students' comments, this is probably because of the usability issues in answering the array questions.

In the third section the students were asked to evaluate the usefulness of different features of ViLLE (see Table 13). All features – except the visualization of programs in different languages – got an average of five or higher. The features the students found most useful were the visualization of variable states and the automatic assessment of exercises (averages 5.90 and 5.80). The worse average in "visualization in multiple languages" is probably due to a fact, that the students didn't use the feature. For that matter, the feature can be found most useful when the students already know a programming language, and a different language is taught.

Table 12

Usefulness of ViLLE in understanding programming concepts

How useful did you find ViLLE in understanding the following concepts? (scale 1–7)	
Variables and assignments	5.41 (1.37)
Conditional statements	5.52 (1.15)
Loops	5.61 (1.19)
Boolean statements	5.38 (1.23)
Subprogram definitions	5.38 (1.25)
Subprogram calls	5.34 (1.32)
Subprogram parameters	5.24 (1.33)
Arrays	4.73 (1.58)

Table 13

Usefulness of ViLLE's individual features

How useful did you find the following features in ViLLE (scale 1–7)?	
Visualization of programs in different languages	4.93 (1.46)
Visualization of subprograms with call stack	5.35 (1.24)
Visualization of variable states	5.90 (1.17)
Explanation of program code line	5.40 (1.49)
Questions about program execution	5.50 (1.24)
Automatic assessment of exercises	5.80 (1.28)

Additionally, the students were asked to evaluate the number of ViLLE exercises in the course and the number of questions in the exercises in the scale of one to seven (1 – too few, 7 – too many). Based on the answers, the amounts were quite suitable (exercises avg. 4.48 and questions avg. 4.13).

Moreover, the students had a possibility to give additional comments about the tool. Most of the feedback was quite positive. However, some criticism and thoughts about improving the tool were also given. Some of the positive things mentioned were:

- "In my opinion, ViLLE exercises were more effective than lectures."
- "I found ViLLE really useful: because of the ViLLE exercises, I didn't practically need the lecture handouts at all to learn programming."
- "With ViLLE I was able to pick up programming far better than with lectures."
- "There were a lot of different exercises and it was easy to do them wherever and whenever I felt like it."

The negative comments were mostly related to the functionality of the user interface:

- "ViLLE is great when it functions properly; however, the GUI needs some improvement: e.g., handling arrays is illogical and difficult."
- "It would be handy if you could see the execution history when answering the questions:"
- "All the essential things won't fit on the screen simultaneously. Why do you need to login to ViLLE?"
- "It's irritating that you can't scroll the window when the question appears."

Moreover, some of the students would like to have more conventional teaching, instead or alongside ViLLE exercises:

- "I'd rather attend the traditional computer lab sessions, where there would be some kind of help available on request."
- "ViLLE was a good tool for studying; however, more training in small groups would be highly appreciated."
- "ViLLE is ok for learning the basics, but personally I learned more during the lectures."

All in all, the students seemed to have quite positive image about ViLLE's usefulness, though some improvements were also wished for. In conclusion, the students would prefer a course, where ViLLE exercises are integrated with more traditional teaching methods. The reported problems related to the user interface will be taken into account in the further development of the tool – e.g., the answering to array questions should nowadays work more logically than in the version, which the opinions were gathered about.

4. The Future of the Tool

ViLLE is nowadays in use in basic programming courses in most universities in Finland, and in addition, it will be mobilized at least in Australia during this spring. In future, we plan to develop the tool further based on the user opinions and experiences, and further research the effects of the tool when used in programming teaching. Our goal is to add more features to support the higher levels of engagement taxonomy. Moreover, exercise templates are developed to enable the randomization of the exercise parameters (i.e., variable types and values etc.); this makes redoing the exercises more meaningful. Additionally, different kinds of exercise types are under development, including code sorting exercises, where program code lines are randomly shuffled and the students are supposed to sort them in an order which implements the given task or algorithm.

5. Conclusions

Based on the presented results, experiences and feedback we can conclude the following:

- By using ViLLE the novices can improve their learning results significantly, even when the tool is used quite briefly…
- … but only if the tool is used in the higher level of engagement. The mere viewing of visualization doesn't seem to have the same effect. This supports the hypothesis presented by Naps *et al.* (2002).
- Moreover, to ensure the learning results, the students must be familiarized with the tool beforehand, hence reducing the cognitive load of learning to use the tool.
- Most of the students think that ViLLE is beneficial when learning the basic programming skills. However, some of the students seem to think that the best way to use such tool is to integrate it with the more conventional forms of teaching.
- ViLLE gives students a chance to learn and practice the very basics of programming independently. These basics can't normally be covered on the lectures as thoroughly as required by some students.
- The focus of ViLLE is in program code reading and comprehension skills. However, Lopez *et al.* (2008) state that the tracing skills correlate with students performance on code writing tasks.
- From a teacher's point of view, ViLLE's key feature is the flexibility, and most importantly the support for almost any imperative programming language.

All in all the experiences seem quite encouraging so far: the results and the students' feedback both confirm the assumption that ViLLE can be used effectively in teaching the basic programming skills to novices. As we all know – unless we have already forgotten – the first steps in programming really are quite difficult. Therefore there seems to be a demand for such system, both now and in the future.

References

Ala-Mutka, K. (2005). Ohjelmoinnin opetuksen ongelmia ja ratkaisuja. In *Tekniikan opetuksen symposium*, 20–21.10.2005. Helsinki University of Technology.
`http://www.dipoli.tkk.fi/ok/p/reflektori/verkkojulkaisu/index.php?p=verkkojulkaisu`

Hundhausen, C.D., Douglas, S.A. and Stasko, J.D. (2002). A Meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, **13**, 259–290.

Laakso, M.-J., Myller, N. and Korhonen, A. (2008a). Comparing learning performance of students using algorithm visualizations collaboratively on different engagement levels. *Journal of Educational Technology and Society* (to appear).

Laakso, M.-J., Rajala, T., Kaila, E. and Salakoski, T. (2008b). The impact of prior experience in using a visualization tool on learning to program. In *Proceedings of CELDA 2008*, Freiburg, Germany.

Lopez, M., Whalley, J., Robbins, P. and Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *Proceeding of the Fourth International Workshop on Computing Education Research*, September 6–7, 2008, Sydney, Australia, 101–112.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O. and Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, **3**(2), 267–288.

McCracken, M., Almstrum, V., Diaz, D., Gudzial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, **33**(4), 125–140.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, **35**(2), 131–152.

Rajala, T., Laakso, M.-J., Kaila, E. and Salakoski, T. (2008). Effectiveness of program visualization: a case study with the ViLLE tool. *Journal of Information Technology Education*: *Innovations in Practice*, **7**, 15–32.

Wiggins, M. (1998). An overview of program visualization tools and systems. In *Proceedings of the 36th Annual Southeast Regional Conference*, 194–200.

E. Kaila has written his master's thesis on program visualization in programming learning at University of Turku. His research interests include program visualization systems and IT education.

T. Rajala is a PhD student at University of Turku. He received his master's degree from the same university in 2007. His research focuses on visualization of programs and algorithmic problem solving.

M.-J. Laakso is currently working as a researcher at University of Turku. He received his MSc (computer science) in 2003. His research interest covers program and algorithm visualization, learning environments, computer aided and automatic assessment in computer science education.

T. Salakoski is a professor of computer science at University of Turku, where he received his PhD in 1997. His main research focus has been in methodology development using machine learning and other intelligent techniques. He is leading a multidisciplinary research group studying various task domains, including problems related to human learning and computing education research.

Programų vizualizavimo priemonė: poveikis mokymui, patirtis ir studentų atsiliepimai

Erkki KAILA, Teemu RAJALA, Mikko-Jussi LAAKSO, Tapio SALAKOSKI

Laikoma, jog pradedančiuosius mokant programavimo pagrindų naudinga taikyti programų vizualizavimo metodą. Tačiau tėra labai nedaug tyrimų apie šio metodo poveikį. Turku universitete buvo sukurta programų vizualizavimo priemonė, pavadinta ViLLE. Straipsnyje supažindinama su šios priemonės poveikio tyrimais, pateikiami nauji kokybiniai duomenys: studentų, naudojusių šią priemonę, atsiliepimai. Ir mūsų tyrimų rezultatai, ir atsiliepimai rodo, kad programos vizualizavimo priemonė ViLLE gali būti veiksmingai naudojama pradedančiuosius programuotojus mokant pagrindinių programavimo sąvokų.

Paper 2

Kaila, E., Rajala, T., Laakso, M.J. and Salakoski, T., 2010.

Effects of Course-Long Use of a Program Visualization Tool.

In Proceedings of the Twelfth Australasian Conference on Computing Education (ACE 2010) - Volume 103, 97 – 106. Australian Computing Society, Inc.

Effects of Course-Long Use of a Program Visualization Tool

Erkki Kaila, Teemu Rajala, Mikko-Jussi Laakso, Tapio Salakoski
University of Turku and Turku Centre for Computer Science (TUCS)
Informaatioteknologian laitos, 20014 Turun yliopisto, Finland
{ertaka, temira, milaak, sala}@utu.fi

Abstract

We studied the course-long use of a program visualization tool called ViLLE in high school in Finland. The study was conducted in three consecutive instances of the first programming course. In the first two instances of the course, the students did not utilize ViLLE – except for a short session – while in the last instance students did ViLLE exercises throughout the whole course. The students who used ViLLE got significantly better results from the course's final exam. This supports our hypothesis that program visualization can be an effective method in teaching programming, and indicates that we should continue developing program visualization methods to further enhance learning.

Keywords: Programming education, program visualization, long-term effects of visualization, integrating visualization into a course.

1 Introduction

The difficulties of novice programmers in introductory courses are discussed in various studies (see McCracken et al. 2001, Lister et al. 2004, Tenenberg et al. 2005). In addition to the cognitive difficulties in learning, problems often arise from lack of resources in teaching. When courses are quite large, teachers or lecturers cannot pay enough attention to single students. Moreover, the time available for introductory courses is usually quite brief, forcing the teachers to advance to more complicated topics without a possibility to ensure the level of learning. Hence, the students often lack even the basic reading and writing skills after these introductory courses.

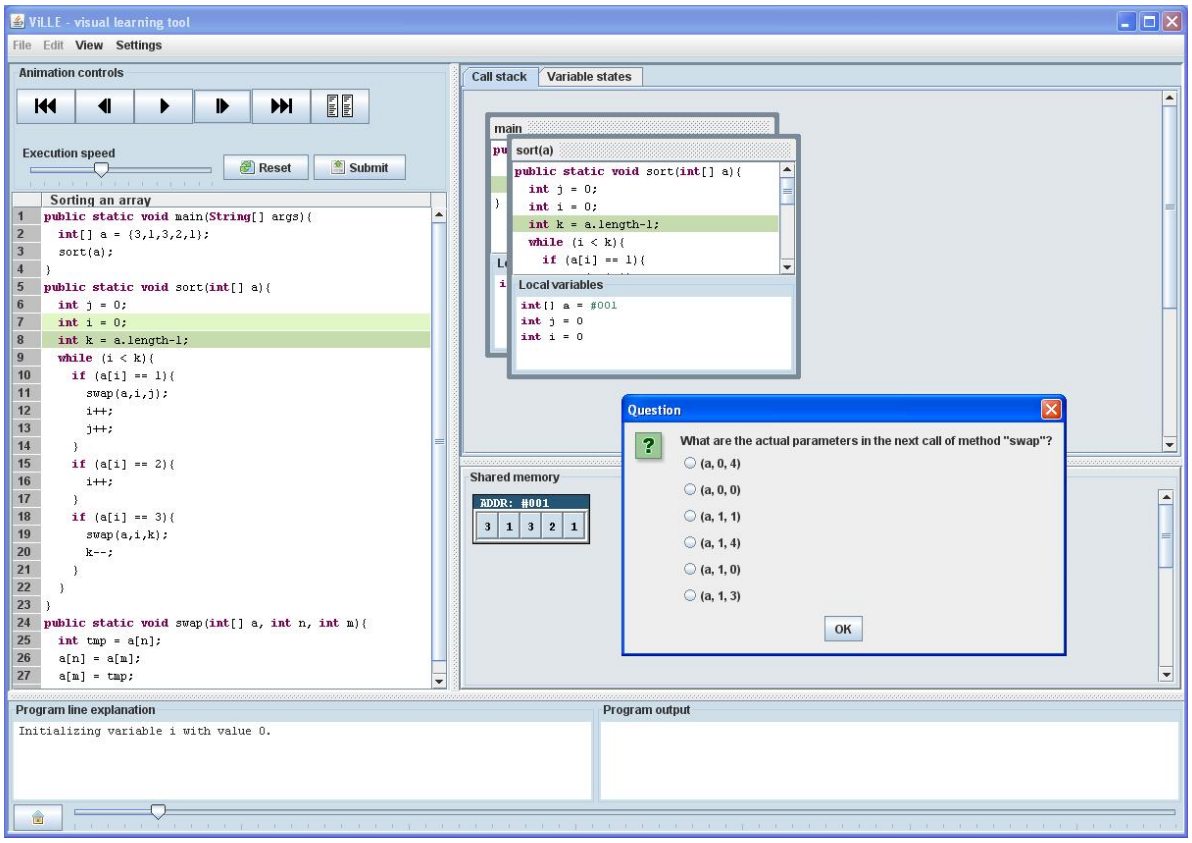
Program visualization (or program animation) is a method of illustrating the program behaviour in different states of the execution. Program visualization (PV) systems typically visualize the execution and program states (such as variable values, expression evaluation or object and function dependencies) with various graphical and textual components. Program visualization tools aim to enhance learning in two distinctive ways: firstly, a teacher can use such systems in lectures to illustrate the changes in program states during the execution of programs, and secondly, students can use tools independently to rehearse topics they found difficult. The potential benefit of independent use is the possibility to focus on more advanced topics in the lectures, and let the students rehearse the basics on their own. However, this scenario presents a couple of issues: firstly, how to encourage the students to use the tool independently, and secondly, how to pick a tool that can be used to produce learning results?

1.1 Integrating a visualization tool into a programming course

In order to gain the possible benefits from PV tools in the course they should be properly introduced and integrated into a course. Introduction should be done with enough care. In our earlier research (Laakso et al. 2008) we found out that the students who were properly familiarized with a visualization tool beforehand gained statistically significantly better learning results than the other students. Though it is likely that the familiarization will eventually happen during the course long use, the additional cognitive load of learning to use the tool itself may frustrate the students (even more since the students at the introductory phase usually have a lot of different systems and tools they need to adapt to). Hence, the introduction should be more than a mere link at the course web site and the graphical notation of the tool should be carefully explained to the students.

It is important to have a plan of what kind of role a visualization tool will have in a course. The tool can be used to replace or complement the traditional methods of teaching, however, some things that need to be considered are:

- What kind of topics will the tool be used for? Will it be fully integrated into all areas of teaching or used specifically to teach only some topics?
- What kind of "reward" will be awarded to students for using the tool? Lehtonen (2005) notes that getting in a top list of the best achievements can be enough motivation for students. Additional or alternative rewards can be, for example, extra points for the course grade or replacement of some of the course exercises with use of the tool. Though the improved learning should be a reward on itself, it often does not seem to be enough.
- How hard will it be for the teacher to mobilize the tool, and what kind of advantages (other than improved learning results) could be gained? Could the tool be used, for example, for the automatic assessment of exercises, what kind of statistics is available and so on?



Sorting an array

```
public static void main(String[] args){
  int[] a = {3,1,3,2,1};
  sort(a);
}
public static void sort(int[] a){
  int j = 0;
  int i = 0;
  int k = a.length-1;
  while (i < k){
    if (a[i] == 1){
      swap(a,i,j);
      i++;
      j++;
    }
    if (a[i] == 2){
      i++;
    }
    if (a[i] == 3){
      swap(a,i,k);
      k--;
    }
  }
}
public static void swap(int[] a, int n, int m){
  int tmp = a[n];
  a[n] = a[m];
  a[m] = tmp;
```

main

sort(a)

```
public static void sort(int[] a){
  int j = 0;
  int i = 0;
  int k = a.length-1;
  while (i < k){
    if (a[i] == 1){
```

Local variables

```
int[] a = #001
int j = 0
int i = 0
```

Shared memory

```
ADDR: #001
| 3 | 1 | 3 | 2 | 1 |
```

What are the actual parameters in the next call of method "swap"?

- (a, 0, 4)
- (a, 0, 0)
- (a, 1, 1)
- (a, 1, 4)
- (a, 1, 0)
- (a, 1, 3)

Program line explanation

```
Initializing variable i with value 0.
```

Program output

Figure 1: the visualization view in ViLLE

In this article we present the results of a course long study of the program visualization tool called ViLLE. Earlier results from two-hour controlled studies on the effectiveness of ViLLE were encouraging. We consider it is important to also study the effects of the tool while used throughout programming courses.

2 Related Work

Over the years, various program visualization tools have been developed, but there are very few studies on the effectiveness of such tools, and even less on their long-term effects on learning performance. In general, algorithm and program visualization tools are studied with qualitative methods; there are very few studies where the results are based on quantitative data.

Crescenzi and Nocentini (2007) describe a two-year experiment of integrating an algorithm visualization (AV) tool called ALVIE into a programming course. The course also included a textbook associated with the AV tool. They report no quantitative results, but student feedback was mainly positive.

Laakso et al. (2005) report an experiment in introducing a web-based AV system called TRAKLA2 (Malmi et al. 2004) to students at two universities. They report that the tool seems to be very useful as the midterm pass rates raised significantly after the tool was introduced, and it promoted student activity in the course. The student feedback was also very positive.

Carlisle et al. (2005) used a flowchart based visualization tool called RAPTOR in introductory computing courses and found out that students using the tool got better results from two questions in the final exam when compared with the earlier course where the tool was not utilized. However, they also got worse results from the third question in the exam

Ben-Bassat Levy et al. (2003) utilized a program visualization tool Jeliot 2000 (an earlier version of Jeliot3 (Moreno et al. 2004)) in a year-long course. They conclude that since the control group had so little room for improvement, the quantitative data only shows improvement in the animation group (the group that used the tool).

Stasko (1997) studied the effects of the visualization tool called Samba, which students used in building their own visualizations. No quantitative data was presented, but student feedback indicated that students thought that they understood algorithms well. Moreover, according to an informal measure by the writer, in the final exam the students answered nearly perfectly to questions regarding the algorithms they had built earlier.

Brown and Sedgewick (1984) describe the use of Balsa AV tool in an introductory course and in an algorithm and data structures course. It was found out that the students understood algorithms in less time compared with traditional teaching methods.

3 ViLLE

In this section we present the tool and describe some earlier results from the studies about ViLLE's effectiveness.

3.1 Features

ViLLE is a program visualization tool which illustrates the changes in a program's states with various graphical and textual components. The tool combines visualizations with the automatic assessment of program code reading exercises. Using ViLLE, a teacher can create example sets and include questions (multiple choice and graphical array questions in the version were used in this research) which the students answer during the execution of the example programs. As we have previously stated (Kaila et al., 2009a), mere visualizations are not enough to have a substantial effect on learning. Providing questions engages students to perform at the active level of learning.

Some of ViLLE's key features are:

- Possibility to visualize programs with various programming languages. Several syntaxes are included and the teacher can define new syntaxes with the built-in syntax editor.
- Flexible controls of animation: students can execute the program step by step or continuously in adjustable speed. Stepping backwards is also possible – a feature that is not commonly included in such tools.
- A built-in question editor which allows the teacher to attach questions in the chosen points in the example program. The current version includes also coding and code shuffling exercises; however, these were not in use in the version used in this research.
- Flexibility and ease of use both for teachers and students.

Complete list of ViLLE's features can be found at tool's website, at *http://ville.cs.utu.fi*.

3.2 Motivation for developing ViLLE

Although other program visualization systems have been developed (see section 2), they seem to lack some features we found necessary for such system:

- The tool should have support for various programming languages instead of focusing on one particular language.
- Visualizations should always be combined with exercises to actively involve students.
- The tool should support the creation of new exercises and visualizations. The process should be as straightforward as possible to encourage teachers to develop their own materials.

These features were the basis for the development of ViLLE.

3.3 Previous Studies

We have previously conducted series of experiments about the effectiveness of ViLLE in various studies. In this section we present the most significant results of those studies.

The effectiveness of ViLLE was studied at University of Turku, in a course called "Introduction to information technology". Seventy two students participated in the study, randomly divided into two groups: 32 in the treatment group and 40 in the control group. At the beginning of the session all students took a pre-test, which measured their earlier programming knowledge. After the pre-test the students rehearsed the programming concepts with a web-based tutorial. In addition, the treatment group could visualize the examples in a ViLLE tutorial.. After this, a post-test was arranged to measure the learning effects. Both groups performed significantly better at the post-test, but no significant differences between the groups were found. However, when students' earlier programming experience was taken into account, we found out that the difference between novices and experienced ones in the treatment group narrowed down and statistically disappeared during the session. Hence, we concluded that ViLLE is especially useful for novice programmers. The study is presented in detail in Rajala et al. (2008).

Following this, the study was extended so that there were three groups, each in a different engagement level (see Naps et al. 2002). The aim of the study was to find out whether we could confirm the hypothesis, which states that visualization tools can produce learning results only if used in active levels, i.e. the mere tracking of the visualizations is not enough. In addition to the earlier research, a third group was formed. This group (the viewing group, N=62) could control the animation of program executions, but had no other ways to actively take part in them. The responding group (N=32) had a full version of ViLLE in use, with multiple choice questions about examples activated. The results showed, that the difference between experienced and the novice programmers remained in the viewing group (as well as in the tutorial-only group, i.e. the no-viewing group). This confirms the earlier findings that ViLLE is especially useful for novices, but only if used in engagement level higher than viewing. The study is presented in detail in Kaila et al. (2009a).

Another study investigated whether prior experience in using a tool has an effect on learning results. The study was conducted in two instances of an introductory programming course in high school, with a session similar to the ones mentioned earlier in this section. The only difference between groups was that the (treatment group (selected randomly out of the two) was familiarized with ViLLE before the session was arranged. There were no statistically significant differences between the groups in the pre-test. However, in the post-test a statistically significant difference was found: the treatment group outperformed the control group in shared questions (i.e. the questions that were similar in pre- and post-tests) and in all questions. Based on the results we can conclude that the earlier experience of the tool has a significant effect on learning results, a fact that should be taken into account when arranging such studies. The study is presented in detail in Laakso et al. (2008).

Next we arranged a study to find out if the learning effects of ViLLE can be further enhanced by using the tool in collaboration with other student. The students were randomly divided into two groups: the students in treatment group (N=62) used ViLLE in collaboration with other students, while the students in the control group (N=50) used it alone. The results confirmed our previous findings: all students' succeeded statistically significantly better in the post-test, thus proving that it is possible to teach basic programming concepts effectively with ViLLE, even in such short time (45 minutes). While there were no differences between the groups in the pre-test, a significant result was found in the post-test. This supports the earlier findings of Laakso et al. (2009), that collaboration is highly beneficial when using a visualization tool. The study is presented in detail in Rajala et al. (2009).

In addition to all quantitative tests proving that ViLLE is beneficial in learning, we also wanted to find out what students think about the tool. We gathered students' opinions at the course "Introduction to information technology" at our university. The students had used ViLLE throughout the course, with more than 10,000 exercises taken. 114 students answered the questionnaire, consisting of questions about the features, usability and usefulness of the tool. Based on the answers, most of the students think that ViLLE is beneficial when learning the basic programming concepts. Some students even thought that they learned better using ViLLE than with traditional learning methods (exercises, demonstrations and lectures). However, some students think that the best way to use such tool would be to integrate it with more traditional forms of teaching. The study is presented in detail in Kaila et al. (2009b).

In this study, in contrast to our earlier studies, we wanted to examine the effects of a visualization tool when used throughout a programming course. Since the long-term effectiveness of program visualization is rarely studied, and since we have found encouraging results from two-hour sessions, we find this very relevant topic of interest.

4 Research

The effectiveness of ViLLE was studied in three consecutive high school programming courses. In the first two course students used ViLLE only in a two hour lab session at the beginning of the course. In the third course students used ViLLE throughout the course. The idea was to find out what kind of effects program visualization has on learning programming when used throughout the course.

4.1 Method

The experiment was a between subject design with a pre-test and final exam results (dependant variable). We had one between-subject factor (independent variable): the amount of usage of ViLLE.

4.2 Materials

All course material was distributed via the course learning management system Moodle (Dougiamas & Taylor 2003). The material included background theory, code examples, and coding exercises for each topic taught. Python was used as the teaching language.

In the third week of the courses, a computer lab session was arranged where students rehearsed programming concepts with a programming tutorial. In the tutorial some programming concepts (variables, selection, loops, and methods) were shortly explained and students solved included ViLLE exercises. At the beginning of the course, before the lab session, students were familiarized with the syntax of Python, variables, user input and selection statements. The session begun with students answering a pre-test (see Appendix A), which included three questions about selection, loops and methods, and ended with a similar post-test. The results from these sessions are presented in Laakso et al. (2008).

The final exam was divided into two parts (see Appendices B and C). The first part included five code reading exercises which were solved on paper. The exercises included selection statements, explaining the meaning of each line in a program, and what the program does, the number of short questions about syntactical features in Python, and explaining what certain code fragments do and writing down their output. When students returned the first part, they got three more coding exercises which were solved with computers. The coding exercises included writing a program that counts the average of an array, a program that asks a number from user and counts factorial of the number, and program that reads a file, and counts the line and word count inside the file. Each of the five questions in the first part of the exam was worth 6 points, while three coding exercises in the second part were worth 10 points each. Thus, the maximum score in the final exam was 60 points.

4.3 Participants

The participants were students from the high school of Kupittaa, a school that focuses on teaching information technology and media. There were two instances of the course in the fall 2007: in the first course there were 12 students and in the second there were 8 students, totalling 20 students in the non-ViLLE group; however, since there were two students, who took the course independently (i.e. they did not participate in lectures or other teaching, but merely took the final exam), and three students who dropped out from the course, the total number in non-ViLLE group becomes 15 (N=15). In the 2008 course the student count was 7 (the ViLLE group), because one student dropped out. The selection of the ViLLE group was randomized. The course was the first programming course in the curriculum for each student.

4.4 Procedure

All lessons in the programming courses were held in a computer lab. Each topic was first introduced by the teacher. After the introduction students solved exercises with the help of code examples and background theory. The teacher also did live-coding by explaining all his actions during the coding process to further clarify things.

The courses were identical in the following aspects:

- Both were taught by the same teacher.
- All materials were identical.
- Final exams were identical.

Hence, the only difference was that in the 2008 course students did visualization exercises with ViLLE throughout the course. When each programming concept was introduced, a number of ViLLE-exercises covering the concept were prepared for the students in the ViLLE group. The non-ViLLE group had similar program code examples included in the web material, but they could not visualize them. Hence, the time used in studying different programming topics was the same for both groups.

The starting level of students was measured with the pre-test (see section 4.2). Student performance was measured by comparing the final exam scores between the ViLLE and non-ViLLE groups.

The results between the groups were analyzed with a two-tailed t-test. Levene's test was used to calculate variances for all statistics to determine if the data holds equal or non-equal variances.

5 Results

In this section we present the results on the research question "is there any difference in learning when ViLLE is used throughout the course".

In addition, correlations between previous math and computer science (CS) knowledge and success in reading and writing exercises are presented.

5.1 Previous knowledge

Participants' previous programming knowledge was determined by reviewing their earlier success in CS and math studies, and by comparing their scores in the pre-test of the computer lab session. CS and math grades are presented in Table 1. The table includes averages (on the scale of 4 to 10), standard deviations (in parentheses) and p-values of the t-test between groups.

	Non-ViLLE group (N=9)	**ViLLE group**	**p-value**
Math	6.56 (1.82)	7.53 (1.35), N=5	0.278
CS	7.83 (1.15)	8.54 (1.10), N=6	0.254

Table 1: Participants' Math and CS grades

In math studies the grades are presented for the three first advanced math courses, and naturally for only those students who took these courses (because of this the table does not include all students). Similarly, CS studies reported in the table include two first introductory courses. The courses were the same for all the students. As seen as the table, no statistically significant differences were found between the groups.

The results of the pre-test in the computer lab session were analyzed similarly (see Table 2). The ViLLE group performed better in the absolute scale (10.83 vs. 7.79 on the scale of 0 to 30), but no statistically significant differences were found (p-value 0.430). In both groups there was on student who could not participate in the pre-test, but did the final exam (thus the N-values). Based on the results we can conclude that there were no significant differences between the groups' programming knowledge before taking the course.

	Non-ViLLE group (N=14)	**ViLLE group (N=6)**	**p-value**
Q1	5.35 (2.53)	6.50 (3.89)	0.529
Q2	1.64 (2.50)	2.16 (3.92)	0.721
Q2	0.79 (0.80)	2.17 (3.87)	0.425
Total	7.79 (4.34)	10.83 (8.38)	0.430

Table 2: Pre-test results

5.2 Final exam scores

Learning effects on the course were measured with final exam results. The exam consisted of five code reading (Q1-Q5) and three code writing (QW1 – QW3) exercises (see section 4.2 for details). Results are presented in Table 3.

	Non-ViLLE group (N=15)	**ViLLE group (N=7)**	**p-value**
Q1	3.73 (2.12)	4.43 (2.15)	0.484
Q2	3.13 (1.19)	4.14 (1.46)	0.099
Q3	2.67 (1.59)	4.07 (1.74)	0.075
Q4	4.73 (1.44)	6.00 (0.00)	*0.004*
Q5	1.27 (1.67)	3.00 (2.16)	0.052
Reading total	15.53 (5.58)	21.64 (6.34)	*0.033*
QW1	3.20 (2.24)	6.29 (3.73)	0.077
QW2	4.67 (2.92)	8.00 (2.52)	*0.017*
QW3	2.93 (3.17)	3.86 (4.26)	0.574
Writing total	10.80 (7.03)	18.14 (9.06)	*0.050*
Exam Total	26.33 (11.84)	39.79 (14.11)	*0.030*

Table 3: Final exam results

As seen on Table 3, ViLLE group performed statistically significantly better in Q4 (t (20) = -2,301), in total of reading questions (t (20) = -2,294), total of writing questions (t (20) = -2,084) and in combined total score (t (20) = -2,339); moreover, the difference in QW2 was almost significant. Since the Q4 was a question about function calls, ViLLE's visualization of functions seemed to be especially helpful.

In absolute scale and by looking the p-values of different questions, it seems that only in Q1 and QW3 groups performed somewhat similarly. Q1 was about if-statement, a topic already presented in the course before the lab session. That might be one reason why students got good results from a similar assignment in the pre-test (see table 2). QW3 was about reading a file, which is not possible to visualize in ViLLE, and thus the topic was taught similarly to both groups. In all other questions the trend was favouring the ViLLE group.

5.3 Correlations between math and CS grades and total scores

Finally, we wanted to examine correlations between students' grades and the scores of reading and writing sections on the final exam. Math and CS grades had medium correlation with reading exercises (0.699, $p < 0.01$ and 0.557, $p < 0.05$, respectively) and strong correlation with writing exercises (0.781, $p < 0.01$ and 0.725, $p < 0.01$). Since the CS course grades were from introductory courses, and did not include any programming, we suggest that the grades actually depict more about students' motivation than actual programming knowledge. The pre-test results (see section 5.1) support this.The quite low averages indicate that students' programming skills were rather poor before taking the course.

Another interesting aspect is the strong correlation between reading and writing exercise points in the final exam (0.776, $p < 0.001$). This seems to be in line with the findings by Lopez et al. (2008) which state that there is a strong correlation between students' tracing and writing skills.

6 Discussion

The results show that the group that used ViLLE throughout the course got statistically significantly better results from the final exam in total scores, reading total, writing total, and in question 4 in the reading exercises. Hence, ViLLE seems to be beneficial in tracing program code. Since the question 4 was about functions, ViLLE seems the most helpful when tracing the execution between the main program and subprograms. Remarkably, all students in the ViLLE group got full points from the question. Moreover, there was a trend favoring the treatment group in absolute scale in all questions, although in Q1 and QW3 the groups performed quite similarly. Based on the results, ViLLE also seems to help in developing program writing skills. Cronbach's alpha value 0.866 calculated for the questions in the final exam indicate high reliability.

It is hard to estimate how other factors influenced the results. Although the materials in the courses were identical and the courses were taught by the same teacher, the teacher might have had more experience in teaching the course in the spring. Moreover, the group sizes were quite small, and it is impossible to isolate all the factors that influence the learning in a six-week course. One confounding factor might be the randomization of groups; although the ViLLE group was selected randomly, the courses were taught in different semesters.

However, since the difference between the non-ViLLE and ViLLE groups was so substantial in total points, and all our previous studies indicate that ViLLE has a positive effect on learning, we can conclude that ViLLE is beneficial when learning basic programming skills.

It seems that the biggest advantage of ViLLE is the possibility to offer lots of program code reading exercises to students. Students get direct feedback on their success from the tool. This is especially helpful in mass courses where the teacher does not have enough time to address each student individually, but also applicable in courses with fewer students; the teacher can not always help all students that need guidance, no matter how few there are. The feedback given by the tool helps students understand some concepts by themselves and the teacher is freed from having to repeatedly answer basic questions. Even if the feedback from a tool is not as good as personal feedback from the teacher, students still get useful information on the events occurring in the program. Moreover, they can retake examples as many times as they see necessary – thus, the tool can be very helpful in ensuring that all the concepts taught are sufficiently rehearsed.

7 Conclusions

We conducted a study on the effects of course-long use of a program visualization tool in a high school in Finland. The results indicate that the students who used the tool throughout the course got significantly better results on their final exam. The difference was significant in all total scores, which indicates that program visualization is a highly beneficial method of teaching basic programming concepts. This confirms our earlier findings from controlled two hour sessions. Moreover, we could confirm our other earlier finding that ViLLE is especially useful when tracing the execution of function calls. In future we plan to retake a similar study at the university level, and study the effects of new features (mainly code sorting and coding exercises) of ViLLE.

8 References

Ben-Bassat Levy, R., Ben-Ari, M. and Uronen, P.A. (2003): The Jeliot 2000 program animation system. *Computers & Education*, **40**(1): 15-21.

Brown, M.H. and Sedgewick, R. (1984): Progress report: Brown university instructional computing laboratory. *SIGCSE Bull.* **16**(1): 91-101.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., and Hadfield, S. M. (2005): RAPTOR: a visual programming environment for teaching algorithmic problem solving. *Proc. of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 176-180.

Crescenzi, P. and Nocentini, C. (2007): Fully integrating algorithm visualization into a cs2 course: a two-year experience. *Proc. of the 12th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Dundee, Scotland, June 25 - 27, 2007). ITiCSE '07. ACM, New York, NY, 296-300.

Dougiamas, M. and Taylor, P. (2003): Moodle: Using Learning Communities to Create an Open Source Course Management System. *Proc. of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Lassner & C. McNaught (Eds.), Chesapeake, VA: AACE, 171-178.

Kaila, E., Laakso, M.-J., Rajala, T. & Salakoski, T. (2009a): Evaluation of Learner Engagement in Program Visualization. To appear in *12th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2009)*, November 22 – 24, 2009, St. Thomas, US Virgin Islands.

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.

Kannusmäki, O., A. Moreno, N. Myller and E. Sutinen. (2004): What a Novice Wants: Students Using Program Visualization in Distance Programming Courses. *Proc. of the Third Program Visualization Workshop*: 126-133.

Laakso, M.-J., Rajala, T., Kaila, E. and Salakoski, T. (2008): The Impact of Prior Experience In Using A Visualization Tool On Learning To Program. *Proceedings of CELDA 2008, Freiburg, Germany*, 129-136.

Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A. and Malmi, L. (2005): Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, **4**(1): 49-68.

Laakso, M.-J., Myller, N. and Korhonen, A. (2009): Comparing learning performance of students using algorithm visualizations collaboratively on different engagement levels. *Journal of Educational Technology & Society*, **12**(2): 267-282.

Lehtonen, T. (2005): Javala – Addictive E-Learning of the Java Programming Language. *Proc. of Kolin Kolistelut / Koli Calling – Fifth Annual Baltic Conference on Computer Science Education*, Joensuu, Finland, 41-48.

Lister, R., Adams, S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O., Simon, B. and Thomas, L. (2004): A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bulletin*, **36**(4): 119-150.

Lopez, M., Whalley, J., Robbins, P. and Lister, R. (2008): Relationships between reading, tracing and writing skills in introductory programming. *Proc. of the fourth international workshop on Computing education research*, Sydney, Australia, 101-112.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., and Silvasti, P. (2004): Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. *Informatics in Education*, **3**(2): 267-288.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001): A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students. *ACM SIGCSE Bulletin*, **33**(4): 125-140.

Moreno, A., Myller, N., Sutinen, E. and Ben-Ari, M. (2004): Visualizing Programs with Jeliot 3. *Proc. of the Working Conference on Advanced Visual Interfaces (AVI 2004)*, Gallipoli (Lecce), Italy. ACM Press, New York: 373-380.

Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Velásquez-Iturbide, J. Á. (2002): Exploring the Role of Visualization and Engagement in Computer Science Education. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, **35**(2): 131-152.

Petre, M. (1995): Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. *Communications of the ACM*, **38**(6): 33-44.

Rajala, T., Laakso, M.-J., Kaila, E. and Salakoski, T. (2008): Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education: Innovations in Practice.* 7: IIP 15-32.

Rajala, T., Kaila, E., Laakso, M.-J. & Salakoski, T. (2009): Effects of Collaboration in Program Visualization. *Technology Enhanced Learning Conference 2009 (TELearn 2009),* Taipei, Taiwan.

Stasko, J.T. (1997). Using student-built algorithm animations as learning aids. *SIGCSE Bull.* **29**(1): 25-29.

Tenenberg, J., Fincher, S., Blaha, K., Bouvier, D., Chen, T.-Y., Chinn, D., Cooper, S., Eckerdal, A., Johnson, H., McCartney, R. and Monge, A. (2005): Students designing software: a multi-national, multi-institutional study. *Informatics in Education*, **4**(1): 143-162.

Appendix A: Pre-test

The pre-test consisted of three assignments: in each assignment a block of program code was presented, and the students were asked to either write down the output (assignments 2 and 3) or variable values in different states of execution (assignment1).

Assignment 1:

```
a = 2
b = 6
c = 4
d = 8
if c < a:
       c = c * 2
       d = d - 1

if b < d:
       a = a + 3
       b = 1

else:
       d = d - 1

if c > a:
       c = c / 2
else:
       a = a + 1
       b = a + 1
       c = a + b
       d = c + b
```

Assignment 2:

```
x = 1
y = 4
while (x+y) < 39:
       if x < y:
              x = x + 5
       else:
              y = y + 9
              x = x + 1
       x = x + 1
       print x
       print y
```

Assignment 3:

```
def main():
    n = 6
    while n > 2:
        print n,"FU:",fu(n)
        n = n-1

    print n,"FU:",fu(n)

def fu(x):
    if x+1 > 5:
        return 2+x

    return x+14
```

Appendix B: Final exam, part 1

Q1: What are the values of variables a, b, c and d after the execution of this program?

```
a = 0
b = 5
c = 7
d = 10

if c < d and a < c:
   a = a + b
   b = c - 1
   d = d - a
if d == b or d == c:
   d = d * 2
   c = c / 2
   b = b * a
elif not(a > b and d > c):
   a = a + 2
   b = b + 3
   d = d – c
else:
   a = b
   b = c
   c = d
   d = d * 2
```

Q2: Add comments to each line of the following program. Finally, explain what the whole program does.

```
list = [0,0,0,0,0]
i = 0
while i < len(list):
   try:
      list[i] = input('Give a number: ')
      i = i + 1
   except Exception:
      print 'That is not a number!'
result = list[0]
for j in list:
   if j < result:
      result = j
print result
```

Q3:

a) What command is used to include functions square() and pow() from module *math*?

b) What is the value of variable a after this statement is executed: `a = len(‘exam’) + (2 * len(‘answer’))?`

c) What is the output of this program?

```
list = [‘one’,’five’,’eight’,’seven’,’ten’]
print list[4] + list[1]
```

d) What is the output of this program?

```
s = {‘three’ : 3,’five’ : 5, ‘four’ : 4, ‘two’ : 2}
print s[‘four’] 0 s[‘two’]
```

e) What is the output of this program?

```
i = 2
while i < 10:
   if i % 2 == 0:
      i = i + 1
      continue
   print i
   i = i + 1
```

f) What is the output of this program?

```
i = 2
while i < 10:
      if i % 2 == 0:
          i = i + 1
          break
      print i
      i = i + 1
```

Q4:

a) What does this function do?

```
def first(nmbr1,nmbr2):
      if nmbr1 <= nmbr2:
          return nmbr1 ** nmbr2
      else:
          return nmbr2 ** nmbr1
```

b) What does this function do?

```
def second(nmbr1,nmbr2,nmbr3):
      if nmbr1 < nmbr2 and nmbr1 < nmbr3:
          return nmbr1
      elif nmbr2 < nmbr1 and nmbr2 < nmbr3:
          return nmbr2
      else:
          return nmbr3
```

c) When using the functions declared in a) and b), what is the output of this line?

```
print second(first(3,2),first(2,2),7)
```

Q5: Function foo() is declared and called in program. What is the output of the program?

Function:

```
def foo(a, b):
      result = 0
      while a < b:
          a = a + 1
          print a + b
          result = result + a
      return result
```

Program:

```
first = foo(2,4)
second = foo(5,3)
if first < second:
      print foo(first, second)
else:
      print foo(second, first)
```

Appendix C: Final exam, part 2

Q1: Define function `average()`, which gets a list as a parameter and calculates and returns the average of the values in the list. Example of usage:

```
list = [2,3,3,4,7]

print "The average is:", average(list)
```

The output of the example above would be:

```
The average is 3.8
```

Q2: Write a program that asks for a number and prints out the factorial of that number. Use exception handling to ensure that a number is given. Factorial of e.g. number 5 is calculated like this: 5 * 4 * 3 * 2 * 1 = 120. Example run:

```
Give a number: 6
The factorial of number 6 is 720
```

Q3: Write a program that reads a file and prints out the number of rows and words in that file. For example, for a file:

```
This is a file
There is more text here
And even more here
```

The output would be:

```
There are 3 rows and 13 words in the file.
```

Create a file `test.txt` to test your program. The words can be calculated with `split()` function, which can be used to split a string around spaces.

Paper 3

Kaila, E., Rajala, T., Laakso, M.J., Lindén, R.,
Kurvinen, E. and Salakoski, T. 2014.

Utilizing an Exercise-Based Learning Tool Effectively in Computer Science Courses.

Olympiads in Informatics, *8*, 93 – 109. Vilnius University.

Olympiads in Informatics, 2014, Vol. 8, 93–109
© 2014 *Vilnius University, IOI*

Utilizing an Exercise-Based Learning Tool Effectively in Computer Science Courses

Erkki KAILA, Teemu RAJALA, Mikko-Jussi LAAKSO,
Rolf LINDÉN, Einari KURVINEN, Tapio SALAKOSKI
University of Turku
e-mail: {ertaka, temira, milaak, rolind, emakur, sala}@utu.fi

Abstract. Educational technology and learning environments are becoming more and more common in all levels of education. Still, the main focus in research seems to be on which tools to use rather than how to effectively use them. In this paper, we first discuss the aspects that should be considered when adapting an exercise-based learning environment into curriculum. Based on our earlier research on the topic, we present three rules for adapting the tools. Next, a six-year study on using a learning environment in two courses is presented. Throughout the six course instances, the adaptation and integration of the tool is gradually altered. The results seem to confirm the positive effect of changes made in adaptation. When the three rules presented earlier are revisited in correlation with the results obtained, we can state that following the rules of adaptation lead to better student performance.

Keywords: learning environments, automatic assessment, course design, tool adaptation.

1. Introduction

According to multinational study by McCracken *et al.* (2001), programming is one the most difficult skills to acquire. There are various educational tools developed to aid the process, but comprehensive research about their pedagogical usage is still quite rare. Any tool or system, no matter how proficient, can only produce real educational value if adapted and utilized properly. In this article, we consider various factors that may have an effect on the efficiency of the tool usage: tool introduction, student engagement, motivation and reward. Based on our earlier research, most of these factors have a considerable effect on learning results. Hence, it seems that in addition to considering which tool to use, it's equally important to consider how to use it.

In this article, we present three rules for adapting a learning environment, based on our earlier experiments. Though named rules, they are actually ideas to consider when designing course structure and educational technology adaptation. We also present a comprehensive 6-year study, where a learning environment was used in two courses throughout six years. In latter instances, new exercise types were introduced to try to improve motivation. Some other changes in the tool and the usage were also introduced throughout the years. The holistic idea has been to gradually improve the tool adaptation to find out how the learning effects and motivation can be maximized. The student performance and grades are presented to find out whether the changes had effect in particular years.

2. Literature Review

There are various learning environments developed over the years. Crescenzi and Nocentini (2007) present a two year experiment of adapting an algorithm visualization tool into a programming course. The student feedback was mainly positive, but they don't report any changes in student performance. Laakso *et al.* (2005) adapted an algorithm visualization tool called TRAKLA 2 into two courses at separate universities at Finland. They found out that the pass rate increased significantly, and the student feedback was mainly positive. Still, the same group (Laakso et al, 2009) found out later that using the same tool in collaboration with another student has an even higher positive effect on learning. Hence, anyone adapting a tool should be encouraged to find out further whether the positive effects can be enforced.

Educational technology can of course be used in all kinds of courses. De Lange *et al.* (2002) surveyed students' opinions on adaptation of WebCT on accounting course, and found out that their satisfaction with environment is tightly associated with lecture notes, forum, on-line assessment and other tools in that environment. Paechter *et al.* (2010) suggest that the key factor in affecting students' motivation is making the learning objects transparent and providing possibility for self-assessment. Self-assessment should hence have a major role in any exercise-based learning environment. Students' attitudes should have a considerable effect on adaptation: Saunders and Klemming (2003) reported a two-year experiment where they integrated technology into traditional learning environment, and found out that though the students found the module harder to complete than others, their performance was actually better. The cognitive load for adapting new tools (see for example Chandler and Sweller, 1996) is an issue that should be considered when designing technology enhanced curriculums.

Liaw *et al.* (2007) also surveyed the attitudes of students and educators towards e-learning, and found out that the instructors' attitudes are highly positive. The analysis on students' attitudes revealed, that an effective learning environment is influenced by learner autonomy and teacher help, among other things. Hence, it is important to remember that educational technology is not something that can be added into curriculum and then forgotten. Lockyer and Patterson (2008) in fact state, that "the lecturers may have to play a considerable technical support role in helping students who are new to such technologies".

There are other studies that emphasize the instructors' satisfaction in educational technology. For example, Zuvic-Butorac *et al.* (2010) present a huge effort of implementing an e-learning environment of more than 400 courses and 15,000 students in Croatia. The teachers' attitudes were surveyed and found out to be highly positive towards the technology. Still, as O'Neill *et al.* (2004) state in their literature review about eLearning implementation, if new technology is to be integrated into learning properly, comprehensive training and support for instructors should be provided.

3. How to Adapt an Exercise Based Learning Environment

3.1. *Selecting a Suitable Environment*

The first step in adaptation is selecting a proper environment. There are various issues that should be considered when selecting the tool. First – and probably the most important issue – is that the selected tool should provide adequate benefits for both the teacher and the student. As discussed earlier, the most obvious benefit for teacher is the time saved in assessing exercises and assignments. However, to gain any real benefit in time saving, the environment either needs to come with an existing set of usable exercises or the time cost of preparing the exercises needs to be tolerable. As stated in Naps *et al.* (2001), the lack of time is the most important reason for teachers not using visualization tools; the same can probably be applied to any learning environment.

From the student's point of view, the most obvious benefits come from automatic assessment and immediate feedback. The ability to do the exercises any place and any time, and still get supportive feedback, is something that is hardly possible with traditional methods. Improved learning results (Kaila *et al.*, 2009A) are also a significant benefit both for the student and the teacher. Evaluating the learning effects outside controlled studies might be difficult as there are various factors influencing the learning outcome. Still, as shown in Kaila *et al.* (2010), and furthermore in the later sections of this paper, it is possible to significantly improve the results in CS course if a learning environment is introduced and used properly.

There are also some technical issues that need to be considered when selecting an environment. First, there is the initial cost of tool utilization and management. Though most of the common learning environments can be adapted free-of-charge, there might be hidden costs, such as upgrading server equipment and training the users. If the tool is hosted externally, these costs can however be kept in minimum. Moreover, the technical requirements for using the tool should be evaluated beforehand. Some exercises may need plugins – such as Java or Flash – installed into browser before working properly. In some physical environments installing additional components may be difficult or impossible.

3.2. *Three Rules for Adaptation*

In this section, we present three rules that should be taken into account when adapting a learning environment into a course. The rules are based on our earlier results on the topic, and are revisited when the results of this research are discussed.

3.2.1. *Rule 1: Introduce and Integrate*

The first rule is that the tool should be properly introduced and integrated into course. We have previously studied the effects of cognitive load on students when using a visualization tool (Laakso *et al.*, 2008). In the study, the students who went through a comprehensive tutorial about using the tool statistically significantly outperformed the control group. Hence, we suggest, that a separate introductory session should be arranged before the tool is adapted into actual learning. The introduction should be made from two points of view: technical and pedagogical. The technical introduction contains issues such as logging in and user interface. The pedagogical introduction should address issues such as the order and schedule of the exercises taken, using additional materials to assist learning, and the role of exercises as a part of comprehensive learning experience.

This leads as to the second part of this rule: we suggest that the learning environment should be properly integrated into the course. This means that the exercises in the environment should substitute and supplement the existing materials, where relevant. In practice, this may mean that the course needs to be partially redesigned. In Laakso *et al.* (2014) we presented a programming course reform, where half of the lectures were replaced with interactive tutorials that emphasized active and collaborative learning. The results were remarkable, as the dropout rate decreased and the grades improved statistically significantly after the change. Moreover, the students seemed to find the active approach more motivating and enjoyable.

3.2.2. *Rule 2: Engage the Students*

Naps *et al.* (2002) presented a hypothesis of engagement taxonomy, where they divided the usage of visualization tool into passive (no-viewing and viewing) and active (responding, changing, modifying and presenting). They suggested that using a visualization tool may only produce considerable learning if the tool is used in active levels. We later confirmed the hypothesis in Kaila *et al.* (2009B). We suggest that the results gained from using a visualization tool can be generalized to any types of exercises: if the students are engaged into active learning process, the results are better. Moreover, collaboration can be used to deepen the level of engagement. In Rajala *et al.* (2009) we found out, that if exercises are done in collaboration with another student, the learning results can be significantly improved. In Laakso *et al.* (2014) we presented a programming course reform (see previous Section), where collaboration was brought to classroom exercise sessions by introducing a collaborative mode in learning platform.

3.2.3. *Rule 3: Make it Mandatory, but Reward the Students*

As a third rule, we suggest that the usage of the tool should be made mandatory, but the students should still be rewarded from doing the exercises in the environment. A typical approach is to set minimum limits that need to be reached, and reward the students from exceeding that limit. The reward can be divided into two categories: an internal reward is something gained within the tool. Typically points are awarded when a student successfully completes an exercise or assignment. An external reward is something the students gain outside the learning environment. For example, the students may be awarded with grade improvement, bonus points for exam, or other forms of compensation from completing the exercises in the environment.

In Laakso *et al.* (2014) we present a case where students were required to complete at least five out of seven tutorials during the programming course. However, no minimum score limit was set. The students however completed a remarkable amount, 91% of all points on average, though reaching this amount meant doing extra work outside tutorial sessions. The students could pass the course without a final exam by completing at least 90% of all points awarded from all course components (including lectures, tutorials and course assignments). Still, of all students that reached the 90% level, only a handful skipped the final exam.

4. ViLLE

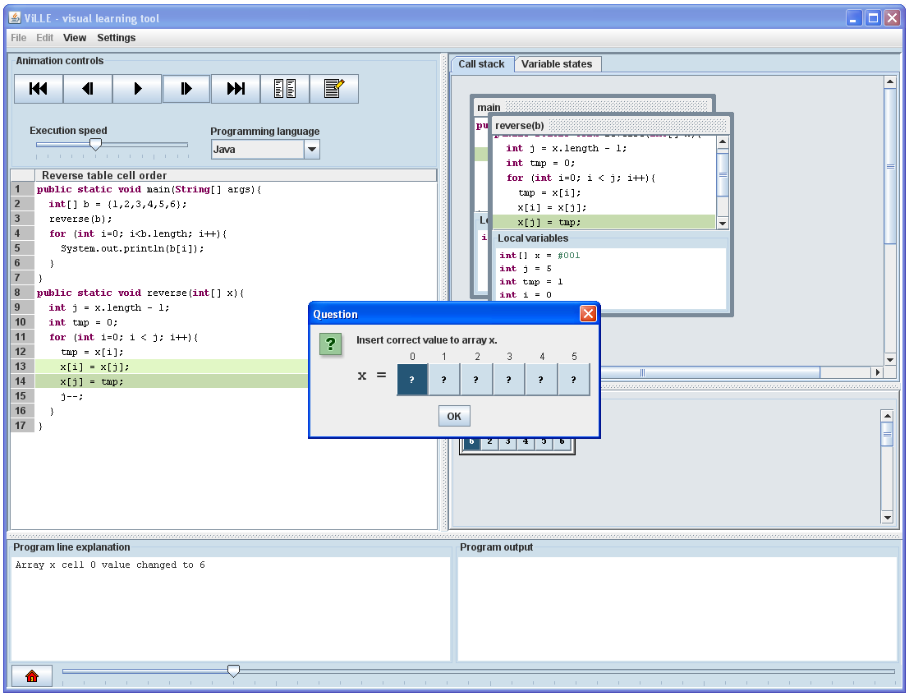
4.1. *Background*

ViLLE is a learning environment, developed at the University of Turku, Finland. It started out as a program visualization tool in 2004, and later expanded into comprehensive collaborative exercise and course management environment. From the beginning, ViLLE has been developed based on the research done. All major features have been tested with controlled experiments, and only the useful ones have been included in the published version. For example, the engagement taxonomy hypothesis (Naps *et al.*, 2002) lead into developing interactive questions into then-passive visualization tool, and the good experiences on collaborative use (Rajala *et al.*, 2009) encouraged us to develop collaborative mode that enabled two or more students working at the tasks together.

Since the earlier version was used in the first course presented in this paper, both versions are introduced separately.

4.2. *The Early Version of ViLLE – The Visual Learning Tool*

The first version of ViLLE is a program visualization tool (see Fig. 1) that can be used to display the execution of programs one row at a time. The execution is visualized with various components: the current and previous rows are highlighted, the variable states are displayed in their own area, and each subprogram with its local variables is displayed



Reverse table cell order

```
public static void main(String[] args){
  int[] b = {1,2,3,4,5,6};
  reverse(b);
  for (int i=0; i<b.length; i++){
    System.out.println(b[i]);
  }
}
public static void reverse(int[] x){
  int j = x.length - 1;
  int tmp = 0;
  for (int i=0; i < j; i++){
    tmp = x[i];
    x[i] = x[j];
    x[j] = tmp;
    j--;
  }
}
```

```
int[] x = #001
int j = 5
int tmp = 1
int i = 0
```

```
Array x cell 0 value changed to 6
```

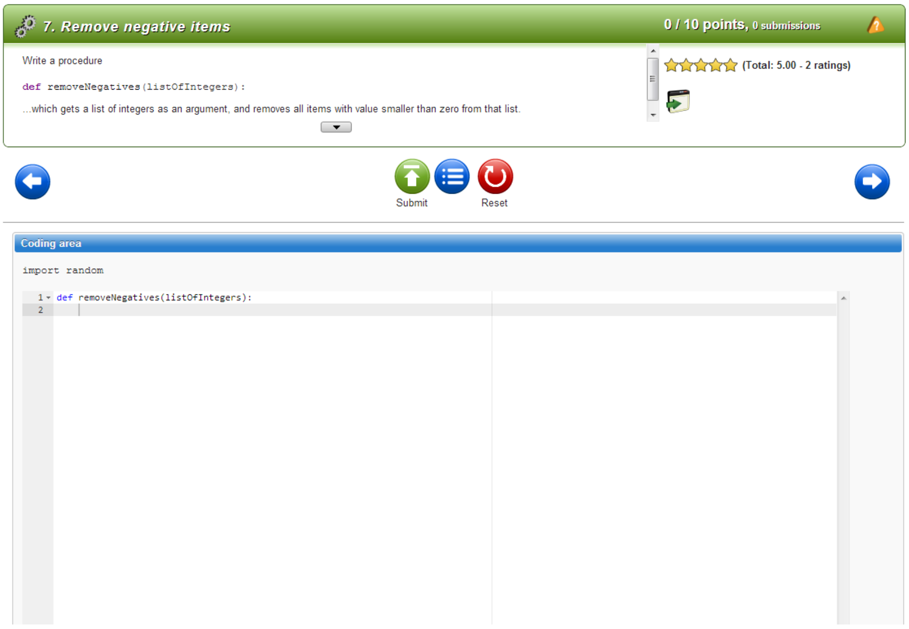
Fig. 1. ViLLE version 1: the student view displaying visualization exercise.

in a single frame in call stack and so on. Moreover, ViLLE displays a verbal explanation about the currently executed line. The tool supports a variety of imperative programming languages – including for example Java, Python and C++ – and automatically translates the programs written in Java to other supported languages. Students can view the execution in parallel view, which displays the executed program in two selectable languages at the same time.

To enhance active learning, the example programs can be accompanied with multiple choice questions or graphical array questions. The questions are inserted into desired steps in program. When a question is encountered the program execution halts until student gives an answer. ViLLE version 1 was deployed as a Java applet or Java application, but it could be connected to a TRAKLA II server (Malmi *et al.* 2004). In this case the server tracks student logins and all achieved points in different exercises. A complete description of the tool can be found in Rajala *et al.* (2007) and in Kaila *et al.* (2009A).

4.3. *ViLLE Now – a Collaborative Learning Environment*

As of 2009, ViLLE was expanded into an exercise-based collaborative learning environment. New client-server architecture was designed, with a focus on teachers' collaboration and with a support for various exercise types. In ViLLE version 2 (see Fig. 2), the teachers can use the built-in editors to create and edit virtual courses and assignments. Moreover, all content set as public can be browsed, utilized and modified by all other teachers registered in ViLLE. New exercise types for various topics were created. For programming, coding exercises, code sorting exercises and simulation exercises were designed among many others. Moreover, exercise types for mathematics, language


7. Remove negative items

0 / 10 points, 0 submissions

Write a procedure

```
def removeNegatives(listOfIntegers):
```

...which gets a list of integers as an argument, and removes all items with value smaller than zero from that list.

(Total: 5.00 - 2 ratings)

```
import random

def removeNegatives(listOfIntegers):

```

Fig. 2. Coding exercise in current version of ViLLE.

teaching and various other topics have been developed over the past few years. A comprehensive list about ViLLE exercise types can be found in Appendix A.

Since the new version introduced a dedicated ViLLE server, there was no more need to utilize the TRAKLA II server. ViLLE automatically collects a vast amount of data on student performance, including for example all achieved scores and time used to complete the exercises. Additional exercise specific data is also collected: for example, in visualization exercises ViLLE automatically saves all control usage data, including time stamps, when the student does the exercise. All data gathered can be viewed in ViLLE's statistical view by course's teachers.

The new version also supports collaborative learning where more than one student can join the same session. Besides exercises, there are various other tasks that can be used in courses: if accompanied with RFID readers, ViLLE can be used to easily record course attendances and demonstrations. It also supports study journals and course assignments, to name a few. All exercises, whether they are automatically assessed or not, can be used in electronic exams. It also has an editor for building tutorials that combine exercises with other materials, and a research project management system for research collaboration.

The complete description of the tool can be found in Laakso *et al.* (2014).

5. Methodology

5.1. *Overview*

The research was carried between years 2007 and 2012. The data was collected from two separate courses: in the first course – observed in three instances between 2007 and 2009 – the version 1 of ViLLE was used, while in the second course – with three instances between 2010 and 2012 – the newer version 2 was utilized. The usage of tool varied in different instances of the course: the tool was adapted more thoroughly year by year. A gradual increase in the usage was justified by excellent results and feedback gathered from teachers and students.

5.2. *Course Instances*

The first course observed (from now on **Course 1**) was called an Introduction to Information Technology. The goal of the course is to teach computer science fundamentals as well as introductory programming concepts to CS majors at University of Turku. The course is somewhat typical introductory course in computer science, containing basic principles of algorithms and data structures, accompanied with programming fundamentals in Python. Three instances of the course were researched: in 2007, ViLLE was introduced to the course. The usage of the tool was not mandatory; instead, a link to exercises was provided in course web page. In two consecutive instances, 2008 and 2009, ViLLE was made a mandatory part of the course: if the students did not complete at least 40% of all ViLLE exercises, they failed the course. All course instances were taught by the same teacher, and no other significant chances between instances were made.

The second course observed (from now on **Course 2**) is called an Introduction to Programming. It is a mandatory course in Bioinformatics program at University of Turku, and aims at teaching basic programming concepts in Python. The course hence contains essential topics in imperative programming, such as variables, loops and functions, as well as some Python specific topics, but does not include object oriented programming. Three instances of Course 2 were also observed: at 2010 ViLLE was included – as mandatory component, but with visualization exercises only. In two latter instances various other exercise types were introduced as well. In the latest instance (2012) ViLLE was also used to keep track on lecture attendances and demonstration scores, with bonus awarded for good performance on these components. Moreover, in the last instance, ViLLE was also used as a platform for course final exam. As was the case with Course 1, all instances were taught by the same teacher, and no other substantial chances were made in course through these instances.

The usage of ViLLE throughout the course instances is displayed at Table 1**.**

5.3. *Exercises*

In Course 1, ViLLE was first introduced as an optional supplement. At two later instances the usage of the tool was made mandatory. A total of 60 exercises were divided into seven categories: variables and conditions, strings, loops, sub programs, arrays, recursion and sorting algorithms. Each exercise consisted of visualized program code and 5 to 10 questions. Each exercise was scored in scale of 0 to 10 based on the correctness of answers. All exercise rounds were open from the beginning of the course, and references to suitable exercises were made on other course materials.

In Course 2, ViLLE was mandatory in all three instances. At the first instance only visualization exercises were used – the exercise collection was roughly equivalent to the collection used in Course 1 with minor modifications. At the two latter instances other exercise types were introduced. The course was hence divided into eight exercise rounds, based on the topics in course: the first round was an introduction to ViLLE, and the latter rounds about variables and data types, strings, selection, loops, functions, lists and tuples, followed with a round of additional exercises. Each round consisted of five different types of exercises:

Table 1
Usage of ViLLE at course instances

Year	Course	ViLLE exercises	Mandatory
2007	Course 1	Visualization	No
2008	Course 1	Visualization	Yes
2009	Course 1	Visualization	Yes
2010	Course 2	Visualization	Yes
2011	Course 2	Various	Yes
2012	Course 2	Various, including other performance and course exam	Yes

- *Visualization exercises*: these were similar to exercises used in Course 1 and in the first instance of Course 2. A handful of existing visualization exercises were selected, of which some were slightly modified to suit the topics better.
- *Code sorting exercises*: exercises where code lines were shuffled into random order, and the student needed to sort them into correct order according to given task.
- *Puzzle exercises*: an exercise, where the student needs to combine for example variable types and value ranges or string operations and results.
- *Coding exercises*: an exercise where the student needs to write a program – or a missing part of the program – in Python to fulfil given task. The program written in ViLLE can be instantly translated and executed.
- *Quizzes*: ten multiple choice and open questions about the topic at hand.

In Course 2 the exercises were integrated into course curriculum more tightly. Each round of exercises was opened after the lecture about corresponding topic was given. The exercises were designed to cover all aspects of the topic at hand as thoroughly as possible. After opening, all rounds were open until the final exam.

5.4. *Method*

Since the research contains two different courses, only instances of the same course are compared. From each course instance, final grades were obtained. The experiment is a between-subject design with final exam results a dependent variable. ViLLE usage was the only significant between-subject factor (independent variable), since no other significant changes in courses during the observed period were made: the instances were taught by the same teacher, and there were no substantial changes in other course components or materials. Since Course 2 used the most recent version of ViLLE, we also had access to comprehensive exercise data on those instances; hence, statistics about ViLLE usage in Course 2 are also discussed.

6. Results

6.1. *Course 1*

All instances of Course 1 were graded on scale of one to five, five being the best. If the student did not pass the course, no grade was given. The final grade distribution in all course instances is displayed in Table 2 and visualized in Fig. 3**.**

As seen on Table 2, the pass rate and the average grade improved at latter instances, when the exercises were made mandatory. The grade distribution is visualized in Fig. 3.

As seen on Fig. 3, the amount of lesser grades (1, 2 and 3) is clearly smaller at the latter instances of the course, compared to first year when ViLLE exercises where optional. To confirm this, a chi test between course grade distributions was used to calculate the independence between all instances, using the formula

$$\chi^2 = \sum_{i=1}^{6}\sum_{j=1}^{2}\frac{(C1_{ij} - C2_{ij})^2}{C2_{ij}}$$

where $C1$ and $C2$ are the course instances compared. The results are displayed at Table 3.

As seen on Table 3, the grade distribution at first instance is independent, while latter instances seem to follow the same pattern more tightly.

Table 2
Grade distribution in Course 1 instances

	2007 (N=131)	2008 (N=134)	2009 (N=181)
5	34	40	46
4	17	19	27
3	9	20	33
2	21	16	25
1	25	15	23
Fail	25	24	27
Total passed	106	110	154
% of all passed	80.92 %	82.09 %	85.08 %
Grade mean	3.13	3.48	3.31

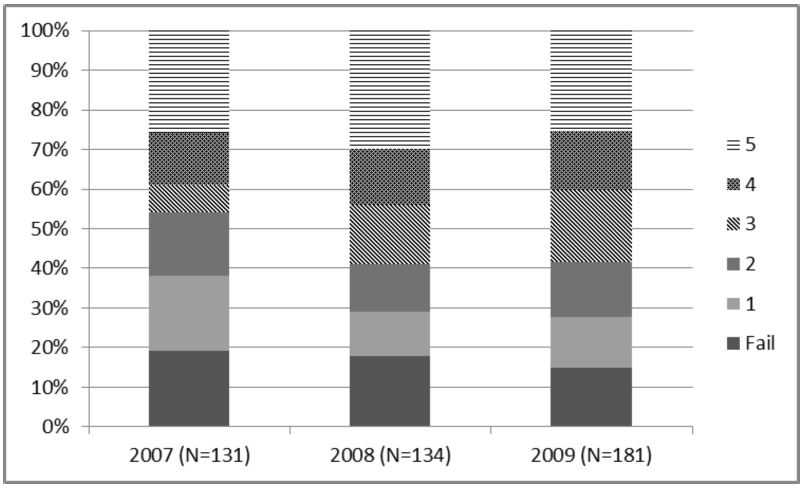


Fig. 3. Grade distribution at Course 1 visualized.

Table 3
Independence between grade distributions of Course 1 instances

Courses	2007 and 2008	2007 and 2009	2008 and 2009
χ^2	0.002	<0.0001	0.022

6.2. *Course 2*

It is to be noted, that the number of students in all instances of Course 2 was rather small. Still, certain trends can be observed. Course 2 was also graded in standard scale of 1 to 5. The final grade distribution of all instances is displayed in Table 4.

The percent of students who passed the course has been extremely high in all instances. However, it seems that there is a trend to be seen on the average grades: the average is higher on the latter instances of the course, where varied types of ViLLE exercises were used. The grade distribution is visualized at Figure 4.

Since all of the instances did use ViLLE exercises, and in all instances the usage was required, course grade averages were also compared to earlier instances (<2010) of the course (see Table 5). However, since the teacher was different, and there were other minor changes in the course at 2010 as well, the data should be observed with caution.

Points gathered from ViLLE exercises in all instances of Course 2 are displayed at Table 6.

Table 4
Grade distribution in instances of Course 2

	2010 (N=23)	2011 (N=16)	2012 (N=25)
5	10	10	16
4	3	1	2
3	4	1	3
2	2	1	1
1	3	1	2
Fail	1	2	1
% of all passed	95.65 %	87.50 %	96 %
Grade mean	3.52	3.75	4.04

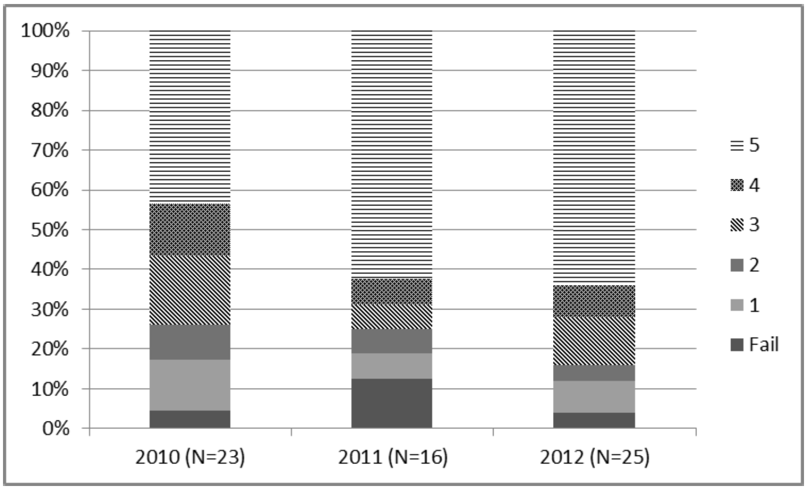


Fig. 4. Grade distribution at Course 2 visualized.

Table 5
Course 2 instances' mean grades throughout 2006…2012

Year	Grade mean
2006…2009 (N=21)	3.14
2010 (N=23)	3.52
2011 (N=16)	3.75
2012 (N=25)	4.04

Table 6
Points gathered in ViLLE in instances of Course 2

Year	Total maximum	Mean score	Std. dev.	% of maximum
2010	700	552.36	99.57	78.91 %
2011	660	567.06	128.06	85.92 %
2012	660	588.73	83.39	89.20 %

Though the differences are rather small, it seems that the students completed more exercises when visualization was accompanied with other exercise types starting from 2011.

7. Discussion

Based on the results presented, it seems that ViLLE exercises have a positive effect on learning. The average grade in both courses increased – though in Course 2 there are no significant changes (though this might be because of the low N). The pass rate in Course 1 also improved. In this section, the results for both courses are first discussed separately. Then the rules for adaption presented earlier are revisited in context of the results. Finally, as there are issues when measuring and comparing the performance at whole course level, some critical points of view are presented.

7.1. *Performance at Course 1*

Three instances of Course 1 were observed: at the first instance (2007) a link to ViLLE applet was given to students at course web page, but no points were collected and hence no minimum score limits set. In consecutive instances (2008 and 2009) ViLLE was made mandatory at course, as the minimum of 40% of all points in ViLLE needed to be collected to pass the course. Based on the results, it seems that this had an effect on learning results. The mean average increased, and the amount of lower grades (1, 2 and 3) decreased. Also, the passing percent increased from 80.92% to 82.09% and 85.08%, respectively.

It seems, that the visualization exercises combined with active learning in the form of questions has a positive effect on results. We have previously shown (see e.g. Kaila *et al.* 2009A, Laakso 2010), that visualization seems to have a highly positive effect on learning. It seems that the results gained from controlled two hour experiments can be generalized to learning at whole course. This also seems to confirm our earlier results on high school level programming course (Kaila *et al.*, 2010).

7.2. *Performance at Course 2*

There were also three instances observed in Course 2. In all of them ViLLE was made a mandatory part of the course, with minimum amount of 50 % of all points to be gathered to pass the course. The difference between instances was that at two latter instances (2011 and 2012) new exercise types were presented: only a handful of earlier visualization exercises were kept and four new exercise types were presented.

The same trend seems to exist at Course 2 results as well: when compared to earlier instances with no ViLLE (2009 and earlier) of the course, the grade mean seems to be higher when ViLLE exercises were used. Moreover, it seems that at the latter instances (2011 and 2012) of observed courses the distribution of grades seemed to focus more on the higher level of grades. No statistical differences could be found between groups, though one possible reason for this might be the low N. The trend in number of exercises completed at latter instances is still interesting: the students seemed to do more of the exercises when new types were introduced among the visualization. It is likely, that more heterogeneous set makes doing the exercises more motivating.

7.3. *The Rules of Adaptation Revisited*

The **first rule** we presented about adapting learning technology was to *introduce and integrate.* The results from Course 1 seem to underline this: when ViLLE was presented as an external tool with no connection to course otherwise, it did not seem to have a strong effect on learning. When the tool was made a mandatory part of the course, with connections drawn to other material, the grade and pass rate got higher. In Course 2 the introduction and integration was even tighter: there was a special introductory round in ViLLE where the exercise types were presented. Moreover, the exercise rounds in ViLLE were tightly integrated into course curriculum. Each round was opened after the lecture about the topic was given.

The **second rule** was to engage the students. The engagement taxonomy presented by Naps *et al.* (2002) states, that higher the level of engagement, the better the learning results. In latter instances of Course 2, new exercise types were presented. While visualization exercises lie in the engagement level of responding, most of the new types are on the higher levels of engagement. Based on the results, it seems that the students were more motivated in doing the exercises after the change, and it also seems, that the learning results were better. Though, as mentioned before, no statistically significant differences were found due to low number of students in course.

The **final rule** was to make the tool mandatory, but reward the students on using it. This rule was adapted on two final instances of Course 1 and in all instances of Course 2. In Course 1 the effect can be clearly seen: the results got better as soon as the tool was made mandatory. It is possible, that not all students find the visualization exercises motivating enough to complete them on their own. It is also likely, that at least the weaker students might not have enough patience to go through the more difficult exercises if they are not required. In Course 2 bonus points for final exam were rewarded if enough ViLLE points were gathered. This also seemed to have a motivating effect, as seen on scores obtained in ViLLE exercises: the 50 % minimum limit was clearly exceeded in all instances of the course.

7.4. *Issues in Course Long Performance Measurement*

There are some known issues when measuring the learning effects throughout the course. First, there are usually several factors that affect the learning results. In both courses, other variables were kept as steady as possible: the same teacher taught all instances of both courses and no significant changes in materials or course curriculum were made between instances. Still, isolating all factors that affect the learning is practically impossible. Also, measuring the actual learning outcome is difficult. The best we can do on course level is to compare the total grades obtained from course. As long as the components affecting the grade – and the components used to measure the grade – are kept somewhat similar, the mean grade should be reliable enough, – especially if the number of students in the course is high enough.

References

Chandler, P., Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, 10, 151–170.

Crescenzi, P. and Nocentini, C. (2007). Fully integrating algorithm visualization into a cs2 course: a two-year experience. In: *Proc. of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* : *Dundee, Scotland, June 25–27, 2007*. ITiCSE '07, ACM, New York, NY, 296–300.

DeLange, P., Suwardy, T., Mavondo, F. (2001). Integrating a virtual learning environment into an introductory accounting course: determinants of student motivation. *Accounting Education*, 12(1), 1–14.

Kaila, E., Rajala, T., Laakso, M.-J., Salakoski, T. (2009A). Effects, experiences and feedback from studies of a program visualization tool. *Informatics in Education*, 8(1), 17–34.

Kaila, E., Laakso, M.-J., Rajala, T., Salakoski, T. (2009B). Evaluation of learner engagement in program visualization. In: *12th IASTED International Conference on Computers and Advanced Technology in Education* (*CATE 2009*) : *November 22–24, 2009, St. Thomas, US Virgin Islands*.

Kaila, E., Rajala, T., Laakso, M.-J., Salakoski, T. (2010*)*. Long-term effects of program visualization. In: *12th Australasian Computing Education Conference (ACE 2010)* : *January 18–22, 2010, Brisbane, Australia*.

Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A., Malmi, L. (2005). Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1), 49–68.

Laakso, M.-J., Rajala, T., Kaila, E. and Salakoski, T. (2008). The impact of prior experience in using a visualization tool on learning to program. In: *Proceedings of CELDA 2008*. Freiburg, Germany, 129–136

Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A. and Malmi, L. (2005*)*. Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1), 49–68.

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

Laakso M.-J., Kaila E. and Rajala T. (2014). Ville – collaborative learning environment. Sent to *Computers and Education*.

Liaw, S.-S., Huang, H..M. and Chen, G.-D. (2007). Surveying instructor and learner attitudes toward e-learning environments. *Computers & Education*, 49(4), 1066–1080.

Lockyer, L., Patterson, J. (2008). Integrating social networking technologies in education: a case study of a formal learning environment. In: *Proceedings of 8th IEEE international conference on advanced learning technologies*. Santander, Spain (2008), 529–533.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment : TRAKLA2. Informatics in Education, 3(2), 267–288

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J. Á. (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.

O'Neill, K., Singh, G., O'Donoghue, J. (2004). Implementing e-learning programmes for higher education : a review of the literature. *Journal of Information Technology Education*, 3, 312–23.

Paechter, M., Maier, B, Macher, D. (2010). Students expectations of, and experiences in e-learning : their relation to learning achievements and course satisfaction. *Computers & Education*, 54, 222–229.

Rajala, T., Kaila, E., Laakso, M.-J., Salakoski, T. (2009). Effects of collaboration in program visualization. In: *Technology Enhanced Learning Conference 2009* : *TELearn 2009, October 6 to 8, 2009, Academia Sinica, Taipei, Taiwan*.

Rajala, T., Laakso, M.-J., Kaila, E., Salakoski, T. (2007). VILLE – a language-independent program visualization tool. In: Lister, R. and Simon (Eds.) *Koli Calling 2007 – Proceedings of the Seventh Baltic Sea Conference on Computing Education Research*, *Koli National Park, Finland, November 15–18, 2007*. (Conferences in Research and Practice in Information Technology, 88). Koli National Park, Finland, ACS.

Saunders, G., Kelmming, F. (2003). Integrating technology into a traditional learning environment. *Active Learning in Higher Education*, 4, 74–86.

Zuvic-Butorac, M., Nebic, Z., Nemcanin, D. (2011). Establishing an institutional framework for an e-iearning implementation –experiences from the University of Rijeka, Croatia. *Journal of Information Technology Education*, 10, 44–56.

Appendix A. ViLLE Exercise Types

Exercises for Computer Science

Visualization exercise: Combines the graphical, step-by-step execution of the example program with three types of questions: multiple choice questions, open questions and array questions.

Code sorting exercise: Commonly known as Parson's puzzle: the students need to arrange the shuffled program code lines into correct order so that given task is fulfiled.

Coding exercise: The task is to write a program – or a missing part of the program according to given specifications. ViLLE supports a variety of programming languages, including Java, C++, C# and Python.

Robot exercise: The goal of the exercise is to move number of boxes into specified target locations. The boxes are moved by writing an algorithm that controls a robot crane. Idea is to teach loops and methods in Java.

Clouds & Boxes: Reverse-visualization type exercise: the students are supposed to simulate the state of program after each step executed.

Other CS exercises: In addition, there are exercise types for testing binary calculations and conversions between hexadecimal, decimal and binary.

Exercises for Mathematics

Math exercises for elementary school level: There are several exercise types meant for teaching elementary level mathematics. In these exercises, the students for example need to find out the missing number, drag and drop numbers into number line, do long division, find out values in bar charts, calculate with fractions and so on.

Math exercises for higher levels: There are also exercise types meant for students in higher levels: for example, solving quadratic and first degree equations, doing differential coefficient calculations and writing inequality equations and sign charts.

Other Exercises

Quiz: The most basic exercise type: contains multiple choice and open questions with attachable materials. Quizzes can be utilized in any level and topic.

Survey: Can be used for course opening and closing surveys. Moreover, ViLLE surveys are typically utilized to implement assignments that are graded by teacher, for example essays.

Sorting: ViLLE contains exercise types for image puzzles, and for general sorting and pair matching of textual items.

Language exercises: There are several exercise types meant specifically for language teaching (such as fill-in, dialog, word ordering, punctuation and case, vocabulary test, compound exercise and so on). However, most of these can be utilized under other topics as well.

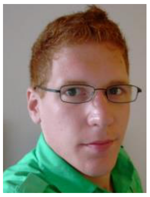
Image tagging: Exercise where the students need to identify areas in given or uploaded image.

E. Kaila, M.Sc., is working on his PhD about utilizing learning environments and assignments effectively. He has 25 scientific publications in peer-reviewed journals and conference proceedings. His research interests include program visualization, learning environments, automatic assessment and course design utilizing new technologies. He has been working on development of ViLLE from the beginning of the project.

T. Rajala, M. Sci., is a university teacher in Software Engineering at University of Turku, Finland. In addition to teaching, his work and research mainly focuses on developing educational software tools and studying their effectiveness in learning programming and algorithmic problem solving. Rajala finished his master's degree at University of Turku in 2007. He has 25 scientific publications in peer-reviewed journals and conference proceedings. He has also been working on ViLLE project since the beginning.

M.-J. Laakso, PhD(tech), is a lecturer and adjunct professor at Department of Information Technology at University of Turku. He has more than 35 scientific publications in internationals journals and conference proceedings. His main research interests are educational technologies, learning environments, automated assessment, visualization, immediate feedback, eAssessment and effect of collaboration in aforementioned topic. He is heading ViLLE team research group at University of Turku studying all these aspects and developing ViLLE – the collaborative education platform.

R. Lindén, M.Sc., is working on his PhD about automated student profiling and counselling. His research interests involve data mining, systems analysis and graph theory. He has three publications in peer-reviewed journals and conference proceedings. Lindén has been working on ViLLE project since the beginning of 2012.



E. Kurvinen, M.A. (Education), is finishing his M.Sc. in computer science, and starting his PhD about recognizing learning disabilities in mathematics. His research interests mainly concern usage of educational technology in mathematics education in all levels. Kurvinen has been working on ViLLE project since the beginning of 2012.

T. Salakoski, PhD, is a professor of Computer Science at University of Turku. He is the Dean of Science and Technology Education at the university, and the Head of the Department of Information Technology. He has more than 200 scientific publications in international journals and conference proceedings, and has supervised more than 10 PhDs and numerous MScs. He serves in scientific editorial boards and has organized and chaired international conferences. He is heading a large research group studying machine intelligence methods and interdisciplinary applications, especially information retrieval and natural language processing in the biomedical and health care domain as well as technologies related to human learning, language, and speech.

Paper 4

Kaila, E., Rajala, T., Laakso, M.J., Lindén, R.,
Kurvinen, E., Karavirta, V. and Salakoski, T. 2015.

Comparing Student Performance between Traditional and Technologically Enhanced Programming Course.

In Proceedings of the Seventeenth Australasian Computing Education Conference (ACE 2015). CRPIT, 160, 147 – 154. Australian Computing Society, Inc.

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, rolind, 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.

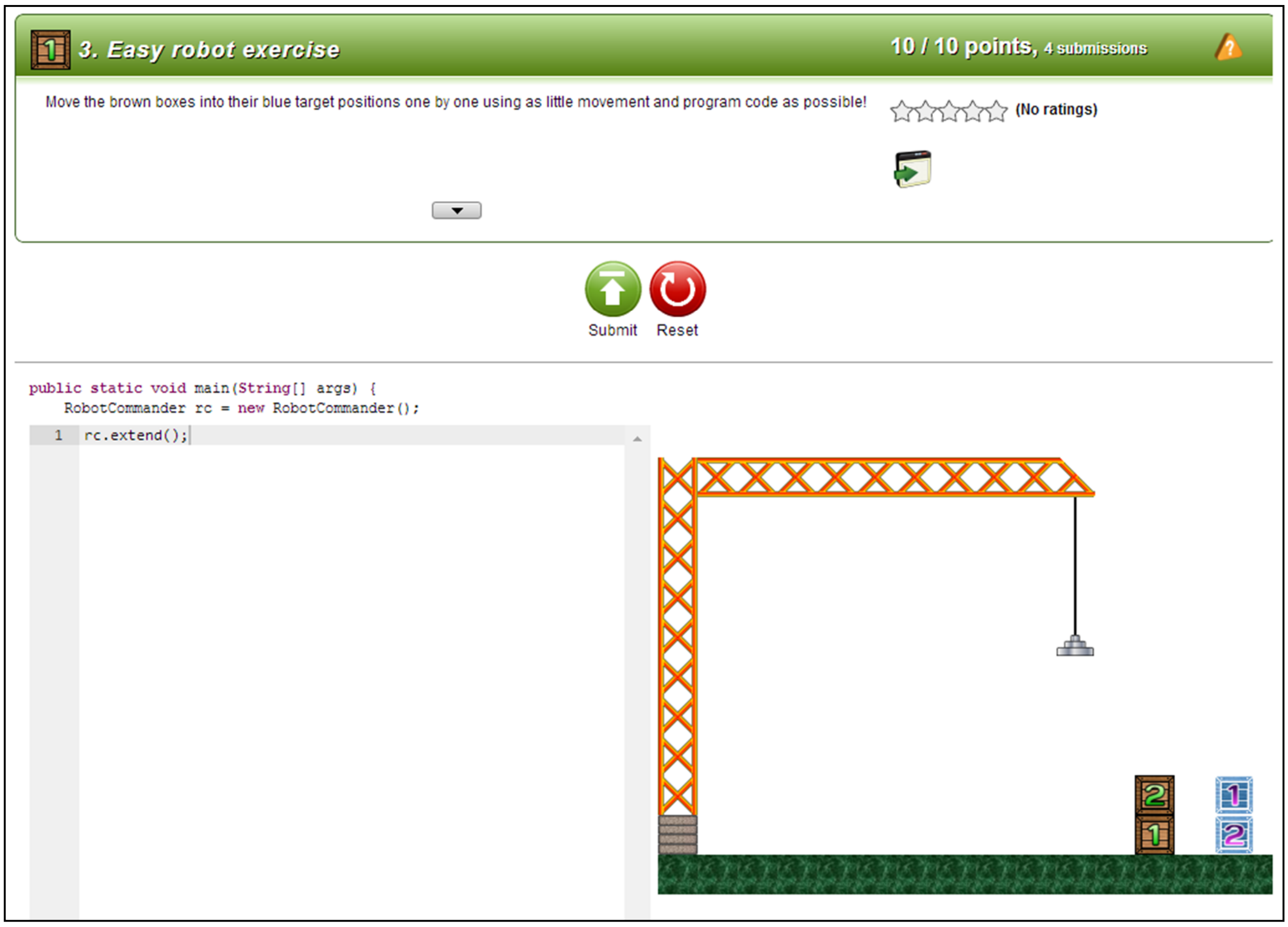
In education, active learning is generally considered as a valid method for engaging students and for improving motivation and results (Freeman et al., 2014). According to constructivist learning theories (see e.g. Papert, 1980, Moons et al., 2013), the knowledge can be constructed by actively participating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


3. Easy robot exercise
10 / 10 points, 4 submissions
Move the brown boxes into their blue target positions one by one using as little movement and program code as possible!
(No ratings)
Submit
Reset
2
1
1
2

```
public static void main(String[] args) {
    RobotCommander rc = new RobotCommander();
rc.extend();
```

Figure 1: Robot exercise in ViLLE

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 (http://ville.cs.utu.fi).

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

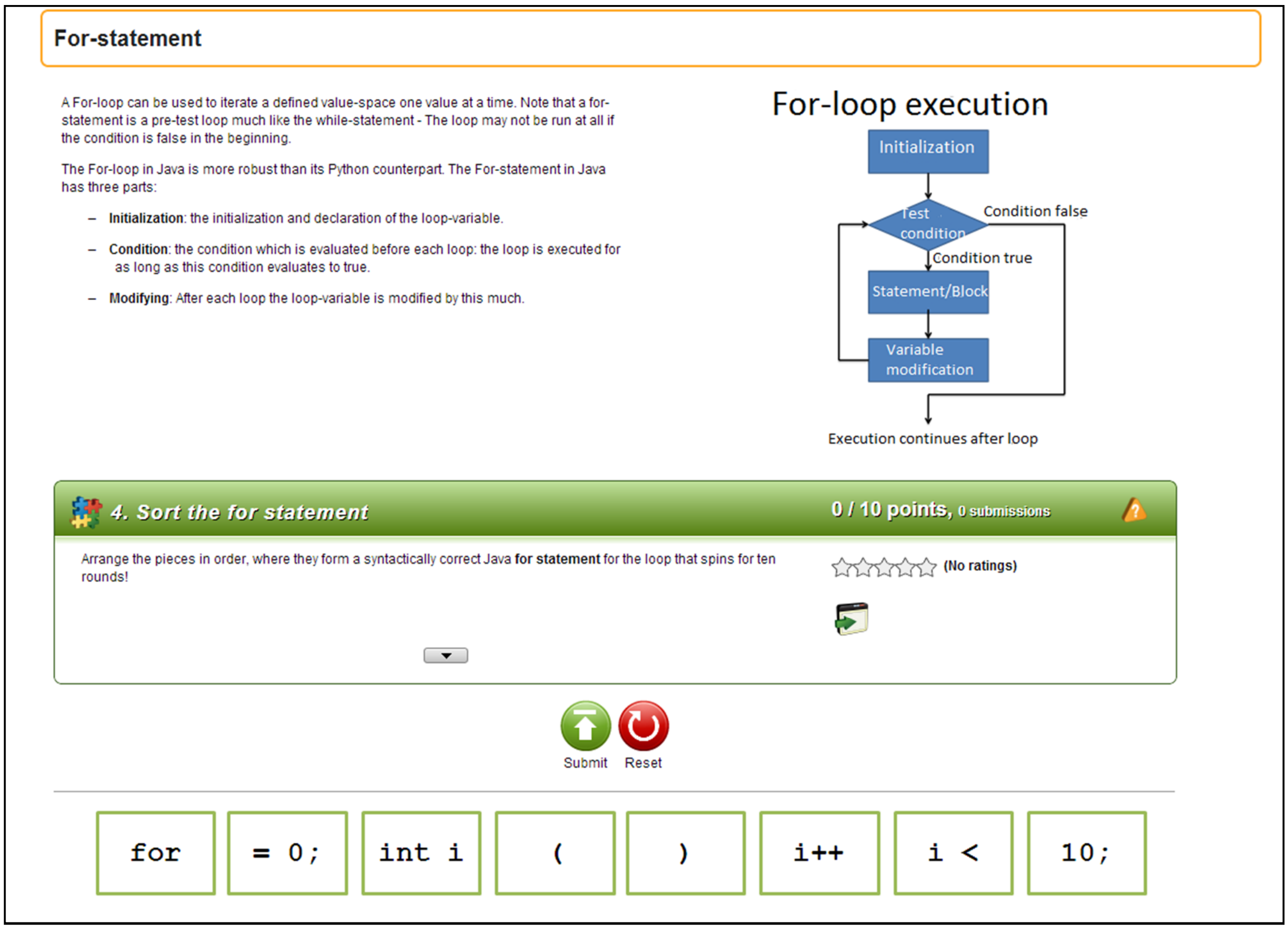
The first, and probably the most important, step was to introduce a concept of more active learning by using tutorials. In the 2011 instance of the course, there were two 2-hour lectures each week. In C2012, one of the lectures each week was replaced with a tutorial-based active learning session. The 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.


For-statement

A For-loop can be used to iterate a defined value-space one value at a time. Note that a for-statement is a pre-test loop much like the while-statement - The loop may not be run at all if the condition is false in the beginning.

The For-loop in Java is more robust than its Python counterpart. The For-statement in Java has three parts:

- **Initialization**: the initialization and declaration of the loop-variable.
- **Condition**: the condition which is evaluated before each loop: the loop is executed for as long as this condition evaluates to true.
- **Modifying**: After each loop the loop-variable is modified by this much.

4. Sort the for statement — 0 / 10 points, 0 submissions

Arrange the pieces in order, where they form a syntactically correct Java **for statement** for the loop that spins for ten rounds!

(No ratings)

```
for    = 0;    int i    (    )    i++    i <    10;
```

Figure 2: Tutorial view in ViLLE

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 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.

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

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.

	C2011	C2012
Course time	October to December, 2011	October to December, 2012
Methodology	Traditional	Renewed
N	210	193

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.

C2011	C2012
50 % of demonstration points	50 % of demonstration points 50 % of tutorial points 50 % of ViLLE exercise points Participation in minimum of 5 tutorial sessions

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.

	C2011	C2012
N	210	193
Students participating in exam	149	167
% of all students in exam	70.95 %	86.53 %

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.

Grade	C2011	C2011 proportion	C2012	C2012 proportion
5	45	30 %	70	42 %
4	21	14 %	19	11 %
3	20	13 %	23	14 %
2	12	8 %	19	11 %
1	14	9 %	25	15 %
Fail	37	25 %	11	7 %
Total	149	100 %	167	100 %

Table 5: Grade distribution in course instances

The distribution is visualized in Figure 3.

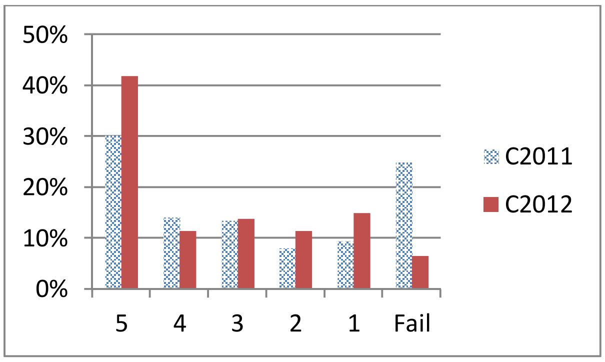


Figure 3: Grade distribution in course instances visualized

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**.**

	C2011	C2012
Total N	210	193
Qualify to take exam	70.95 %	86.53 %
% passed exam (of qualified)	75.17 %	93.41 %
% passed course	53.33 %	80.82 %
Grade mean (of passed)	3.63	3.57
Grade std. dev.	1.53	1.41

Table 6 Course performance results

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

Beck, K. & Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional

Bradford, P., Porciello, M., Balkon, N. & Backus, D. (2006) The Blackboard Learning System: the be all and end all in educational instruction? *Journal of Educational Technology Systems*, 35, 3, 301-314.

Caspersen, M & Bennedsen, J. (2007). Instructional design of a programming course: a learning theoretic approach. In *Proceedings of the third international workshop on Computing education research* (ICER '07). ACM, New York, NY, USA, 111-122

Cole, J. & Foster, H. (2008). *Using Moodle: Teaching with the Popular Open Source Course Management System (2nd ed.).* Sebastopol: O'Reilly Media Inc.

Crescenzi, P. and Nocentini, C. (2007). Fully integrating algorithm visualization into a cs2 course: a two-year experience. *Proc. of the 12th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Dundee, Scotland, June 25 - 27, 2007). ITiCSE '07. ACM, New York, NY, 296-300.

Freeman, S., Eddy, S., McDonough, M, Smith, M., Okoroafor, N., Jordt, H. & Wenderoth, M. (2014) *Active learning increases student performance in science, engineering, and mathematics.* PNAS 2014; published ahead of print May 12, 2014.

Grissom, S., McNally, M. and Naps, T. 2003. Algorithm Visualization in CS Education: Comparing Levels of Student Engagement. In *Proceedings of the ACM Symposium on Software Visualization*, San Diego, California, 87-94.

Hall, S., Fouh, E., Breakiron, D., Elshehaly, M., & Shaffer, C.A. (2013). Evaluating Online Tutorials for Data Structures and Algorithms Courses. In *Proceedings of the 2013 ASEE Annual Conference & Exposition,* Atlanta, GA, June 2013

Hundhausen, C.D. and Brown, J.L. 2007. What You See Is What You Code: A 'Live' Algorithm Development and Visualization Environment for Novice Learners. Journal of Visual Languages and Computing, 18, 1, 22-47.

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.

Kaila, E., Rajala, T., Laakso, M.-J. & Salakoski, T. (2010). Long-term Effects of Program Visualization. In *12th Australasian Computing Education Conference* (ACE 2010), January 18- 22, 2010, Brisbane, Australia.

Kaila, E., Rajala, T., Laakso, M.-J., Lindén, R., Kurvinen, E. & Salakoski, T. (2014). Utilizing an Exercise-based Learning Tool Effectively in Computer Science Courses. *Olympiads in Informatics 8.*

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

Kölling, M., Quig, B., Patterson, A. and Rosenberg, J. 2003. The BlueJ system and its pedagogy. Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13, 4.

Laakso, M.J, Kaila, E. & Rajala, T. (2014) ViLLE: designing and adapting a collaborative exercise-based learning environment. Sent to *Computers & Education*

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

Lahtinen, E., Ala-Mutka, K. & and Järvinen, H.-M.. (2005). A study of the difficulties of novice programmers. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05)*. ACM, New York, NY, USA, 14-18

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O. and Silvasti, P. 2004. Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. *Informatics in Education*, 3, 2, 267-288.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001). *A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students.* ACM SIGCSE Bulletin, 33, 4, 125-140.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. In Proceedings of the 33rd *SIGCSE technical symposium on Computer science education* (SIGCSE '02). ACM, New York, NY, USA, 38-42.

Moons, J. & De Backer, C. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education* 60, 368-384.

Oechsle, R. and Schmitt, T. 2001. JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). *Revised Lectures on Software Visualization, International Seminar*, May 20-25, 76-190.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY, USA: Basic Books, Inc.

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 (ACE '06)*, Denise Tolhurst and Samuel Mann (Eds.), Vol. 52. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 157-163

Rajala, T., Kaila, E., Laakso, M.-J. & Salakoski, T. (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.

Rajala, T., Salakoski, T., Kaila, E. & Laakso, M-J. (2010). How Does Collaboration Affect Algorithm Learning? A Case Study Using TRAKLA2 Algorithm Visualization Tool. In *Proceedings of 2010 International Conference on Education Technology and Computer (ICETC 2010)*, Jun 2010. [A4]

Rajaravivarma, R. (2005) A Games-Based Approach for Teaching the Introductory Programming Course. *Inroads – The SIGCSE Bulletin.* 37, 4, 98-102.

Saunders, G. & Kelmming, F. (2003) Integrating technology into a traditional learning environment. *Active Learning in Higher Education* 4: 74–86.

Tan, P.-H., Ting, C.-Y & Ling, S.-W. (2009). Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. *International Conference on Computer Technology and Development* (ICCTD ’09).

Paper 5

Kaila, E., Kurvinen, E., Lokkila, E. and Laakso, M.J. 2016.

Redesigning an Object-Oriented Programming Course.

ACM Transactions on Computing Education (TOCE), 16 (4), Article no. 18. ACM New York, NY, USA.

Redesigning an Object-Oriented Programming Course

ERKKI KAILA, EINARI KURVINEN, ERNO LOKKILA, and 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, facilitate communication between students and teachers, and remodel the evaluation procedure by utilizing automatically assessed tasks. The redesign was based on the best practices found in our own earlier research and that of the research community, with a 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 toward higher grade averages in the redesigned instances. Based on the results, we can conclude that the utilization of educational technology has a highly positive effect on 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 Trans. Comput. Educ. 16, 4, Article 18 (August 2016), 21 pages.
DOI: http://dx.doi.org/10.1145/2906362

1. INTRODUCTION

During recent years, an imminent need to redesign teaching methods in information technology education has become obvious. Students find the topics difficult and seem to have problems with the abstract concepts [Dunican 2002]. The problems are evident in programming courses, where the drop-out rates typically 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 (OOP). 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 with similar tools and assessed by the teaching staff. Utilizing educational technology makes it possible

Authors' address: Department of Information Technology, University of Turku, 20014-Turun yliopisto, Finland; emails: {ertaka, emakur, eolokk, milaak}@utu.fi.

DOI: http://dx.doi.org/10.1145/2906362

to increase the level of active learning: By facilitating features such as automatic 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 article, we describe a comprehensive redesign of an OOP 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 because various earlier studies have proved 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 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. Various researchers [Ben-Ari 2001; Jenkins 2002; Pattis 1993; Bennedsen and Caspersen 2004; McCracken et al. 2001] raise concerns about 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 and 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 that the required skills do 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, by Bloom's taxonomy [Bloom 1954]. Hence, teaching programming as a single skill is futile. However, good results can be achieved using a combination of different teaching methods [McCracken et al. 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, teaching should let students build on their old experiences and knowledge [Wertsch 1985]. Thus, active methods of learning are preferred. According to Freeman et al. [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 a *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 et al. 2001]. Immediate feedback can, in the best case, provide students with a cognitive conflict between what they thought was correct and what actually is. Such a conflict forces the student to reassess her beliefs, possibly seek out new information, and finally assimilate all newly gathered and re-evaluated information with her 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. 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, 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 and 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 group processing to promote student learning and understanding of the material. 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] and McDowell et al. [2002]). Moreover, Nagappan et al. [2003] found that pair programming can also improve student experience of the course because pair programmers were more self-sufficient and more likely to complete the class than were 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 computer science 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 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 individually but then are allowed to change their answers after a group discussion. The group discussion enables 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, one 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 to 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. They point out that instilling change in any educational system is challenging; however, 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 their quantitative effect on student performance in CS1 courses. They discuss several methods of intervention (including, e.g., 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 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 article describes the refactoring of an OOP course. The refactoring was 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's solution against the model's solution written by the teacher. The exercise provides authentic compiler and runtime exceptions, enabling students to fix 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 in which graphical tracing of code execution is accompanied by different types of questions.
—*Program simulation*: An exercise type in which students need to simulate program execution one code line at a time by creating and manipulating variables and methods.
—*Sorting exercise*: ViLLE supports Parsons puzzles [Parsons and 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 attendance using RFID readers. This functionality can be extended to demonstrations, where students can use ViLLE to record assignments they have completed. This allows the course staff and students to see all obtained scores in real time.

4. REFACTORING THE COURSE

The basic course of OOP is a typical OOP course taught in the University of Turku. The course is mandatory for all computer science majors (and for some other students in the faculty, including mathematics, physics, and chemistry majors), and 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 Java programming. In the second course, the fundamental concepts of OOP, 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 general concepts instead of features typical for any particular language.

In this section, we describe the changes made in the course between 2011–12 (the old course) and 2013–14 (the new course). The topics covered by the course as well as the learning goals and the credits awarded remained the same. The goal of the refactoring was to lower drop-out rates by facilitating active learning, student collaboration, and communication between the students and the teachers. By refactoring, we 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 refactoring has no effect on the drop-out rates, and, if it does, 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 article, 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 × 2 hours). Additionally, there were four weekly demonstrations, starting from the third week. In these two-hour sessions, the students provided their answers to programming tasks given earlier. At least 50% of demonstrations needed to be completed to attend the exam.

The old course structure is displayed in Table I.

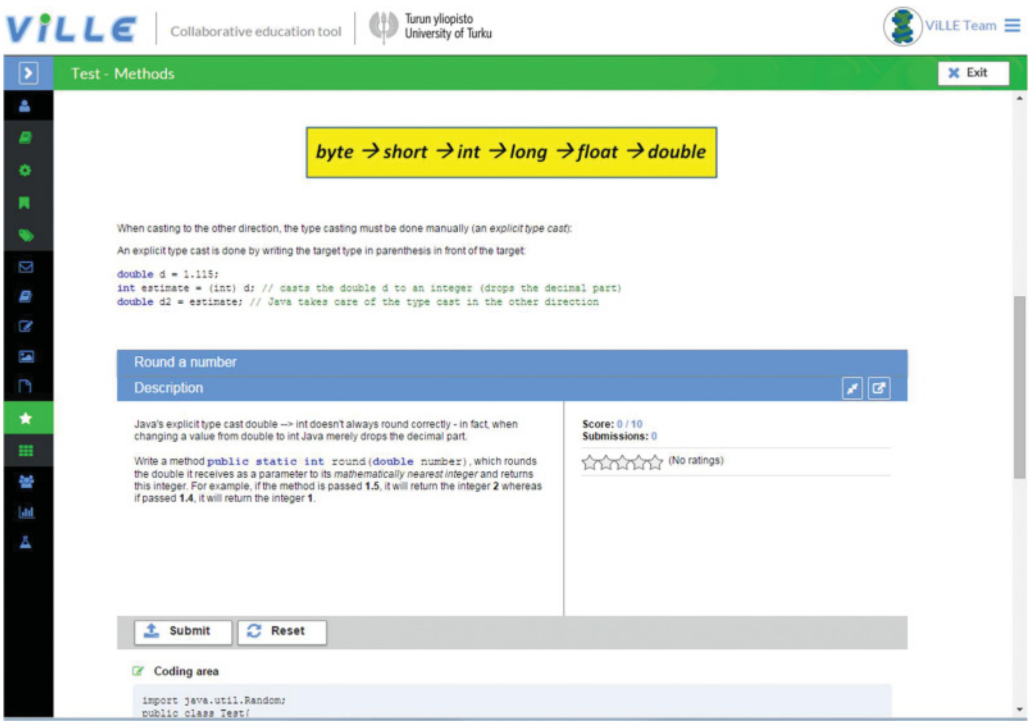
As seen in Table I, the course follows the typical structure of most programming courses.

4.2. Step 1: Enhancing Active Learning

In refactoring, the first step was to facilitate active learning. As proved 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

Table I. Summary of the Old Instance of the Course

Component	Amount	Description
Lectures	$7 \times 2 \times 2$ h = 28 h	Traditional lectures in a lecture hall
Demonstrations	4×2 h = 8 h	Programming assignment presentation in front of class; done in smaller groups (typically 20–30 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



When casting to the other direction, the type casting must be done manually (an *explicit type cast*):

An explicit type cast is done by writing the target type in parenthesis in front of the target:

```
double d = 1.115;
int estimate = (int) d; // casts the double d to an integer (drops the decimal part)
double d2 = estimate; // Java takes care of the type cast in the other direction
```

Round a number

Description

Java's explicit type cast double --> int doesn't always round correctly - in fact, when changing a value from double to int Java merely drops the decimal part.

Write a method `public static int round(double number)`, which rounds the double it receives as a parameter to its *mathematically nearest integer* and returns this integer. For example, if the method is passed **1.5**, it will return the integer **2** whereas if passed **1.4**, it will return the integer **1**.

Score: 0 / 10
Submissions: 0

☆☆☆☆☆ (No ratings)

```
import java.util.Random;
public class Test{
```

Fig. 1. Example of a tutorial in ViLLE.

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 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 that were directly connected to ViLLE server. This enabled an up-to-date view of the attendance for both teachers and students.

Each tutorial was designed to underline the topics discussed in that week's lecture. Hence, the lecture and 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. In this way, 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 tutorial exercises. ViLLE was also used to collect lecture and tutorial attendance. Lecture attendance was 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 in Rajala et al. [2009, 2011], learning performance can be significantly improved if students do the exercises in collaboration with other students. With this in mind, the tutorials were built to support collaboration. In the tutorial sessions, students paired with other students 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 15 minutes. Points collected from the tutorial exercises were awarded to both students.

The demonstration sessions were retained in the redesigned model as well because they offered more complex programming tasks as well as problems that do not have straightforward solutions. 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 completed assignments, although now using ViLLE. The demonstrator then used ViLLE to divide the students randomly into groups (an algorithm was used to ensure that each group had access to 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 previous ones were presented. The remaining time in the session (usually around 15–30 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 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 about 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 readdressed 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 that students reported on the surveys.

Also, after each of the demonstration sessions, a short survey was conducted. In this survey, students could simply give feedback on the demonstrations. Again, the answers were analyzed and the consecutive demonstrations were modified based on this feedback. In practice, this meant, for example, giving more detailed instructions on assignments if some 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, students participating in the course could ask for assistance with tutorials, demonstrations, or the weekly assignments. At least two mentors were present to assist students in these sessions. These mentors were also present at the tutorial sessions to assist 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 unsatisfactory. Instead, an electronic exam was implemented using ViLLE. In this way, 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 in an exam was as close to an authentic situation as possible. In addition to programming assignments, some other tasks, such as quizzes and sorting exercises, were also included.

The typical exam programming task contained a task description and (in some cases) a predefined set of code. Students were 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.

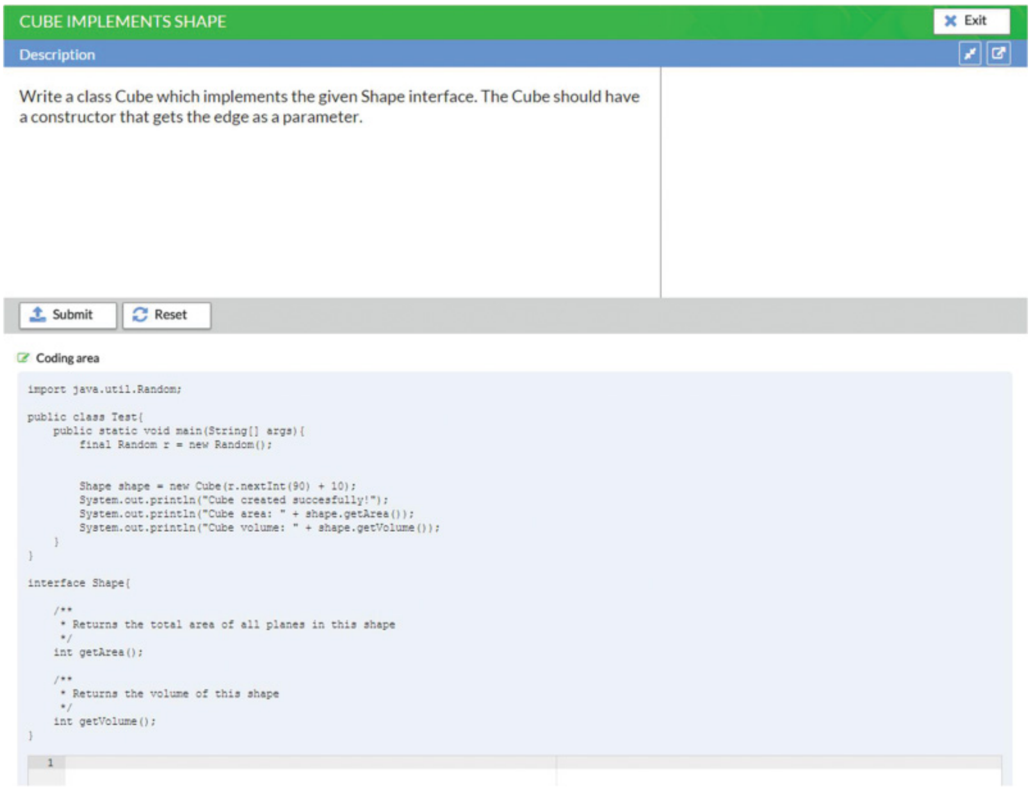
The electronic exam was fully automatically assessed. Unlike in the tutorial exercises, students couldn't see the feedback (except for the program output and possible compiler and run-time errors) or the score when submitting their answers. In addition to providing students with the possibility for resubmitting their answers as many times as they 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. C1 exams typically consisted of two or three questions, one or two of which 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 C2 exam was at least as difficult as the C1, the exam was audited by three nonaffiliated university-level teachers and researchers who all agreed that the new exam was at least as challenging as the old one.

4.6. Conclusion: Redesigned Course

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

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



Write a class Cube which implements the given Shape interface. The Cube should have a constructor that gets the edge as a parameter.

```
import java.util.Random;

public class Test{
    public static void main(String[] args){
        final Random r = new Random();


        Shape shape = new Cube(r.nextInt(90) + 10);
        System.out.println("Cube created succesfully!");
        System.out.println("Cube area: " + shape.getArea());
        System.out.println("Cube volume: " + shape.getVolume());
    }
}

interface Shape{

    /**
     * Returns the total area of all planes in this shape
     */
    int getArea();

    /**
     * Returns the volume of this shape
     */
    int getVolume();
}
```

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

Table II. Summary of the Redesigned Course

Component	Amount	Description
Lectures	$7 \times 2 = 14$ h	Traditional lectures in a lecture hall
Tutorials	$7 \times 2 = 14$ h	Tutorial sessions done in collaboration in lecture hall
Demonstrations	$4 \times 2 = 8$ h	Programming assignment presentation in front of class; done in smaller groups (typically 20–30 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.

As seen in Table III, active learning, collaboration, and communication are enhanced heavily in the new version of the course. In Section 4.7, we present the earlier studies on which the redesign was based, followed in Section 5 by the results on student performance in old and new instances. In Section 6, the effects and our experiences are discussed.

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 × 2h	4 × 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

Table IV. Course Instances

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

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 had previously shown in two-hour controlled tests [Rajala et al. 2008; Kaila et al. 2009] that educational technology can be highly beneficial for learning but only if used with 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 also showed in Rajala et al. [2009, 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. As in Laakso et al. [2008], we found 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, 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.

The total number of students in instances of the old methodology (C1) was 387, and in 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 number of students taking the course in 2014 is likely due to a higher pass rate in 2013 (and hence smaller numbers of students retaking the course the following year). The course is typically taken as part of first-year studies and is the second programming course in the curriculum.

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

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%

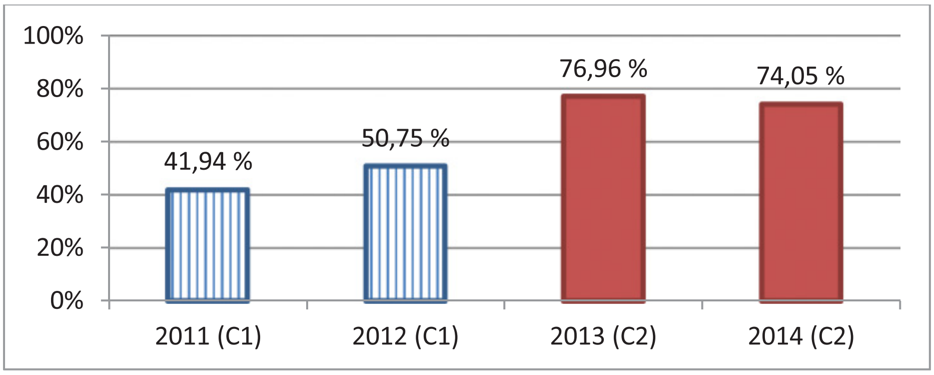


Fig. 3. Pass rates of all instances.

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 students who collected this number of points also took the exam. The grades from all course instances are displayed in Table V.

As seen Table V, the number of failed students is significantly higher in the older instances. The pass rates for all instances are shown graphically in Figure 3.

As seen in Figure 3, the pass rate was significantly higher in both instances of the course using the new methodology. This also applies to the total average of C2 (75.64%) compared with the 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. 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 (p-value < 0.001).

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

As seen in Figure 4, there seems to be a trend toward higher grade averages 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.

Notably, all differences are statistically significant ($p < 0.05$), although the difference between instances of C1 is only barely significant. Moreover, the difference between instances of C2 is much smaller than the 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

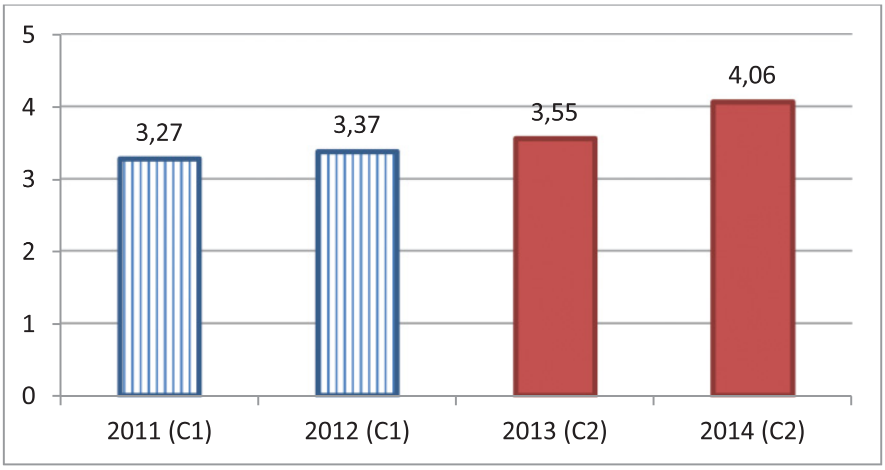


Fig. 4. Grade averages of all instances.

Table VI. 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

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, although 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 could complete them within the given one-week time frame. As this is close to impossible, 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 2013 and 2014 if they seemed too difficult or too easy.

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 difficulty: Some students reported the tutorial as too difficult and some too easy. Still, most feedback on the tutorials was positive. The feedback from tutorials could be roughly divided into seven categories. Some students thought that the tutorial exercises were too hard or that there were 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 though that the exercises were too easy or that there should have been more of them: *"Tutorial was short, it took only an hour to finish the exercises. Exercises were pretty straightforward."*

Most students had a very positive attitude toward 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 were students who did not see 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 though that the tutorials fit well with the 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."*

From a technical perspective, the tutorials worked quite well. The sessions were organized in a 250-seat lecture hall where the major concern was network connectivity. The hall's wireless router 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 student pairs, and the only thing needed was to supply the LAN cables. This also meant that 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.

Student collaboration on tutorials seemed to work very well. The discussion in the lecture hall was continuous, and, for the most part, students seemed to discuss the topic at hand. We 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 this issue, students were seated in every other row in consequent tutorials.

6.2. Demonstrations and Lecture Attendance

In the demonstration sessions, 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, students wished that the groups would remain the same when completing the additional assignment after the presentations because "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 later 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 being too laborious. Still, most feedback from the demonstrations was highly positive.

The students had a chance to attend mentoring sessions if the tutorial or the demonstration tasks seemed too difficult. Approximately 30–40 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

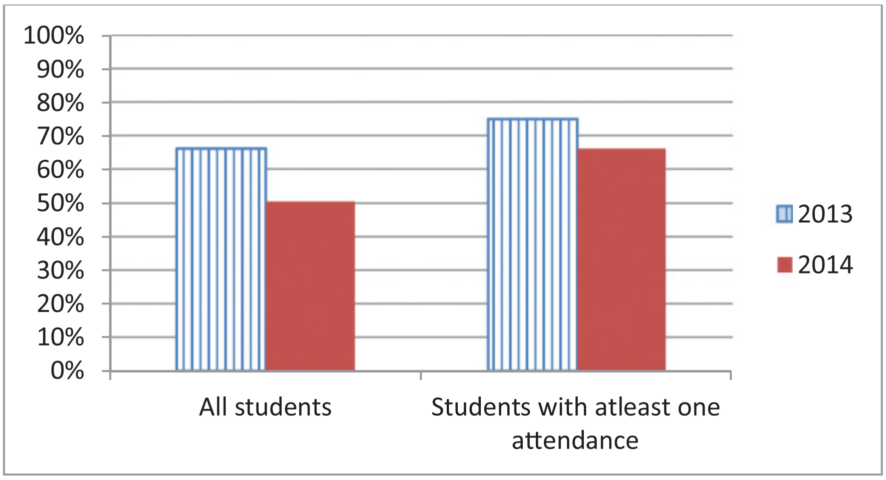


Fig. 5. The average number of lectures that students attended in instances of C2.

session to work collaboratively on the tasks—a practice that was highly encouraged by 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 reviewed, and the issues that were raised were 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 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. Students participated in the lectures quite actively, as seen in Figure 5.

Figure 5 shows the average of all students in attendance, as well as the average of students who attended at least once. 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 an actual correlation between two variables would require analyzing more instances. The overall high level of participation in both instances is probably partially due to bonus points that could be collected on lecture attendance. Still, the bonus at maximum was less than 0.17% 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 its features by 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.

All exercises except for 1 and 3 were coding exercises. As seen in Figure 6, there is a lot of variation in submission numbers. For noncoding exercise types, the average was close to once, 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,

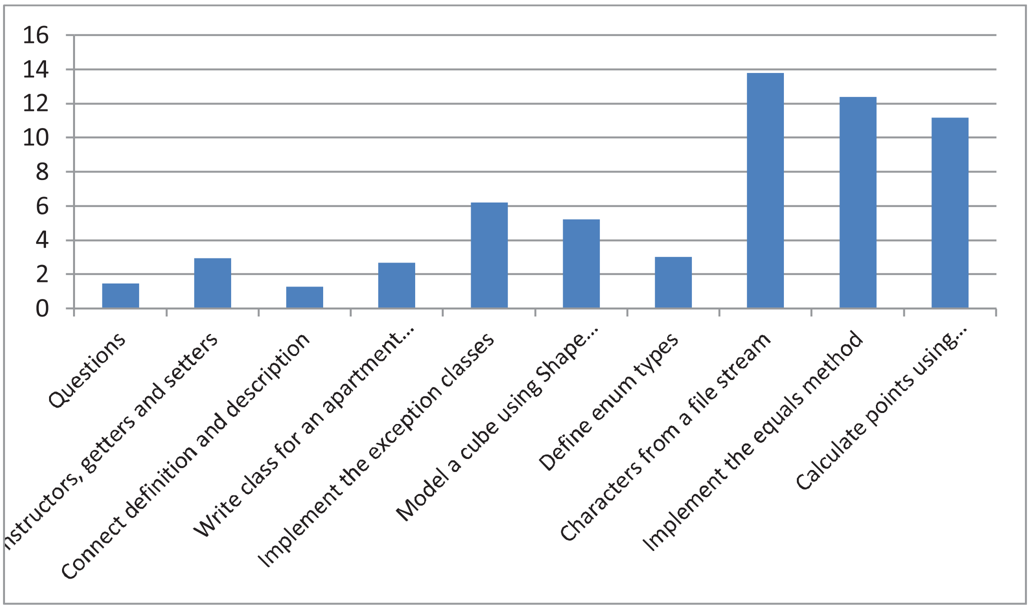


Fig. 6. Average number of submission made to exam questions in instances of C2 (combined).

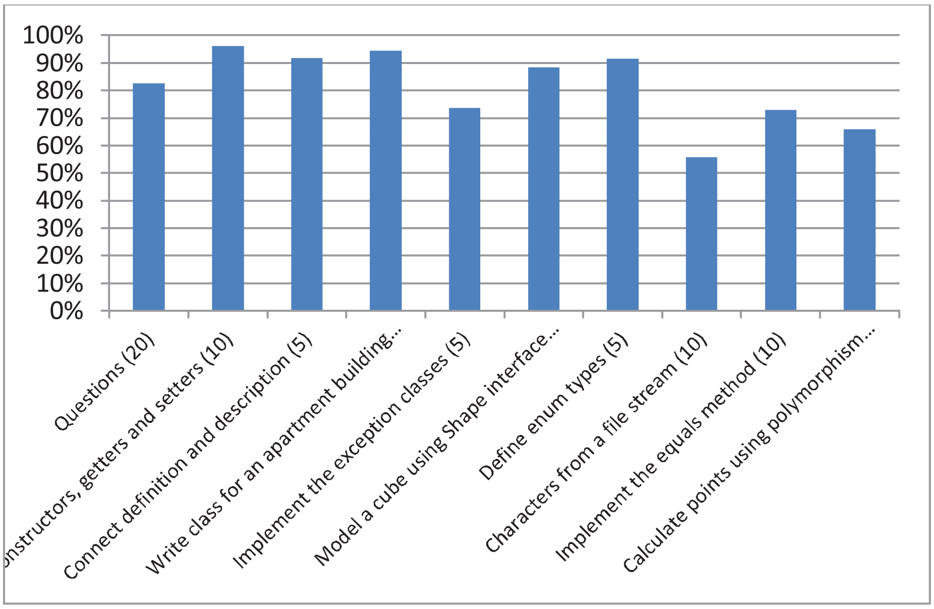


Fig. 7. Average scores for each question in exams of C2. Maximum points are displayed in parentheses.

the average number was more than 10. The average exam scores for individual tasks are displayed in Figure 7.

The students also answered a survey after the exam. There were 11 questions answered using a Likert scale of 1–5 (1 = completely disagree, 5 = completely agree). There were three exam instances each year. The results obtained after the first exam

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)

instance are displayed in Table VII. Averages and standard deviations (in parentheses) are displayed.

As seen in Table VII, 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, students seemed to think that the exam on a programming course should be taken in an electronic form instead of with 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, these 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 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 earlier studies. 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. Although the redesign described in this article cannot be categorized as flipped learning (see, e.g., Sarawagi [2013]), some similarities (such as emphasizing active learning) can be found. Although electronic exams in programming courses have not been studied widely, Barros et al. [2003] and Navrat and Tavrozek [2014] have had similarly encouraging experiences when they replaced a traditional pen-and-paper approach with a more authentic approach. Finally, the results obtained seem to confirm the results of our research group studying introductory programming courses (see, e.g., Rajala et al. [2009, 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 that the additional support received positive feedback from both students and 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 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 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 that (similarly to our experiences) applying this approach led 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. Although 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 course outcome. This was somewhat expected, as various other studies have already proved that active learning methods are more effective than passive listening in lectures. For example, the whole concept of flipped classrooms (although 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 level for the tutorials. Unfortunately, there is no strict advice on the difficulty level because the only way to properly evaluate difficulty is to test the tutorials with a large, heterogeneous group of students.

Second, enhancing collaboration seemed to work as we intended. As we found 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, we must suggest that fellow educators actively monitor the switching of the controller role once every 15 minutes during 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. Collaboration was also utilized in our demonstration sessions: We found out that the tasks prepared for this need to be very carefully planned beforehand because 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 to offer a small reward for filling them in (in our case, a couple of ViLLE points). Still, the most important suggestion we can make is that the results collected via survey need to be analyzed and utilized during the course. The students are likely to be more motivated to report issues using 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 (although 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 we considered. 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 nonaffiliated 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 (e.g., Sheard et al. [2011, 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 effectively as possible, preferably by whitelisting only those sites necessary for the exam. Also, supervision in the classroom where the exam is organized should be provided, as 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 undertaking complete refactoring at once because the individual steps can be utilized separately as well. In our experience, the redesign was worth the effort: The course's total pass rate increased significantly, and the number of drop-outs during the course decreased likewise. Student perception was also enthusiastic: 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 was done before the first class. Later classes are a lot easier to organize. Still, it is wise to prepare for adjustments after the first class because 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 the nature of the holistic redesign. First, there are the external concerns: The data from the earlier instances of C1 are not conclusive; for example, lecture attendance was not recorded, and detailed statistics or feedback about the exam was 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 in and general contentment with 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. The first limitation is the number of factors in the redesign: Since the changes were made in varied 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 had no effect at all. Also, the evaluation method differed. 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 very unlikely that passing the electronic version would be easier because the evaluation was stricter (e.g., 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 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 OOP course. After all, teaching interventions that are proved effective are what most educators should be looking for—hence, validating research should never be underrated.

9. CONCLUSION AND FUTURE WORK

We redesigned a programming course based on the best practices obtained from our own research and that of the research community. The redesign was done in four areas: by enhancing active learning, collaboration, and 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% 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 on the whole appears to be very effective, it is impossible to isolate the effects of individual factors because when course-long performance is measured, various variables affect it. Still, as the content of the course remained the same and the number of participants was fairly high, we feel confident about the significance of the results.

In future, we plan 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 continue fine-tuning the methodology and 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 *Proceedings of the Frontiers in Education Conference, IEEE.* 733–735.

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. 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. In *Proceedings of the 8th 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.

William Damon. 1984. Peer education: The untapped potential. *Journal of Applied Developmental Psychology* 5, 4, 331–343.

Enda Dunican. 2002. Making the analogy: Alternative delivery techniques for first year programming courses. In *Proceedings of the 14th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2002),* J. Kuljis, L. Baldwin, and R. Scoble (Eds.). Brunel University, London, June 18–21, 2002. 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. In *Proceedings of the National Academy of Sciences*. 8410–8415.

Lucia M. M. Giraffa, Marcia Cristina Moraes, and Lorna Uden. 2014. Teaching object-oriented programming in first-year undergraduate courses supported by virtual classrooms. In *Proceedings of the 2nd International Workshop on Learning Technology for Education in Cloud*. Springer Netherlands, 15–26.

Anabela Gomes and Antonio Mendes. 2014. A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations. In *Proceedings of the Frontiers in Education Conference (FIE 2014)*. IEEE, 1–8.

Joanna Goode and Jane Margolis. 2011. Exploring computer science: A case study of school reform. *ACM Transactions on Computing Education (TOCE)* 11, 2, 12.

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.

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)*. IEEE. 178–183.

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, 4 (2012), 1267–1281.

Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. 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. In *Proceedings of the 12th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2009)*, November 22–24. 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*, 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. In *Proceedings of the 17th Australasian Computing Education Conference (ACE2015)*. Sydney, Australia.

Hansi Keijonen, Jaakko Kurhila, and Arto Vihavainen. 2013. Carry-on effect in extreme apprenticeship. In *Proceedings of the Frontiers in Education Conference (FIE 2013)*. IEEE. 1150–1155.

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. In *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 to *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, 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. *ACM SIGCSE Bulletin* 34, 1, 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. 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. *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. 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. *Transactions in Computing Education* 11, 4, Article 24 (November 2011). 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). Australian Computer Society, Inc. 157–163.

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. In *Proceedings of the 3rd 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*, 7, 15–32.

Teemu Rajala, Erkki Kaila, Mikko-Jussi Laakso, and Tapio Salakoski. 2009. Effects of collaboration in program visualization. In *Proceedings of the Technology Enhanced Learning Conference 2009 (TELearn 2009).* October 6–8. 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. In *Proceedings of the Frontiers in Education 2011 Conference.* October 12–15. Rapid City, South Dakota.

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. *IEEE Transactions on Software Engineering* 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 7th International Workshop on Computing Education Research.* ACM. 33–38.

Judy Sheard, Simon, 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 17th 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*. ACM. 341–345.

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 10th Annual Conference on International Computing Education Research (ICER'14)*. ACM. New York. 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.

Received May 2015; revised February 2016; accepted March 2016

ISBN 978-952-12-3677-8
ISSN 1239-1883

Erkki Kaila

Utilizing Educational Technology in Computer Science and Programming Courses