Lappeenranta University of Technology
LUT School of Business and Management
Degree Program in Computer Science

Antti Herala

DEVELOPING AN INTRODUCTORY OBJECT-ORIENTED PROGRAMMING COURSE

Examiners : D.Sc. (Tech.) Uolevi Nikula
M.Sc. Erno Vanhala

Supervisors: D.Sc. (Tech.) Uolevi Nikula
M.Sc. Erno Vanhala

ABSTRACT

Lappeenranta University of Technology
LUT School of Business and Management
Degree Program in Computer Science

Antti Herala

Developing an introductory object-oriented programming course

Master's Thesis

2015

86 pages, 10 figure, 28 tables, 2 appendices

Examiners: D.Sc. (Tech.) Uolevi Nikula
M.Sc. Erno Vanhala

Keywords: Object-orientation, programming, reverse classroom education, Java, teaching, open data, CS2

The state of the object-oriented programming course in Lappeenranta University of Technology had reached the point, where it required changes to provide better learning opportunities and thus the learning outcomes. Based on the student feedback the course was partially dated and ineffective. The components of the course were analysed and the ineffective elements were removed and new methods were introduced to improve the course. The major changes included the change from traditional teaching methods to reverse classroom method and the use of Java as the programming language. The changes were measured by the student feedback, lecturer's observations and comparison to previous years. The feedback suggested that the changes were successful; the course received higher overall grade than before.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
LUT Kauppatieteet ja tuotantotalous
Tietotekniikan koulutusohjelma

Antti Herala

Olio-ohjelmoinnin johdantokurssin kehitys

Diplomityö

2015

86 sivua, 10 kuva, 28 taulukkoa, 2 liitettä

Työn tarkastajat: Dosentti, TkT Uolevi Nikula
Nuorempi tutkija, DI Erno Vanhala

Hakusanat: Olio-ohjelmointi, käänteinen luokkahuoneopetus, Java, opetus, avoin data
Keywords: Object-orientation, programming, reverse classroom education, Java, teaching, open data, CS2

Lappeenrannan teknillisen yliopiston Olio-ohjelmointi-kurssi oli saavuttanut pisteen, jossa sen oli muututtava oppimismahdollisuuksien ja sitä kautta tulosten parantamiseksi. Kurssi oli opiskelijapalautteen mukaan osin vanhentunut ja oppimistulokset eivät olleet niin hyviä kuin ne voisivat olla. Kurssin osa-alueet analysoitiin, toimimattomat osat poistettiin ja kurssia kehitettiin käyttämään uusia menetelmiä. Suurimpiin muutoksiin kuuluivat opetustavan muutos perinteisestä luokkahuonetavasta käänteiseen luokkahuoneeseen ja kurssilla käytettävän ohjelmointikielen vaihtaminen Javaan. Muutoksia mitattiin opiskelijapalautteella ja luennoitsijan huomioilla sekä vertailulla edellisien vuosien kursseihin. Palaute antoi viitteitä, että muutokset olivat toimivia, sillä kurssin saama yleisarvosana oli korkein tähän mennessä.

ACKNOWLEDGEMENTS

This thesis was done in Lappeenranta University of Technology and I'd like to thank the faculty for presenting me this possibility to construct a study from such an interesting topic. I'd also like to thank the examiners Uolevi Nikula and Erno Vanhala, especially Erno, who had the patience to aid the writing process and was always available to have a discussion.

I'd also like to thank my wife and daughter; without you this thesis might have not finished in time or at all.

I'd like to thank the Finnish Transport Agency for providing the funding for the project.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

CS1	Computer Science 1
CS2	Computer Science 2
DSRM	Design-Science Research Method
GUI	Graphic User Interface
IDE	Integrated Development Environment
OER	Open Educational Resources
LUT	Lappeenranta University of Technology
MOOC	Massive Open Online Course
OO	Object-Oriented
OOP	Object-Oriented Programming
OOPL	Object-Oriented Programming Language
OOPP	Object-Oriented Programming Paradigm
QML	Qt Meta Language
SVN	Subversion
UML	Unified Modeling Language
VLE	Virtual Learning Environment
XML	Extensible Markup Language

1 INTRODUCTION

This section presents the background for this study by assessing the language selection practice for programming courses and the popularity of programming languages. The object-oriented programming course currently in use and its prerequisites are introduced shortly. This section also provides the research questions and the overall structure of this thesis.

1.1 Background

Teaching programming and developing the education in universities tends to monitor what the software industry requires [1]. This has led to multiple programming languages and paradigms being taught in a large variety of courses in varied orders. The language decisions are done by comparing the pros and cons of local and global factors, such as the preferences of faculty and staff, current demands from industry, technical aspects of the language and available tools and materials. The language also has to fulfil the demands of the course and enable students to learn about the necessary topics in question. Surveys from the beginning of the millennium [2]–[4] show that C, C++ and Java have been the most widely used programming languages in the educational landscape and in 2007 this was still the state of the practice. [5]

The programming language preferences in current use have remained virtually unchanged from 2007. Languages such as Objective-C, PHP and JavaScript have been gaining popularity in statistics since 2010. The shift and current status becomes apparent from services providing statistics on programming languages, such as TIOBE [6], LangPop.com [7] and The Transparent Language Popularity Index [8]. The data for statistics is collected throughout the Internet and the results depend on the measurements and the interpretations, creating inaccuracies. While all these statistics services vary in their results, they provide similar outcomes where C, C++ and Java are among the five most used programming languages.

The programming languages in use and the reasons behind their use were surveyed in 2002 by de Raadt et al. [9] by concentrating on 39 universities in Australia. The research asserts

that Java was the most taught language out of all taught languages and along with C++ it was chosen practically solely for industrial demand (Java 70%/C++ 88%). In the same study a comparison between object-oriented languages shows Java as the predominant language to teach object-orientation, while C++ is applied more often to teach procedural than object-oriented programming. [9]

The object-oriented programming course in Lappeenranta University of Technology (LUT) focuses solely on objects and their interactions. In the course, object-oriented programming is taught to sophomores using C++ in open source environment. The students have studied Python and C, introducing procedural paradigm first and objects with object-orientation later. Since Java is a language that enforces object-orientation better than C++, it is justifiable that the course should be updated to use Java, educating students about object-oriented programming more efficiently and enabling students to use object-orientation concurrently with procedural programming. The industrial need of Java is also an important factor; students have to acquire skills with Java to be competitive employees in software industry.

1.2 Goals and limitations

The goal of this thesis is to create a new and improved object-oriented programming course with reverse classroom, that uses Java and open data. This improvement is achieved by creating video lectures new weekly exercises, a programming project and two manuals for students. The manuals consist of the basics of object-oriented programming and open data, but also basic Graphic User Interface (GUI) programming and basics of Extensible Markup Language (XML). The course is evaluated and compared to the previous course using weekly monitoring with a Virtual Learning Environment (VLE), teacher's observations, student feedback and course statistics. The statistics cover the course grades, the amount of work required from the lecturer and the number of returned projects, that were accepted without modifications.

Research questions are as follows:

1. What is the current status of object-oriented paradigm and open data in teaching?
2. How should a new course on OOP be constructed?
    2.1. What technologies should the new course use?
    2.2. How to change the course materials to answer the demands of students?
3. How the materials and the course compare to its previous implementation?
    3.1. How does the material support classroom teaching and self study?
    3.2. How students see the new course compared to the previous implementation of the course?

1.3 Structure of the thesis

The structure of this thesis follows the research questions. In the second section, the current practices of object-oriented programming and open data in teaching are assessed. The section also covers the theory behind the reverse classroom method. The third section presents the research method and how this thesis is built using it. In the fourth section, the state of the practice in teaching programming is presented. The section addresses the current status of teaching programming in Finnish universities as well as in notable international universities. The section ends with the presentation about the current object-oriented programming course in LUT. The fifth section covers the systematic improvements done to the course. The results are presented in sixth section and discussed in section seven. The eight section contains the conclusions of this study.

2 TEACHING METHODS

This section covers the methods for teaching object-oriented programming and using open data in education. The section also presents teaching theory, mainly by presenting the reverse classroom method.

2.1 Object-oriented programming and open data in education

The history of OOP can be considered a rather long one; the concepts of OOP are practically as old as programming itself, stretching back to 1960's. The first Object-Oriented Programming Language (OOPL) called Simula was developed between 1962 and 1967 for creating discrete event simulation models. Simula was later developed into a general-purpose programming language that used many object-oriented concepts: classes, subclasses and polymorphic functions. Another OOPL, Smalltalk, was developed by Xerox PARC in 1970's and is was the first programming language to introduce GUI development. All the components of a Smalltalk-program are objects and the interaction between objects is message passing, making Smalltalk purer OOPL than the currently popular ones, e.g. Java and C++. [10]

The concepts of OOP have been controversial since its creation, dividing opinions and implementations alike. Especially in the early days of object-oriented programming paradigm (OOPP), object-orientation had become a fashionable term and it was used to portray anything from abstract data types to organisational systems and business models that supported inheritance. Because of these different approaches the definite constructs of OOP are complicated to define. From multiple approaches and implementations the existence of objects as the primitive concept of OOPP is found to be an unifying actor. [11] There has been and still is multiple languages that claim to be object-oriented by nature and their development can be more closely explored through the work of Luiz Fernando Capretz. [11]

2.1.1 How to teach objects

The discussion about teaching object-orientation and two different methods, objects-first and objects-late, in introductory programming began when the objects-first model was introduced officially in Computing Curricula 2001 by ACM Joint Task Force [12]. The report listed six viable implementation strategies for introductory programming to substitute the then common programming-first model. The presented best practises were imperative-first, objects-first, functional-first, breadth-first, algorithms-first and hardware-first and the report suggested to use only those that can be scientifically assessed successful. The report recognises that the objects-first model does not remove all of the problems of programming-first model and adds more complexity, demanding wariness in the introduction of the subject. [12]

Discussion about the most effective method for object-orientation has been variant, especially in the last decade. The discussion has mainly revolved around the theme whether the objects should be taught in the programming curriculum as the first topic or is a traditional method, from procedural to objects, more beneficial [13]. The discussion culminated in 2004, when multiple highly recognised programming educators had a discussion about the matter via a SIGCSE mailing list (http://www.sigcse.org/) that did not result to any definite conclusion. What the discussion showed was that both methods have their own supporters in the educational community. [13]

Because of the controversy, the objects-first model has been researched thoroughly: its benefits have been measured [14], [15] as well as its success while being compared to objects-late approach [16], [17]. Some research also notes that the object-orientation is impossible to teach to students who do not have basic understanding of programming or lack the basics of mathematics and algorithmic thinking [18]. A research done by Reges [19] has reported that moving from objects-first model back to programming-first model has been successful when measuring grades and learning results [19].

The most significant findings reported by Ehlert and Schulte [16] were that there is only limited differences between the two approaches. The study used two concurrent courses with identical topics, one using objects-first and the other objects-late. The learning results

were the same on both courses but the difficulty of topics was perceived differently. The researchers minimised all the variables that might affect the results, including students' previous programming experience and teachers' different methods, producing as much data about the models as possible. It is indicated that the difficult topics, such as association are difficult, no matter what approach is used. [16], [17]

Kölling and Rosenberg [20] sketched guidelines for object-oriented programming courses. There has also been multiple instances of different guidelines, check lists and frameworks to use in teaching object-oriented programming. The selected guidelines have been made for introduction programming course in university and are also applicable in this case. They are optimised for BlueJ integrated development environment (IDE), even though most of them are not IDE specific. [20] The guidelines are presented in Table 1 and the selection of the guidelines is explained in Section 5.7.

Table 1. Guidelines for object-oriented programming course [20].

Guideline	**Description**
Objects first	Object should be taught as early as possible.
Don't start with blank screen	Students should start by making small changes into an existing program by using a code skeleton.
Read code	Students can learn by reading well structured programs and imitating styles and idioms.
Use "large" projects	The example programs should be large from student's perspective, so they can get the feel from the syntactic overhead and it does not feel excessive.
Don't start with "main"	The main()-function has no relation to object-oriented programming, it is only a point of communication from the language to the operating system.
Don't use "Hello World"	The problem with "Hello World" is common with main()-function. It does not present OOP and the use of objects is not clear for students.
Show program structure	The relation between objects and classes is the main issue of OOP, so the structure of a solution should be visually presented.
Be careful with the user interface	GUI is an easily distracting component of a program and not relevant to OOP.

2.1.2 Uses of open data in education

Research has introduced two primary methods how open data is linked to education: one is opening the educational data by the educational institutes [21], such as lecture materials/videos and course contents. The other method is to use open data in teaching from third party institutions, such as government agencies and other facilities [22].

The first method is usually called Open Educational Resources (OER), which is a part of a larger movement for openness, providing free access to teaching materials and resources without technical, price or legal barriers, in the limit of possibilities. The resources should also be available to adapt, build upon and reuse as long as the original creator is credited on their work. The subject and scale of material is dependant on the provider, ranging from the scale of video lectures of a course in YouTube [23] to a world-wide Massive Open Online Course (MOOC) [24]. The amount of material depends upon the scale of operation of the provider, where the provider can produce multiple courses through one portal, one custom made course through a common portal or just present information in free format. In Figure 1 the different possibilities are shown in a diagram with examples. [21]

MIT OpenCourseWare is a portal, where anyone can take courses from MIT for free. The OCW is discussed more later in this section. Wikipedia is a large storage of information for anyone to use freely and it is completely community based. University of Western Cape provides only five courses for free [25], which can be considered to be in a smaller scale of operations. Opencourse.org is a community-based portal that offers course materials through YouTube and Google Docs and are free to be modified by anyone [26].

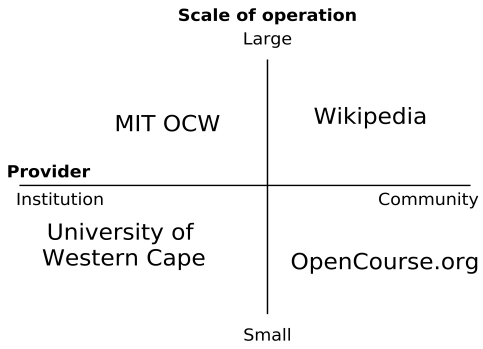

Figure 1. Educational resources providers [21].

Different institutions and government facilities have been collecting data and with the open model, they are starting to release their collections to public use. In the research done by Ferrara et al. [22] it is noted, that efficient education requires materials in addition to textbooks and supplementary materials. They decided to bring material through open data access from different museums, using it to teach more about cultural heritage. By using the data collections created and maintained by the museums instead of textbooks, the data is more likely to be correct and relevant. Teachers were also able to create lectures and materials directly from the data collections through a common platform, using the materials to support the lecture with content. [22]

The free and all-round data from the Internet is accessible to student every day and they can search relevant information with few keywords and clicks. While using open data, students should also be taught to filter the data and contribute to it as much as they can. [27]

2.2 Reverse classroom

The theory behind reverse classroom or inverted classroom has gotten foothold in the educational landscape since 2000. It was defined by Lage, Platt and Treglia: "*Inverting the classroom means that events that have traditionally taken place inside the classroom now take place outside the classroom and vice versa.*" [28 p. 32]. Their study was one of the first to present the idea of using multimedia and World Wide Web as a replacement of lectures. The idea behind reverse classroom is to allow students to study the theory and materials at home and the homework assignments to be done in classroom with the assistance of teaching staff. The method gives students more freedom to learn the theory based on the individual's own preferences and the teacher can concentrate on the desired outcome. The study argues that this method is ideal for any kind of individual learning style. [28]

The idea does not limit itself to the superficial change between lectures and homework but can be abstracted as Table 2 presents [29]. Reverse classroom can be defined as interactive group based learning activities in classroom and direct technology-based instructions for individual uses outside classroom [29]. Through the technology it is possible to grant easy

access to large number of information and especially in reverse classroom this should be adopted into education. By using technology to its full extent, the education can be reformed from memorisation to conceptualisation of knowledge. The costs are the downside for the use of technology, since the classroom should be equipped with the latest technology and the education would also rely on students' own devices. [27]

Table 2. Tasks in reverse classroom [29].

Inside class	**Outside class**
Questions and answers	Video lectures
Group-based/Open-ended problem solving	Close-ended quizzes and problem solving

A research survey conducted by Bishop and Verleger [29] goes through the history and uses of reverse classroom. The survey presented two related movements that have affected to the current educational landscape. The first movement, the technological movement, is related to the evolution of technology that overcomes the real, physical barriers to enable free flow of information. The second movement, the open movement, is trying to take down artificial barriers, legal or otherwise, imposed upon information. As proof of the combination of movements the survey concludes that video lectures slightly outperform the in-person lectures and online lectures are doing even better. The homework is just as effective whether it is done online or on paper and the intelligent tutoring systems can be as effective in teaching as human tutors. To create such systems for education has been remarkably expensive and the costs are being reduced through the efficient use of open movement. [29]

Continuing from the collaboration of the two movements and the use of reverse classroom, MIT made a significant advancement by releasing the OpenCourseWare [30] for public use in 2001. The portal published the materials that were originally available only to students who had paid the tuition fee. Following the example of MIT, the Khan Academy [31] was founded in 2006. Inspired by the advancements done by Khan Academy, Stanford professor Sebastian Thrun founded Udacity in 2011 [32] and with the support of Stanford university professor Andrew Ng founded his own, open online educational initiative called Coursera in 2012 [33]. MIT also produced their own collaborative open educational

initiative with Harvard, edX in 2012 [34] that offers the courses from Harvard and MIT for free. [29]

There are studies that show the difference between the traditional methods and reverse classroom. The research by Day and Foley [35] focused on the quality of online lectures in order to decrease the monologued in-class time and give students possibilities for interaction and active learning. They used the lectures to give students background knowledge of the material and spent the in-class activities to engage with the material. In the research the course was run concurrently with another course using the traditional methods and identical topics. The course consisted of homework, lecture homework, project and exam that all together formed the final grade. The study found that the reverse classroom resulted in higher grades than traditional method in every segment of the course. From the segments the final grade was reported to be significantly higher, traditional methods resulted in the average of 79.95 (scale from 0 to 100) and reverse classroom to 88.23. [35]

Another study by Mason, Shuman and Cook [36] reports similar results. In the study the traditional method was used in the first implementation and reverse classroom in the second implementation. In the traditional version the course consisted of weekly quizzes and a midterm, while the theory was taught in lectures where textbook examples were solved. In the second version the material was provided as video lectures that included instructor audio and live screen capture from a tablet for equations. The class time was then used to problem solving individually or in groups, using the same problems as in the first implementation. The effectiveness of reverse classroom was measured by comparing the course grades from different segments and student perceptions against the traditional method. The main findings of the research was that in the reverse classroom the instructor can cover more material, students performed as well or better in comparison, and students found the new method more satisfying and effective. [36]

In the research by Zappe et al. [37] the reverse classroom was experimented without a control group and the instruments for the survey were tailored. The study points out some guidelines for the use of the method [37]:

1. Require students to do an online quiz before in-class activities.
2. The videos should be relatively short, approximately 20-30 minutes long.
3. Go through a brief summary before the in-class activities.
4. Consider adding multimedia to video lectures to keep students interested.

2.3 Summarising the existing research

There has been multiple theories about how to teach object-orientation, about open data and with reverse classroom. The objects have been taught in the introductory as well as in advanced programming courses with different methods. Currently it appears that the objects-first method is used more in teaching, where the objects are taught from the first lecture onwards. The objects-later method allows the educator to begin with the data types and structures, the so called basic programming topics. While there has been multiple arguments for and against both methods, it has been found that both methods yield the same learning results and despite the method, difficult topics remain difficult.

The use of open data is rather scarce in the educational landscape. Most discussion about opening the data concerns the educational data, lectures etc. and not the use of open data. Few studies have noted that educators should use the open resources from other institutes to improve their teaching materials. This allows students to interact with the real world instead of the simulated world in class and enhance their voluntary advancement in the subject.

Reverse classroom is a pedagogical method where the traditional lecturing is are removed and replaced with interactive sessions. In addition it also includes homework that is done in-class in lieu of lecturing. The method gives more space to the students' individual learning patterns and increases communication and interaction between students and teacher.

The conducted literature survey found only few reports on the use of reverse classroom in teaching introductory programming courses [38], [39]. The slow adoption of the method might be caused by the large group sizes [28], since introductory programming courses

tend to have large groups to teach. Open data is not reportedly being used in programming courses as a base for the data sets.

3 RESEARCH PROCESS

In this thesis the Design-Science Research Method (DSRM) is used because it was found suitable for this study. DSRM is used in research that builds a new innovation from an old artefact with a well defined building process [40]. In this thesis the building process starts from an initial state, where the course is already in use but the course requires development. The goal of the research was already specified before the work began and the implementation was done by building the new course on top of the old one. The goal state is reached, when the new course is built, used and evaluated. The research process is described in Table 3. The thesis follows an inductive report structure suggested by Järvinen [40].

Table 3. Elements of research and uses in this thesis [41], [42].

Element	**Description**	**Uses in this study**
Problem identification and motivation	Define the research problem, where knowledge about the problem's state is required.	The previous course was found to be partially outdated.
Objectives for a solution	Define the objectives for a solution in the light of the research problem.	The techniques and materials are compared with mapped existing research.
Design and development	Design and create the artefact by implementing the necessary functionality.	The new material is made for the course and new techniques are considered.
Demonstration	Demonstrate suitably how the developed artefact solves the identified problem.	The new course is used to teach students the material.
Evaluation	Evaluate by observation, how well the artefact solves the problem.	Evaluation is done by student feedback survey and continuous evaluation by the educators.
Communication	Communicate the problem, solution and process to researchers and other audiences.	Communication is carried out by publishing the thesis by LUT.

The motivation behind this thesis is to systematically improve the old course because of the feedback from previous years and the Finnish industrial shift. The industrial shift from mobile operating system development to mobile gaming industry [43] established a decline in the need of C++ programming language. The programming language in question does not enforce the object-oriented paradigm and it can have a negative impact to students, who have previously programmed procedurally [44]. The previous research can be used to evaluate the current methods and materials of the course and decide whether the old aspects should be conserved or discarded. If the old aspects should be changed, they can be updated with previous research.

While the course is being revised, the pedagogical aspect of the revision also have to be considered, using material in the course that enforces the learning objectives, to give professionalism to students. The course is developed to use reverse classroom for teaching. This method tackles the problem that students have not participated to the lectures; physical lectures are removed and new lecture videos are uploaded online. The problem that students do not participate to exercises is tackled by giving points for demonstrating the exercise solutions in class.

The evaluation metrics to measure the success of the new course are the same as with the old one. The evaluation is done by student feedback and constant improvement cycle, where teachers and assistants evaluate and re-factor the material. Student feedback is collected through a survey form that covers all the materials and aspects of the course. The new course is compared to previous years with statistics of participation and learning results.

This thesis documents the process of improving the course with techniques such as reverse classroom and other methods from literature. Since literature about the reverse classroom in programming was scarce in the scientific channels, this research will add into that gap.

4 STATE OF THE PRACTICE

This section presents the current status of object-oriented programming education in Finnish and international universities and lastly summarises the previous course in LUT thoroughly. The section fills the second element of DSRM by providing the baseline for the change.

The most inclusive view about programming and object-oriented programming can be acquired by searching through the websites of the selected universities and presenting their current programming courses. In this study all the universities in Finland are selected and the best international universities are chosen from university rankings. The requirement is that the university offers a major in software engineering or computer science.

4.1 Finnish universities

The chosen Finnish universities are teaching software engineering or computer science as a major. Appendix 1 lists 11 of 14 Finnish universities, where it is possible to graduate as a Master of Science (M.Sc.) or a Master of Philosophy (M.Phil.) from software engineering or computer science. The selected courses for presentation are basic level programming courses that are available to bachelor studies. The curriculum in Finland is usually divided into four periods and are organised 2 periods in autumn and 2 in spring as presented in Appendix 1. In time of this comparison, LUT was still following the 7 week periods. The data is collected through the portals of the universities and the information about the courses is the responsibility of the university.

From Appendix 1 it is evident that most of the Finnish universities teach object-orientation in the first year. However most of them do not introduce it as the first paradigm; the introductory courses are concentrating to the imperative aspects of programming. While programming procedurally, almost half of the universities teach Java as the first language and continues to use it in the second course, where the objects are being taught, therefore minimising the transition problem [44]. The transition problem happens when the introductory programming course language is chosen solely based on the industrial need and students are taught a specific programming language instead of general programming

[44]. Another popular language is Python from Appendix 1, that is used to teach both procedural and object-oriented programming, but never in the same university. This could be caused by the versatility of Python and its lack of paradigm enforcement.

4.2 International universities

The international universities are selected using two ranking websites, QS World University Rankings 2013 [45] and Times Higher Education World University Rankings 2013-2014 [46].

The universities chosen have reached the top 20 position in both rankings and teach software engineering or computer science as a major, either B.Sc. or M.Sc./M.Phil. The international universities do not present as strict a curriculum as Finnish universities, so the table does not take into account the timing, when the course should be taken. However, the courses are still bachelor level programming courses, that have to be passed for graduation from computer science or software engineering. The universities and courses are presented in Appendix 2.

The curriculum of international universities has more diversity in introduction to programming than in Finland. While in Finnish universities the programming language choices are rather similar, the leading international universities teach multiple programming languages and paradigms. An example is the number of functional programming languages in introductory programming courses. In Finland there is only one university that provides an introductory course in Scala, while international universities provide introductory courses in ML, Haskell, Scala, Scheme and Racket. The differences about object-orientation revolve more around the diversity in the curriculum than the educational methods. In courses where the objects are introduced, the students use either Python or Java as the predominant language. There are some deviant approaches using OCaml, Eiffel or C++ but as Appendix 2 shows Java and Python are the most popular languages in use for objects.

The Appendices 1 and 2 show how students come in contact with Java and Python in the basic courses. Some of the courses do not teach object-oriented programming, but Java and

Python are still familiar programming languages in most of the selected universities and in their introductory programming courses. Some universities use different languages for object-orientation, such as OCaml, Eiffel or C++, but Java and Python are pedagogically easier languages to teach programming [47] while Java's enforcement of object-orientation makes it easier to use when learning object-oriented programming. Basic programming courses also use Java both with procedural paradigm [9] and object-oriented paradigm.

4.3 Object-oriented programming in LUT

The previous version of the course Object-Oriented Programming had multiple lecturers over the years, who emphasised different topics in the course. The main focus of the course is on object-orientation with C++ for second year students, who are required to have a basic understanding about programming. The two preceding courses are Introduction to Programming with Python and Practical Programming with C, where the procedural programming is the dominant paradigm. The previous course's goals for the students were as follows: "*Student learns to use object-oriented programming methods to solve typical programming problems and familiarizes himself with C++ and its features in programming. Student knows how to read and describe C++ code.*" [48 p. 143]

4.3.1 Course structure

The course has been 14 weeks long and it contained weekly lectures, exercises, one larger project and exam. The lectures were recorded during the physical lectures and offered to students online. The project was returned before the end of the course and exercises were scheduled to be returned weekly. The overall grade of the course consisted of the exam (60%) and project and exercises (40%). The exam was held traditionally, where students wrote their answers to a sheet of paper in a classroom. The project was returned as an attachment of an e-mail and part of the weekly exercises were returned into the virtual learning environment (VLE), the rest were checked by the teacher. The VLE serves as an automatic testing platform, where students can insert their code and the VLE simulates input if necessary and validates the output. The exercises and lectures were identical every year, but the project and exam varied each year by covering the same topics with alternating assignments. The course syllabus is presented in Table 4.

Table 4. Course syllabus for the old course in LUT.

	Topic	**Content**	**Video length**
L1	Introduction	Introduction to materials and tools, history of programming, object-oriented paradigm, from C to C++, introduction to objects and classes	50:49
L2	C++ and objects	History and basics of C++, creating the first class and turning it into an object, constructor, methods and attributes	55:28
L3	Objects and classes continued	Basics of classes, constructor and destructor, introduction to compiling process	1:02:32
L4	Variables and pointers	Variables, constants, data types and pointers	1:05:30
L5	References and functions	References, functions, return values, default parameters, overloading, inline-functions, recursion and STL vector	55:04
L6	Object-based design	Version control, basics of UML, class diagram, inheritance, aggregation	1:28:10
L7	Inheritance, part 1	Visibility modifiers, inheritance in practice and overloading	1:00:03
L8	Inheritance, part 2	More about inheritance, virtual functions, abstract classes and multiple inheritance	58:28
L9	Interfaces	this-pointer, inheritance in constructor and destructor, abstract classes and interfaces, their uses and type casting	1:11:43
L10	Copying and assignment	Copying and assignment, redefining operators, serialisation and friend-keyword	1:21:36
L11	C++ techniques: namespaces, class attributes and operations, GUI	Operator execution sequence, namespaces, class variables and methods and graphical user interface with Qt	1:02:16
L12	Exceptions and error handling	Errors, exceptions, exception handling, auto pointers	59:46
L13	C++ models, STL	C++ models and STL	1:13:43
L14	Recap	Quick revision about course materials and hints to the exam	45:57

The course has been systematically improved since 2010 when the lecturer has remained the same. The development steps taken during this time are presented in Table 5.

While the course was running, the students were offered both local instructions as well as online materials. The VLE offered light sets of key points about the exercises. By using that material the students were able to study with text concurrently with the lectures. The environment and lectures were available for students anywhere. This offered the students more freedom to complete and return the tasks and the presence in the exercises was not required.

Table 5. Development of object-oriented programming course at LUT in 2010-2013.

Year	**Changes**
2010	New lecture slides and programming examples were created and the lectures were recorded. The lecturer used Linux operating system with a simple text editor, but the students made exercise solutions in Windows.
2011	The exercise solutions were done using Linux and the simple text editor was replaced with NetBeans IDE in both lectures and exercises. Students were also given the freedom to choose different IDE they were comfortable with, if they wanted to. A GUI example with Qt was included into the course. The lectures were offered in video format and also in mp3-format. Non-mandatory essay assignment was added to the course as a bonus.
2012	The course remained the same as the previous year.
2013	The recordings of the lectures were uploaded into YouTube thus the download became unnecessary and students could watch the lectures with any device. The GUI example was expanded from previous years and students were introduced to version control Git (http://git-scm.com/). The use of version control was not mandatory, but it was awarded with extra points. The project was possible to be returned by sharing the Git repository with the lecturer.

4.3.2 Course feedback

The course feedback has been consistently above average presented in Table 6 and the course has received awards from the university (Best Teaching in University 2013) and from student union (Course of the Year 2012-2013). The university awards a course based on quantitative attributes while the student union's award is considered to measure qualitative attributes as well. In the course the average number of students with final grade has been 31 per year that makes the number of received feedback acceptable.

Table 6. Course average grade from students. (1=lowest, 5=highest)

	2010 (N=19)	**2011 (N=18)**	**2012 (N=19)**	**2013 (N=15)**
Average grade for course	4.00	3.94	4.37	4.36

The different elements of the course were measured using the feedback from the students through a survey form. The results are presented in Tables 7 and 8. Approximately 50% of the students, who got a passing grade gave feedback.

Table 7. The easiness of course's components (1=hard, 5=easy)

	2010 (N=18)	**2011 (N=18)**	**2012 (N=19)**	**2013 (N=14)**	**Average**
Lectures	4.12	3.61	3.63	4.00	3.84
Lecture slides	3.82	3.77	3.68	4.14	3.85
Lecture recordings	3.63	3.56	3.83	4.15	3.79
Example programs	3.59	3.56	3.32	3.86	3.58
Exercise events	3.29	3.00	3.16	2.23	2.92
VLE-tasks	3.77	3.17	3.16	3.93	3.51
Programming project	3.29	2.72	2.58	2.64	2.81
Course literature	3.18	3.00	2.88	3.23	3.07
Manuals in the Internet	3.77	3.13	3.37	3.31	3.40

Table 8. The usefulness of course's components (1=useless, 5=useful)

	2010 (N=18)	2011 (N=18)	2012 (N=19)	2013 (N=15)	Average
Lectures	3.94	3.65	4.11	3.87	3.89
Lecture slides	4.06	3.53	3.68	3.80	3.77
Lecture recordings	3.24	3.82	4.17	4.36	3.90
Example programs	3.88	4.35	4.21	4.20	4.16
Exercise events	3.41	3.50	3.79	3.36	3.52
VLE-tasks	3.89	4.24	3.68	3.67	3.87
Programming project	4.06	4.29	4.47	4.67	4.37
Course literature	3.35	3.54	3.00	3.57	3.37
Manuals in the Internet	3.88	4.06	4.11	4.20	4.06

From the feedback, few observations can be made. While the project is considered the most difficult component of the course (average), it is also the most useful component.

Another noteworthy aspect is the increasing usefulness of the lecture recordings, while the yearly average of the lectures does not express a clear direction. The lecture recording had some technical issues in the first two years, which had an influence on the resulting increase in average. In addition to mentioned issues, there was no clear indications about the different feedback between lectures and lecture recordings, although the contents were identical. It can also be seen from the average that the example programs and manuals in the Internet are being seen as among the most useful components for students.

4.3.3 Problems in the course

Other than the components, there has been four distinct elements in the course that require change. The elements have been gathered by using the open feedback from students and observations from the lecturer about the course. The elements are listed next and each element is further described after the list.

- Students do not participate in the weekly exercises
- Students do not have the required skills to use tools for teamwork
- Students do not participate in the lectures
- Exam on paper

The most stressing problem was the limited student participation in the weekly exercises. In the feedback multiple students admitted returning only the mandatory VLE-tasks and project, ignoring other exercises completely. The exercises exist to offer students the chance to experiment with the introduced techniques from the lectures and prepare for the assignment programming-wise. Since the students did not participate in the exercises and did not complete the voluntary tasks any other time either, they lacked the skills required for the assignment and had to redo it at least once. The redo percentage is presented in Table 9.

Table 9. Returned and redone projects.

Year	Projects returned	Redone projects	Redone percentage [%]
2010	30	14	46.67
2011	33	28	84.85
2012	24	17	70.83
2013	22	14	63.64

In the course the use of pair programming and teamwork tools is permitted and encouraged as can be seen from the number of students who completed the project from Table 10 versus the returned projects from Table 9. Pair programming has caused a habit among students where they share their source codes and documents via e-mail. Sharing via e-mail is not encouraged while other practices, such as version control, are recommended. The previous course introduces the concept of version control, where students use Subversion (SVN) as their primary group tool. While version control is introduced before, there is a possibility that students have not assimilated its best practises and lack the skills for its use. This course is also open for students through other channels, who have not yet completed Practical Programming, decreasing the expectation of previous experience with version control.

Table 10. Students enrolled and completed the course.

	2010	**2011**	**2012**	**2013**
Enrolled to the course	69	68	56	54
Started the course (achieved at least 1 point)	46	51	42	38
Completed the project	29	41	31	28
Completed the exam	28	41	31	28
Got a grade	28	39	30	26

Another problem is the number of students participating to lectures. As an example, in late 2013 the lecture had only two participants and neither of them asked questions. This caused the lectures become a presentation by the lecturer without any interaction. One possible cause for this is the lecture recordings, that make it possible for the students to watch the lectures without physically participating to them. Evidently, when receiving more freedom the students discard the components in the course they do not need in their opinion.

The exam contains at least one programming task and is done on paper. Programming on paper is unreasonable from the point of view of both student and teacher, since students cannot test or debug programs but they have to memorise the syntax. Memorising the syntax does not test how well students have understood the concepts of object-orientation. Programming without a computer or a possibility to compile, run and test the program requires more experience from the author than the course allows in its time frame. This task has remained in the exam as a proof that every student who completes the course can produce readable source code. The exam itself has remained as a part of the course, enabling the assessment of students' level of understanding about the topics of object-orientation. The programming on paper and the exam has led to complaints from students, who regard it as unreasonable and unnecessary.

The weekly exercise tasks that have been a part of the course are tested and assessed by the VLE. This method has granted the lecturer freedom from manually asserting every task from every student but also creates a lack of interaction. By automating the return process, students have grown accustomed to creating a program that is validated by the VLE but the solution is not necessary object-oriented. In object-oriented programming this model of

assessment is not sufficient. Students should instead be encouraged to understand the topic first in addition to getting acceptance from the VLE.

5 IMPROVEMENTS

The improvements for the course were made by considering the feedback from the students, lecturer's observations and best practices from the literature. While the literature usually focuses on teaching objects-first model in Computer Science 1 (CS1), the notions and suggestions are relevant in advanced object-oriented programming course as well. From the literature the suggested syllabus for OOP [16] is similar to the syllabus of the previous course. The improvements serve as the design and development element of DSRM.

A major change that occurred in the university was the decrease in the number of weeks in a period from seven to six, which decreased the total number of course weeks by two. The periods were shortened to give more time for intensive teaching and exams. Taking two weeks away from the syllabus forced changes to the course structure. Instead of just shortening the course, a decision was made to rebuild the course completely from bottom-up, developing and improving it with feedback from previous years.

In this section, all the parts of the previous course and the new course are compared and the changes are explained. In the section, the selection of programming language of the new course is explained, moving to programming environments, teaching materials, exercises and project. The course itself is a small ecosystem, where all the parts make a whole, so the final decision is made after all parts have been presented and evaluated.

The learning outcomes are defined as: "*Student learns to use object-oriented programming methods to solve typical programming problems and familiarizes himself with Java and its features in programming. Student knows how to read and describe Java code and UML diagrams.*" [49 p. 139]

5.1 Languages

For the language evaluation, the metrics are selected from existing literature. The metrics were considered to cover either any programming language education [50], [51] or only object-oriented languages [44]. Considering that the course in question is an advanced

programming course for students, who have previous experience about programming, it was decided that the metrics need to be optimal for the course in question instead of common programming education metrics. As a result the object-oriented language metrics were selected.

For the languages of object-oriented programming education, the requirements sketched by Michael Kölling [44] will be used to evaluate the languages. The requirements are listed and explained in Table 11.

While some of the requirements are evidently meant to protect beginning programmers from difficult data structures and techniques, the table is mostly relevant to more advanced programming courses. For a third programming course, object-oriented programming, the considered languages have to pass at least one of the following requirements:

- The language should have been used by students before the course [1].
- The language should support object-orientation as much as possible [44].
- The language should have a considerable market segment in the industry [1].

Considering these requirements, six programming languages were selected for comparison. The first requirement yielded Python, that students have used as their first programming language and C++, a natural continuum for students proficient in C. The other requirements gave four object-oriented languages to consider: Java, C#, Smalltalk and Eiffel. For example JavaScipt was rejected, because it is a prototype language, PHP is mainly a server-side language and Ruby's market share was considered too low.

Table 11. Requirements for programming language in object-oriented education [44].

Requirement	Explanation
Clear concepts	The concepts in the course should be easily used in the language. The language should be implemented with enough abstraction so it does not contradict with the teaching.
Easy transition to other languages	The language students learn in university should be a language that enables them to understand programming concepts instead of any exact programming language.
High level	Tasks available to be executed by the compiler or the runtime environment should be automated and not be left as the priority of the programmer.
No redundancy	The language should have one practical solution per problem. Different mechanisms to solve the same problem add confusion.
Pure object-orientation	The language should only support object-orientation and should not permit other programming paradigms to be used.
Readable syntax	The syntax should be easy to read and comprehend.
Safety	Safety means that the errors can be easily and automatically detected. Detection only is not enough, the messages should be easy to understand.
Simple object/execution model	The execution should be easy to understand step by step, even for a beginner. This should hold true for object allocation as well.
Small	The language should be as compact as possible, but still powerful enough to be usable. Larger languages tend to have multiple subsets that must be eliminated by the educator.
Suitable environment	The environment should support multiple operating systems and the object-oriented paradigm. By right environment the focus can be directed into the programming and paradigm instead of the features of the environment.
Support for correctness assurance	Students should be aware of the software engineering principles. The techniques in use are clearer if they are supported by the language and not by an external style guide. The language should also support pre- and post-conditioning.

Python

Python (https://www.python.org/) is a well-known scripting language usable with procedural, functional, object-oriented and aspect-oriented paradigms. The language constructs from modules, exceptions, dynamic data types and classes and it has a large standard library for multiple areas, such as string processing. Python also supports operating system interfaces, Internet protocols and software engineering principles. [52]

Python is a language, that covers most of the previously defined requirements in Table 11. It has quite clear concepts, it is a high level language, it offers readable syntax, it is safe, the execution model is simple, it offers it own cross-platform environment (IDLE) and it is also supported by many IDEs. Lastly, the language supports software engineering principles, it has encompassing modules for unit testing, logging and profiling. Python is also used in CS1 at LUT, so the syntax and structure is familiar to students.

The fact that students have already used Python is its strength but also its greatest weakness. Since students have already used Python to program procedurally, the transition from procedural to object-orientation is challenging [53], leading to solutions that do not necessarily follow object-orientation. Especially because Python does not enforce the paradigm. Python does not provide transition to other languages without problems, since Python is such a versatile language with flexible features. For example, Python is a weakly typed language, resulting in possible problems for novice programmer in any strongly typed language.

C++

C++ is a programming language for multiple paradigms and it was originally constructed as an extension to C. This is why the language consists of both high-level features as well as low-level operations, to enable the language to both beginners and advanced C programmers. C++ supports multiple paradigms, such as procedural and object-orientation but also generic programming. While the programming language was developed relatively early in the computer age in 1980s, it is still popular in education and industry as can be seen in Section 1. [54]

When considering C++ as the language to teach object-orientation in the light of the requirements from Table 11, C++ fails to meet almost every one of them. However, it does meet one of the most important ones, the suitable environment. C++ has been in the market for so long that there has been time to develop a generic platform for it. C++ can be programmed and executed practically in all environments and IDEs in the current market and the software engineering principles, such as pre- and post-conditioning, have also been integrated into the language. C++ was also one of the most popular languages is the 1990's and multiple languages have copied it's syntax, rendering it a logical stepping stone for programmers to learn other languages. The students have also previous experience with C, making the language a considerable alternative for the new course.

While being a popular and powerful language, C++ has multiple problems. It does not present any abstract concepts clearly, especially because it has been extended from C. It requires developer to understand multiple subtle concepts before implementing basic constructs. Another negative component of C++ comes also from C: the lack of automated memory management. The language requires the developer to remember all the allocations of objects in case memory must be freed. Managing memory and removing unused objects is a task suitable for the runtime environment. C++ is a hybrid language, providing low-level functionalities and it has a complex object and execution model, mainly because of pointers. The syntax of C++ is derived from C, where the syntax can be difficult to read and understand. It repeats the code by requiring a header file concurrently with a source file. [44]

Java

Java is an object-oriented programming language based on C and C++, but it was created to be small and simple. Java is superficially very similar to C++ in syntax, but some of the features of C++ were removed systematically, to make Java easier to use and understand. Because Java was designed and built on an existing programming language and systematically improved, the learning curve for programmers from other languages can be considered shorter than usual [55]. The language was developed in the mid 90's, but it is one of the most popular programming languages, as can be seen in the language popularity polls mentioned in section 1.

Java is a programming language, that has spread far and wide in the industry and in classrooms. Java was the first language to produce applets inside web browsers, it was the most platform independent language when it came out and it was marketed to be “the better C++”. All these factors made it the leading programming language in a fairly short time. [44]

This however was not without merit from the language itself. Java was created on top of C++ and because of the development process it has become a safe and platform independent language, that can be easily used in software development without major changes. What is especially prominent in Java is its object model and how it handles the object-oriented concepts, such as abstraction and inheritance.

While Java can be seen as a language that solves most of the problems in the area of programming, it still holds some problems. Although most of the object-oriented concepts are presented in a simple way, it uses rather complex concepts early, especially in the main()-function. The function uses keywords public, static and void where the first two are complex entities not required to be understood in the beginning of the course and the third does not have anything to do with object-orientation. Another problematic concept is the use of primitive data types and their wrappers, resulting for example in a data type boolean and a class Boolean, that are considered the same. This problem exists for all primitive data types and causes confusion and mismatches. The biggest problem of Java is the syntax. Syntax is a major hurdle in C++ and it is a problem for Java as well, leading to superficially complex code that is hard to read and understand. [44]

While Java is usually being called a pure object-oriented language, the primitive data types mentioned earlier disrupt this definition. The primitive data types make Java complicated to call a pure object-oriented language, because they are not objects. In can be argued that Java can be used without the primitives by using the wrapper classes, but their existence is against the basic principle of object-oriented programming. Object-orientation requires the existence of objects as the primitive concept.

C#

C# is a programming language developed by Scott Wiltamuth and Anders Hejlsberg from Microsoft. It uses C++ and Java as it's most influential predecessors, abandoning from both languages the features that were categorised too complex and unnecessary for the new language. C# is practically the purest of the popular object-oriented language currently after learning from Java's mistakes and misconceptions. The language itself has been developed to other direction than the enforced object-orientation and it supports multiple paradigms, such as imperative, structured, functional and object-oriented paradigms. C# is also one of the official languages that supports the Microsoft .NET framework. [56], [57]

C# takes most of its basics from Java, and by doing so it inherited most of the strengths of Java. It is a high-level object-oriented language with a more readable syntax than Java. The two languages have multiple common concepts and the most beneficial are automatic garbage collection, also known as automatic memory management, safety of data types, enforcement of methods and variables to classes, runtime check on type casts and interfaces. C# has also removed one of the major problems from Java, the wrapper classes and primitive data types. The data types in C# are all objects by default with namespace masking them, so the objects can be named logically for the environment, but the namespace makes it simple for the programmer. [58]

While C# has fixed the problems in Java and C++, it also has inherited features from C++ that are not beneficial in a modern language. C# makes it possible to use pointers in object-oriented programs in a high-level language, again shifting the memory management to programmer, while actually using automatic garbage collection. Programmer can now allocate memory, that the garbage collector does not remove until the pointer is removed, easily leading into memory leaks. C# also inherited Java's complex way on introducing the main()-function, bringing unnecessary amount of concepts in early on. [57]

C# also contains multiple structures from C++, such as structs. While they do not function exactly like the original, it leads into the fact that C# is a rather large language with multiple different types of classes. As one of the official .NET languages, C# relies heavily

to Microsoft's platform and does not provide that much support to other platforms and operating systems.

Smalltalk

Smalltalk is an object-oriented language, that usually refers to Smalltalk-80, the first version of Smalltalk to be released for public use. The language's first version was released to only a number of private companies (e.g. Apple Computer) and few universities for peer review. The second version was made more publicly available for use with specification. The ANSI Smalltalk specification has been a standard language reference since 1998. [59], [60]

The most beneficial feature of Smalltalk is that it uses only objects and messages to build a properly running program. This leads the developers to use only objects and the language can be effortlessly used to teach object-orientation, since other paradigms are not viable in the language. Another major benefit in Smalltalk is its use of generics, since Smalltalk does not use any data types and everything is generic. Smalltalk also provides dynamic bindings, so the errors can be fixed while debugging and there is no need for type checking. [61]

The drawbacks of Smalltalk are the clean concepts, since the less important features of the object-oriented paradigm are not implemented adequately, e.g. implementation and interface distinction. Also, in Smalltalk all methods are automatically public. One positive feature is also a drawback, since Smalltalk lacks static typing, making it unsafe for students to use. The lack of static typing makes the programs harder to debug and it causes a lot of different problems. Other drawbacks of Smalltalk are its syntax, which cannot be considered readable and its size. Despite the name, Smalltalk is actually a very large language and may cause the students to feel overwhelmed. [44]

Eiffel

Eiffel is an object-oriented programming language developed by Bertrand Meyer in the beginning of 1990s. While developing the language Meyer took multiple references from

C++ and Java as well as Objective-C and Smalltalk. None of these languages developed into the direction he would have wanted [62]. Eiffel as a language strives to be completely object-oriented and it provides the programmer a seamless transition to languages like Java, C++ and Smalltalk, because of its development history. One of Eiffel's major development requirements was the effortless adoption of the language by beginners. The development circled around the fact that cryptic symbols should not be used for the sake of syntax. [63]

Eiffel is a language, that has been designed for object-orientation and education only, not for industrial needs. This is why the language supports the concepts in a clean way, making the language easier for complete beginners. Eiffel also avoids redundancy, it is statically typed and the syntax is clear and readable. Overall, Eiffel is one of the most prominent languages in this comparison but it is not without limitations. [44], [63]

Eiffel is a language that has clean concepts with few exceptions. In Eiffel it is possible to store objects in normal mode (reference) or in "expanded" mode (value), which may cause confusion. This causes the programmer to contemplate the differences in the design stage. The object model of Eiffel is unnecessarily complicated because of the previously mentioned modes and it leads into overly complex structures. While the language has no excessive redundancy, it still supports a large number of constructs. The sheer amount of the constructs makes the language hard to approach and adopt. The environment of Eiffel is also one of the major hurdles of the language. The libraries of Eiffel are much too professional and specialized for introductory course, overcomplicating the concepts in the course. [44]

Language comparison

Each of the languages were validated using the requirements by Kölling [44] from Table 11 and the results can be seen in Table 12.

After a careful comparison of programming languages, it is evident that the best solution would be Python, Java or Eiffel. Considering how students have already gained a procedural experience with Python, it is a safer solution to use programming languages that

enforce object-orientation. A definite decision between Java and Eiffel can be made by comparing the programming environments and seeing, which language could be more suitable for the ecosystem of the course. Java has a slight advantage, since it is more relevant to industry than Eiffel.

Table 12. Mapped languages. (x=covers fully)

	Python	**C++**	**C#**	**Java**	**Smalltalk**	**Eiffel**
Clear concepts	x					
Easy transition to other languages		x	x	x		x
High level	x		x	x	x	x
No redundancy	x		x	x	x	x
Pure object-orientation					x	
Readable syntax	x					x
Safety	x			x		x
Simple object/execution model	x		x	x	x	
Small	x			x		
Suitable environment	x	x		x		
Support for correctness assurance	x	x	x	x	x	x
Σ	**9**	**3**	**5**	**8**	**5**	**6**

5.2 Environments

The selection of environment for the course is practically as important as the programming language [64]. The environments are being evaluated based on Kölling's [64] requirement sketch, that is presented in Table 13. The IDE requirements were selected, since they are an extension to Kölling's language requirements [44].

Table 13. Environmental requirements [64].

Requirement	**Explanation**
Ease of use	The environment should be simple to an inexperienced student to use but usable enough for advanced programming tasks. It should also hide all unnecessary details and allow user to work on a high abstraction level.
Integrated tools	The environment should support integration of tools from other providers, such as third party plug-ins.
Object-support	The environment should give support to objects; objects should exist in runtime. Objects should exist independently and they can be used for operations.
Support for code reuse	The environment should provide a browser where the user can see a list of the available classes. It should also support library creation.
Learning support	The environment has to support techniques for learning. These are interaction/experimentation and visualisation.
Group support	The environment should support group tools, such as version control.
Availability	The environment should be available for educational institute and student without cost.

The programming environments are developing continuously and new environments are being made and published regularly. This is the reason why in the comparison it is necessary to limit the number of environments with the criteria of what is essential for the course. Selection criteria can be passed by IDE, that:

1. support at least one of the languages discussed before.
2. are popular.
3. support the most popular operating systems (Windows, Mac, Linux).

The first and the most important criteria is the support for the programming languages, since an environment is useless if it is impossible to program with the designated language. By taking into account only the environments that can be used (language and operating system support) and measuring their popularity, the final list is composed. As for the popularity metrics, Google Search is used. The name of the environment is searched and

with the number of search hits the popularity can be approximated even if this method is slightly inaccurate. The results are presented in Table 14.

The environments were chosen for the table with the criteria mentioned before. This however does not cover all environments in the market, but it is necessary to limit the amount considered. From Table 14, five most popular IDEs (most hits) are chosen for closer examination and mapping against the requirements.

Table 14. IDE search results from Google.

IDE	**Keyword**	**Number of hits**
Eclipse	"Eclipse IDE"	699 000
Netbeans	"Netbeans IDE"	677 000
IntelliJ	"IntelliJ IDE"	134 000
MonoDevelop	"MonoDevelop IDE"	28 100
Qt Creator	"Qt Creator IDE"	21 600
Geany	"Geany IDE"	10 300
BlueJ	"BlueJ IDE"	5 970
Python IDLE	"IDLE IDE"	4 720
Squeak	"Squeak IDE"	1 800
EiffelStudio	"EiffelStudio IDE"	610

Eclipse

Eclipse's (https://www.eclipse.org/) production began in the late 1990s by IBM. In 1998 the company started to create a programming platform: the production began as a new Java IDE and a broader platform for it. The core of the IDE was assigned to experienced developers, while the company tasked additional teams to build new products on top of it. The ecosystem for third parties was found critical for the competition but the company did not receive any interest from the investors. This led IBM to adopt open source licensing and formed a consortium with eight other companies, both partners and competitors. The principle behind the consortium was to allow the open source community to control the

code, while the marketing and commercial relations were handled by the consortium. The principles in Eclipse's case have been successful and the Eclipse Foundation currently consists of twelve developer member companies and the principles have propelled Eclipse into the market as a major IDE, as can be seen from Table 14. [65]

Currently Eclipse, through its plug-ins, has support to multiple languages other than Java, such as C/C++, Python, JavaScript, PHP and other Web languages. Eclipse, because of its large base of plug-ins and features is one of the most popular IDEs in use, no matter the language or project. [66]

Eclipse is one of the most ideal IDEs for programmers, since it is an easy-to-use platform that supports multiple tools from other developers. Eclipse can also differentiate between objects, it supports group tools, is open source, and free to use. Eclipse also provides documentation through the IDE with easily browsed class libraries.

A drawback in Eclipse is the educational support which is up for debate. The environment has a plug-in for BlueJ (http://www.bluej.org/), a visual learning environment for object-orientation. The plugin can be used in teaching and visual demonstration about objects and their states, but it is not compulsory for Eclipse and easily missed by students and teachers.

Netbeans

Netbeans (https://netbeans.org/) started out as a student project named Xelfi. The goal of the project was to develop a Java IDE with Java that would imitate Delphi (http://www.delphibasics.co.uk/) to some extent. It was the first Java IDE to be developed with Java and the first pre-release was done in 1997. The project started while the developers were students but was interesting enough and the developers started to market it after graduating, forming a company around it. The original plan was to develop network-enabled JavaBeans components, hence the name Netbeans. While the project did not follow the original plan, the name stuck anyway. The Swing support was added in 1999, when the first version, Netbeans DeveloperX2, was released. In 2000 and 2001 most of the development work of Netbeans went towards generic desktop application environment. Netbeans was also bought by Sun Microsystems in 1999 and later in 2010 it was passed to

Oracle, who supports the IDE to date and releases a new version with every major release of Oracle Java. [67]

Netbeans is an IDE that support multiple languages, not just Java. The list includes C/C++, PHP, JavaScript, HTML, XML and Groovy. What makes it prominent with Java is that it is the official IDE for Java and it is supported by Oracle. The IDE also contains some automation for code debugging, since it can fill the missing brackets, match words, suggest methods/functions and offers tips. It also contains easy-to-use project management tools, GUI design tools and analysis tools for debugging. Netbeans is also a open source project and enables third party members to develop the IDE or build their own plug-ins for it. If the environment is missing a feature, any developer can create a plug-in and integrate it into Netbeans. [68]

Netbeans is a IDE for professional use, but it is still easy to use, even for beginners. When a user starts a new project, the environment generates the necessary template for user and it is usually some sort of example program, that can be executed and modified. The IDE also supports third party tools and plug-ins, it supports object development and different version controls, such as Git, SVN, Mercurial and CVS.

The drawbacks of Netbeans are more opinion based than real problems. Arguably Netbeans does not support direct code reuse, since it presents only visually the classes in an indexed search, demanding the class name or parts of it for search. The IDE does not contain documentation, forcing the user to search information about class documentation for methods and variables from various sources. It does, however, support self-made libraries.

When mapping Netbeans to the requirements, the results appear to be very similar to Eclipse. Eclipse and Netbeans are two different IDEs that have different working logic but the user interface the programmer sees is very similar. Where Netbeans falls short is learning support, similarly to Eclipse. It does provide a plug-in for BlueJ environment, but this is not mentioned or compulsory, making the use more difficult. Overall there is not much difference between Netbeans and Eclipse, but it is more of a personal opinion which one to use.

IntelliJ

IntelliJ IDEA (https://www.jetbrains.com/idea/) is developed by JetBrains, founded in Check Republic in February 2000. In the beginning of 2001, IntelliJ IDEA 1.0 was released and on the same year the second version, IntelliJ IDEA 2.0 was released. After a few years of fewer releases, the company started to develop tools to revolve around the IDE, providing plugins and support for other programming languages than Java. The plug-in repository has already over 1200 plug-ins for IntelliJ, covering practically all aspects of software engineering. [69], [70]

For this comparison, the IntelliJ IDEA Community version is used. JetBrains offers two possible version, Community and Ultimate, where Community is free and Ultimate requires a licence. As for Ultimate, it covers all the requirements in the list except for availability, which is one of the most important factors in this comparison. The Community version covers the basic tools for programming, making the IDE easy to use, it has third party tool support and version control tools, it supports objects and is available for free. With the offered functionality the IDE could be used for education but because the full version requires a licence, all features of the IDE cannot be used. This causes the IDE to lack said functionalities in this comparison. [71]

MonoDevelop

MonoDevelop (http://monodevelop.com/) originated when few developers in C# community wanted to have a programming environment for Linux. In Windows, some developers used Visual Studio, Microsoft's own IDE for C# while other developers created an open source SharpDevelop project. That project was constructed for Windows and it lacked the support for Linux. The editor and intelligence engine were then extracted from SharpDevelop project and modified to use Gtk# in 2003. The project was planned as a text editor and its goal was to offer a choice for the use of Windows-only format. The text editor, later known as MonoDevelop, grew when the SharpDevelop community started to convert SharpDevelop components to MonoDevelop and the IDE was finally born as a standalone build. [72],[73]

MonoDevelop is an unusual IDE, since it supports languages based on the operating system. For all three operating systems the supported languages are C# and Visual Basic, usually met in Windows environment. For Linux the IDE offers support for languages like Java, C/C++ and Python. [74]

Just like the other IDEs in this comparison, MonoDevelop also covers most of the requirements. It has been created to be easy to use, it supports objects and group tools and is free for use. MonoDevelop does, on the other hand, support Unified Modelling Language (UML) diagram for visualisation, so it could be called more education friendly than the other IDEs. What the IDE lacks is the amount of tools. It has some basic tools integrated into it, but third party plug-ins are not supported and it causes the IDE to be limited.

Qt Creator

Qt Creator was originally an IDE developed by Creative Friday and was then called Workbench. Afterwards it was named Project Greenhouse and before the first release in 2009, it was named Creator because of company philosophy. The IDE is actively being developed as a part of Qt Project, an open source project for software development. [75], [76]

Qt Creator is an IDE specifically for using Qt (http://qt-project.org/) and its GUI library. It offers support for C++, JavaScript and Qt Meta Language (QML) and extends the platform support. The platforms that can run Qt Creator are Windows, Mac OS X and Linux, but the IDE makes it possible to build on the previously mentioned platforms and in addition to Android, BlackBerry 10, iOS and QNX, making it the most versatile IDE for source code builds. Qt Creator contains integrated tools for GUI building and encompassing support for debugging and testing. [77]

The look and feel of Qt Creator suggests that the IDE is meant for professional use. For a beginner the project initialisation and building seem to be more complicated than in more straightforward IDEs. The integrated tools include the basic tools, but not an extended library like the others in this comparison provide. The IDE has been built for objects; the

whole Qt framework consists of objects. Since the IDE is built for the framework, it has built-in functionality for manuals and object lists. The learning support is as weak as it is in the other IDEs mentioned before, it does not contain any visualisation tools for concepts that students could use. In addition the IDE supports version controls, such as Bazaar, CVS, Git, SVN and Mercurial.

Environment comparison

Each of the environments were validated using the requirements by Kölling [64] and the results can be seen in Table 15. The ease of use metrics are determined by the amount of documentation about the IDE, available tutorials and design focus. To achieve the required ease of use, the IDE must be designed as flexibly as possible: for beginner as well as advanced programmers.

Table 15. Mapped environments. (x=covers fully, (x)=arguably covers)

	Netbeans	**Eclipse**	**IntelliJ Community version**	**MonoDevelop**	**Qt Creator**
Ease of use	x	x	x	x	
Integrated tools	x	x	x		
Object-support	x	x	x	x	x
Support for code reuse	(x)	x			x
Learning support				x	
Group support	x	x	x	x	x
Availability	x	x	x	x	x
Σ	**6**	**6**	**5**	**5**	**4**

As mentioned in Section 5.1, the language candidates for this course are Java and Eiffel. Only usable IDE for Eiffel is EiffelBench, that was not popular enough to enter into the final requirement mapping. One requirement for the IDE was that it should be popular, which EiffelBench clearly is not. While Eiffel might offer students better understanding in

some object-oriented concepts, in the course it is also necessary to provide students tools for industrial development. EiffelBench does not meet this criteria and neither does Eiffel, so Java is chosen as the programming language for the course.

To decide an environment for the language, three possible IDEs provide full support to Java: Netbeans, Eclipse and IntelliJ. The two most popular IDEs, Netbeans and Eclipse are the final environment candidates for the course, since IntelliJ Community version cannot compete with them. From Table 15 its evident, that the two IDEs are similar, almost identical when mapped against the requirements. The final decision basically boils down to individual preference, but Netbeans has the support of Oracle, which is the official source of Java. Java and Netbeans are usually published together by Oracle and it is possible to install both of them simultaneously. This is a tremendous advantage to a beginning programmer, who wants to install the IDE on their own machine without any compatibility problems. Because of the easy installation, the default environment for the course is Netbeans.

5.3 Teaching materials

The previous course consisted of lectures and lecture videos recorded from the lectures similarly to the study of Ronchetti [78]. The lectures however became unpopular, since the students rather watched the recordings than participated in the lectures. This is why the new version adopts the reverse classroom by removing the physical lectures and replacing them completely with video lectures. Students are expected to watch the lectures before the exercises. The lectures are published openly through YouTube, since the publication channel has been found adequate in [23]. By using YouTube, the lectures are open for anyone to watch anywhere and the feedback can be received from outside the course as well. Following the guidelines presented in [37] and mentioned in Section 2, the video lectures are determined to be about 20-30 minutes, even though adequate coverage requires longer videos about some topics, such as UML.

The video lectures are done with a computer software that records the lecturer's computer screen and the audio is added through the microphone (e.g. [23]). This way students can see the actual use of the IDE step by step and can repeat the problematic parts if necessary.

It also covers one of the guidelines for reverse classroom, where it is stated that multimedia should be added to the lectures. This is done by recording the lecturer's screen in the video, where code, lecture notes, and lecturer's audio commentary are used.

Using only lectures without any literature on the course can give students a limited view of the topics at hand. Through the video lectures, students get concrete examples of how to use the IDE and how to program but the theory of object-orientation is lacking. They do not learn how to think with objects. This is why the course requires literary materials, to support the lectures and give students a wider viewpoint of the matter. The idea is to use similar worksheets, that were found rather successful in [79], where students were handed worksheets that contained all the instructions for the task. In this course the exercise tasks are done parallel with the worksheets and the tasks are linked to the theory by using similar topics and models for solutions in examples and in exercises. Then the exercise tasks are removed from the worksheet so they contain only the theory as text and examples to introduce the design and concepts to students and the exercises are handed out separately. The theory was implemented as two manuals and those were published to students as course study materials, while exercises are separate but a linked entity. The manual topics are presented in detail in Tables 16 and 17.

The theory is presented in two manuals, since they have separate topics. The first manual concentrates on object-orientation as objectively as possible with programming examples to support the theory. The manual contains nine sections about course topics and have corresponding exercise sheets to apply the theory. The second manual contains other concepts, that are not object-oriented but still important for a programmer. The concepts presented in it are crucial to understand to complete the course. The manual goes through version control, open data and its uses, XML and GUI building. These components are used in a later part of the course in exercises and in the course project.

Table 16. First manual – Object-oriented Java (94 pages) [80].

Section	Topic	Content
1	Classes and object-oriented programming	Defining object-orientation, classes and objects. Basic constructors, instance variables and visibility modifiers public and private. Java's data types, Array-class, basic printing and few words about main().
2	More about classes and objects	Dismantling objects, basics about references and typecasting. Typecasting in Java, ArrayList, Vector and user I/O.
3	Libraries and data streams	I/O streams and serialisation. I/O streams in Java and data structures Enumeration and Map.
4	Object-based design and UML	Coupling and cohesion, basics about UML and class diagram. Java's use of keyword this, basics about packages/namespaces, Enum.
5	Inheritance and abstraction	Abstract class with visibility modifier protected, overloading.
6	Exceptions and inheritance	Interfaces and error management. How Java implements interfaces versus abstraction, foreach and keyword final, error management in Java.
7	Copying and assignment, class variables	Reference copy, shallow copy, deep copy. Copying in practice with Java, insertions and class variables with keyword static.
8	Inner classes	Few examples and basics about inner classes and their communication.
9	Java with Internet	Using Java to read data from Internet and data manipulation/reuse, especially XML.

Table 17. Second manual – Practical tools (72 pages) [81].

Section	Topic	Content
1	Version control	Overview about different types of version controls.
2	Open data	What is open data and how it is defined and used. Some instruction about where to find open data.
3	XML	Basics about XML-structure and learning to read the syntax.
4	Maps and programming	Map types and coordinates. Problems in implementing maps into electrical format and examples using Oskari, an open source map framework (http://oskari.org/).
5	GUI	How to use graphical tools to create a GUI, adding components and functionalities dynamically, accessing Internet through user interface.

5.4 Syllabus

The course syllabus was constructed from the new teaching material and is presented in Table 18. The new syllabus takes considerable amount of structure and material from the old syllabus. The course was built by reusing as much material as possible to minimise the amount of work and to retain consistency.

Table 18. New course syllabus.

	Topic	**Content**	**Lecture videos [number / length (sum)]**
L1	Introduction to objects	Introduction to OOP and course's tools.	4 / 1:00:57
L2	Classes and objects	Java and objects, user I/O and methods in Java.	3 / 1:20:29
L3	Data structures in Java	Data structures in Java, more about OOP, variables and constants, operations.	4 / 1:15:50
L4	File I/O in Java	File I/O, serialisation and libraries.	3 / 47:07
L5	Object-based design	Object-based design with UML and unit testing.	3 / 1:47:12
L6	Inheritance	Inheritance, abstraction (abstract classes/interfaces) and polymorphism.	3 / 1:21:00
L7	Graphical user interface	Basics for building a GUI.	2 / 1:03:47
L8	Class variables and methods	Basics about class variables and methods.	1 / 27:48
L9	Exceptions and error handling	Object-based design philosophy, exceptions and error handling, Java + Internet and XML.	3 / 1:22:42
L10	Generics and iterators	Generics in Java, iterators.	2 / 58:44
L11	Inner classes, anonymous classes, copying	Basics about inner and anonymous classes, theory behind copy and assignment.	2 / 43:02
L12	Recap	Independently going through the material in the course.	No video

5.5 Exercises and project

While the physical lectures were removed from the course, it was still necessary to hold a programming session for students, where they ask for guidance. In [82] the assisted programming sessions were found to support the basic understanding about programming and students should interact with the teacher more than programming individually without instructor. While the theory and instructions are given through different routes, the classroom session is preferable to students as was evident in the study, where students gave free-text answers in a survey [82].

Some of the problems with the previous course was that the students did not participate exercises and they did not have adequate skills with the course tools. By giving extra points to students who come to the exercises to present their work it was possible to see an increase in weekly participation. And while students applied the theory they have studied before the exercises, they are still guided with problems or difficult concepts, whether they had problems with the concepts or the course tools. In the exercises students also receive points for any completed task, while in the old course students received points only through the VLE. In the new course graphical components and GUI programming were used that the VLE cannot check and grade, leaving the grading to the teacher. This supports the participation to the exercises.

The exercises were divided in two major parts by the two periods. The first period is mostly about basic object-oriented programming and text-based I/O as the interface. During the first period, the exercises are returned into the VLE. For the second period, it was found reasonable to use graphical user interfaces as the user interface, where the students can use the old programs they did in the first period and transform the text-based interface into a GUI. By bringing the GUI building as an essential part of the course, it was aimed that students find the exercises more meaningful and do not lose motivation when comparing their programs to applications they themselves use. The loss of motivation was hinted in [53] as a major hurdle for student motivation caused by text-based interfaces.

In the exercises students were encouraged to program in pairs, using the technique of pair programming as described in [83]. This allowed them to read code while the other one of

the students is programming and also encouraged them to use the group tools. The use of version control was also mandatory as the project can only be returned by sharing the version control repository to the teacher.

The first week of exercises is concentrating on the tools for the course, containing the basics of Netbeans, creation of Java program, using the VLE and Git and creating jar-packages.

The exercises in the weekly workload consist of five tasks, where the first task is a basic task. The first task has to be extended in the following tasks and student gets one point if 3 tasks are completed. Two more tasks are in the exercises to give more challenge to more experienced students and are awarded with another point when completed. The tasks are linked to each other, where the first task is a simple basic task and the following tasks extend it with more functionality and different limitations.

The course project is built entirely on the second manual containing more concrete and useful concepts. In the project students were required to build a graphical application, that can download and parse XML-files and extract location data from them. The location data is then visualised using graphical interface and map view. The course project is to create a software that simulates a postal office, where the user can send different packages from one SmartPost to any other SmartPost. The SmartPost is a postal network in Finland and Estonia, where users can receive and send packages through a package kiosk. The SmartPost location data is open data in XML-format, that students are required to download and visualise on a map. The construct of the project is required to be object-oriented and the interface is suggested to be done last. The students are required to develop a working back-end logic before actually building the GUI, leaving them enough time to finish the software and then use the rest of the remaining time to add visual components and eye-candy to the GUI.

5.6 Exam

In previous years students had given negative feedback about the exam. Some students criticised the existence of the exam while other were not satisfied with the content. In the new course students were offered a choice to attend to a traditional paper exam or do the exam online. To provide equal footing to students, the exam did not contain programming tasks, that would have given unfair advantage to students, who did the exam online.

The physical exam and the online exam were structured similarly and the questions were generated from the same question pool. The physical exam paper was generated by the lecturer for everyone in the exam event, but the online exam was automatically generated for each participant individually, random questions in definite order.

5.7 Overall changes

Before the changes and comparison are presented, the guidelines of object-oriented presented in Section 2 are compared to the new version in Table 19. The number of simultaneous changes is significant, so the final comparison is presented in the end of this section in Table 20. The guidelines were compared to other studies [84]–[86] and these guidelines by Kölling and Rosenberg were found the most suitable, because they are designed specifically for an introductory level object-oriented programming course, similar to the course in LUT.

The exercises and the administrative arrangements were handled by the lecturer, just as they were done in the previous course. The exercises and videos were posted online weekly, each week covering necessary topics, that demanded the knowledge from previous weeks. The exercises were held once a week, each session took up to two hours.

Table 19. New course in comparison to the guidelines [20].

Guideline	**Uses in the new course**
Objects first	In the course objects are brought in right from the beginning by presenting object-orientation and comparing it to commonly used paradigms.
Don't start with blank screen	The examples in the manuals start off as UML-examples of programs, that are gradually explained in code. Students also use one of the examples as the base for programming tasks and start extending it.
Read code	Students have to read code when they read the manuals, since the examples are first shown as code and later explained in detail. To understand the program, students have to see the code first and then read the description.
Use "large" projects	The first example that students see contains a combination of 4 classes. The example demonstrates a car rental, where users can add reservations. After that, students have to build programs, that contain only one class (plus the executable class), but after the second week, students have to start using multiple classes.
Don't start with "main"	The main()-function is generated by the IDE for students and it is emphasised that students do not have to understand how it works. They are only told that they can run a program with it and that the concepts are taught later.
Don't use "Hello World"	"Hello World" is used as the first project, where they only test, that they can log into the VLE-system and can run their project in it. Printing one line in one class is the easiest method of testing if they can use the system and it is not required to understand the code.
Show program structure	The program structure is made visible by showing the UML solution of it. In the fifth week, students have to construct their own UML class diagrams to complete the weekly exercise. They have to draw their own UML for the project as well.
Be careful with the user interface	The graphical interface is presented and taken into use only in the second period, when students should have adequate understanding of objects. It is also required for half of the exercises and course project, but the main emphasise is on the program, not the GUI. Students can receive extra points from the GUI only after the application is satisfactory. The restriction is aimed to minimise the GUI polishing in the expense of application logic.

Practically all the components of the course changed and the changes are presented in Table 20. The lectures were changed from physical and video lectures to only video lectures because of the attendance. The participation to weekly exercises wanted to be raised and as the result, the solutions gave points to students and each student had to achieve enough points to pass the course. The exercises were made partially mandatory to every student. The programming language was changed to Java and Netbeans remained as the IDE. The amount of literary was increased by producing two custom made manuals for the course in addition to the lecture slides. With the example programs the use of version control was increased by producing the necessary programs through the lecturer's repository. The exam was also changed to provide a chance for students to complete the exam online instead of on paper, while the exam on paper remained an option.

Table 20. Old version and new version.

	Old course	**New course**
Lectures	Physical and video	Only video
Weekly exercise tasks	Voluntary, no points gained	Compulsory to achieve 10 of 22 points
Programming language	C++	Java
IDE	Netbeans	Netbeans
Supplementary material	Lecture slides	Two manuals and lecture slides
Example programs	From university VLE as a zip-file	From a git repository in Bitbucket
Exam	On paper	Online or on paper

6 RESULTS

In this section, the results from the feedback survey are presented in comparison to previous years. In the feedback students were also able to give a free-text feedback and some of the students quotes are mentioned. The teacher made observations about the used methods that do not show in the survey and are presented separately. This section focuses on the quality of the solution in DSRM.

6.1 Test phases during the summer

The course was run in two phases during the summer to see if the course is developing in the correct direction and do students require more content. The first phase, alpha test, was performed by a student, who had very little experience in programming. The test was successful and the student was able to cover the whole course in one month. The second phase, beta test, was not as successful. The test used a larger group of students, mixed with students who had previously completed the course and students, who had not. The students who had already passed the course lacked the motivation to redo it – although they had enrolled in – and students who had not taken the course had various issues, ranging from motivational issues to holiday trips, failing to complete the course. In the two phases the course did not yet have the video materials or in-class exercises and students were expected to manage with the worksheets. The issues students had, did not come from the material, so it was impossible to evaluate the material through the beta test and the pilot failed.

6.2 Actual course in autumn

In Table 21 it can be seen, that the course got higher average grade from students than ever before. This result supports the objective to improve the course, when the feedback about the tools and methods in the course got an average of 4.68. In Table 21 can be seen, that the average grade from students rose in both categories. It appears, that students responded well to the tools and methods used in the new course. The averages have improved, while they are not statistically significant partly due the small sample size in the course. The data of Table 21 is visualised in Figure 2.

Table 21. Course average grade from students. (1=lowest, 5=highest)

	2010 (N=19)	**2011 (N=18)**	**2012 (N=19)**	**2013 (N=15)**	**2014 (N=19)**
Average grade for course	4.00	3.94	4.37	4.36	4.47
Average grade for tools and methods used in course	4.00	4.06	4.42	4.43	4.68

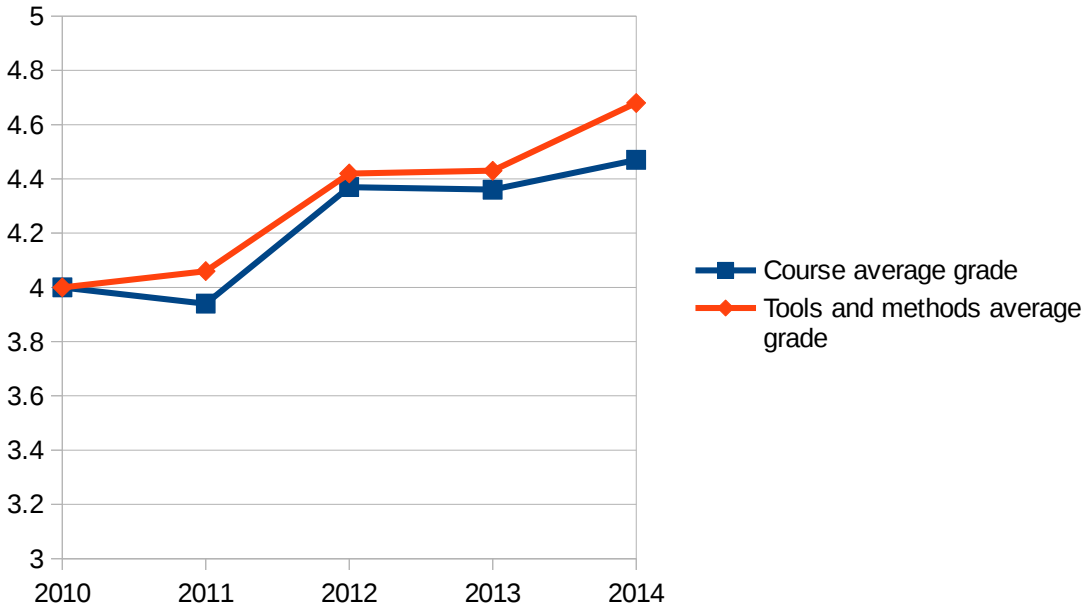

Figure 2. Course averages from student feedback.

From the student feedback, nine students used 100%-125% of time on the exercises and course project than was allocated and one student used 125%-150%. Rest of the students who gave feedback reported their used time to be in 75%-100%. This is presented in Figure 3. The course was allocated for 5 etcs, which means about 130 hours of work. Other components: lectures, literature and the lecture videos were used less than designed; the literature was used the least. Still students regarded the number to credits to be appropriate to the overall amount of work. When students were asked what they would change about the course if the credits remained the same the answer was clear: more programming and less theory from literature. The lectures from YouTube were suitable for the course, since students would not change the lectures but the amount of literature should be decreased.

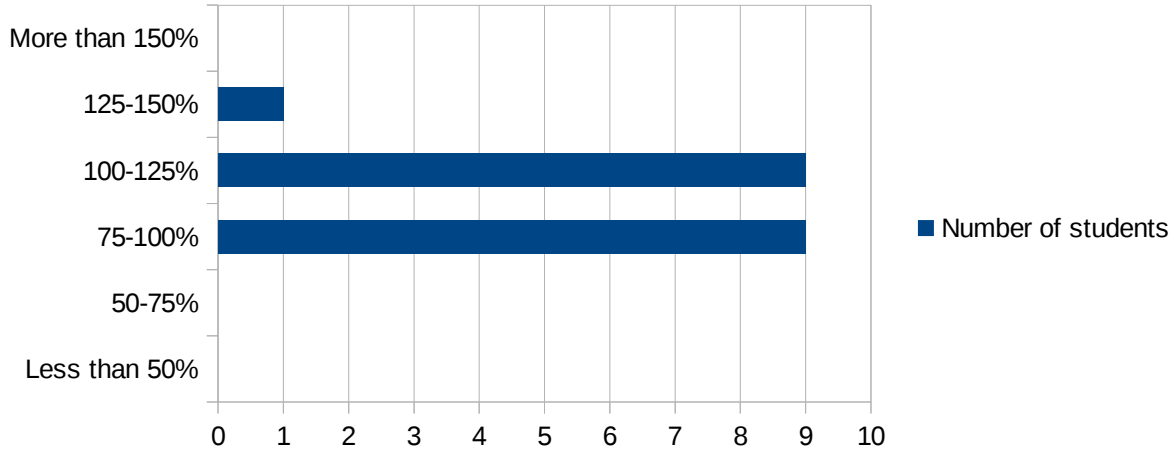

Figure 3. Student feedback about the use of time versus the allocated time of the course in 2014.

The observed challenge of course's components were again asked from the students and the results with comparison to previous years is presented in Table 22. The new course contained multiple changes and some components were not used before or after the change and there were changes in some of the assessed components.

The easiness was regarded similarly to the previous years and comparing the numbers to previous years, there was no component that was observed to be significantly harder than before. The course literature was harder and the newly made manuals were as challenging as the Internet manuals. The project and the exercises had a slight increase in the easiness, which indicates that they were easier than expected.

Table 22. The easiness of course's components. (1=hard, 5=easy)

	2010 (N=18)	**2011 (N=18)**	**2012 (N=19)**	**2013 (N=14)**	**2014 (N=19)**
Lectures / Introduction lecture (2014)	4.12	3.61	3.63	4.00	4.58
Lecture slides	3.82	3.77	3.68	4.14	3.94
Lecture recordings	3.63	3.56	3.83	4.15	4.18
Example programs	3.59	3.56	3.32	3.86	3.71
Exercise events	3.29	3.00	3.16	2.23	3.24
VLE-tasks	3.77	3.17	3.16	3.93	*
Programming project	3.29	2.72	2.58	2.64	2.82
Object-oriented Java manual	-	-	-	-	3.53
Practical tools manual	-	-	-	-	3.50
Course literature	3.18	3.00	2.88	3.23	2.78
Manuals in the Internet	3.77	3.13	3.37	3.31	3.50

* There were no separate VLE-tasks, while it was possible to return half of the exercise tasks into the VLE

From Table 22, the components that were changed the most were selected for Figure 4. Some components, that were not used in the course this year or were used for the first time were left out for the sake of clarity. The figure visualises an interesting development in the lecturing material; lecture slides and example programs had a similar drop from previous year and have developed very similarly in previous years.

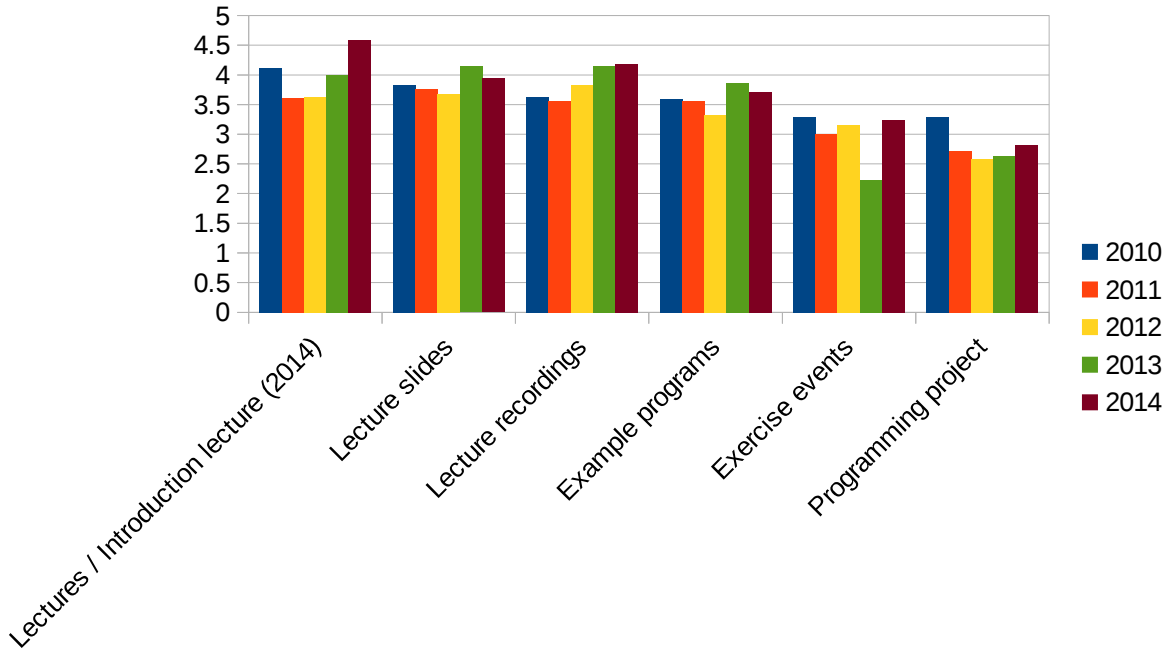

Figure 4. The changes in easiness of the major components of the course.

The usefulness of course's components are presented in Table 23 and similarly to previous figure, the major components in Figure 5. There is a clear decline in the usefulness of the lectures and the lecture slides, while the lecture recordings were ranked higher than ever. The most useful components of the course were the exercises, lecture recordings and the project.

The theory from literature in the course was not graded high, only the Internet manuals reached an average grade over four, while the manuals by the faculty were not any better than course literature. From Table 22 the manuals were regarded easier than the course literature but the usefulness in Table 23 is on the same level. The course literature also reached the lowest grade in the course. The manuals in the Internet, such as tutorials and blogs about programming, have been achieving a higher grade in usefulness every year, since the material is expanding in the Internet, for example stackoverflow (http://stackoverflow.com/).

Table 23. The usefulness of course's components. (1=useless, 5=useful)

	2010 (N=18)	**2011 (N=18)**	**2012 (N=19)**	**2013 (N=15)**	**2014 (N=19)**
Lectures / Introduction lecture (2014)	3.94	3.65	4.11	3.87	3.37
Lecture slides	4.06	3.53	3.68	3.80	3.47
Lecture recordings	3.24	3.82	4.17	4.36	4.58
Example programs	3.88	4.35	4.21	4.20	4.32
Exercise events	3.41	3.50	3.79	3.36	4.68
VLE-tasks	3.89	4.24	3.68	3.67	*
Programming project	4.06	4.29	4.47	4.67	4.47
Object-oriented Java manual	-	-	-	-	2.95
Practical tools manual	-	-	-	-	2.95
Course literature	3.35	3.54	3.00	3.57	2.88
Manuals in the Internet	3.88	4.06	4.11	4.20	4.44

* There were no separate VLE-tasks, while it was possible to return half of the exercise tasks into the VLE

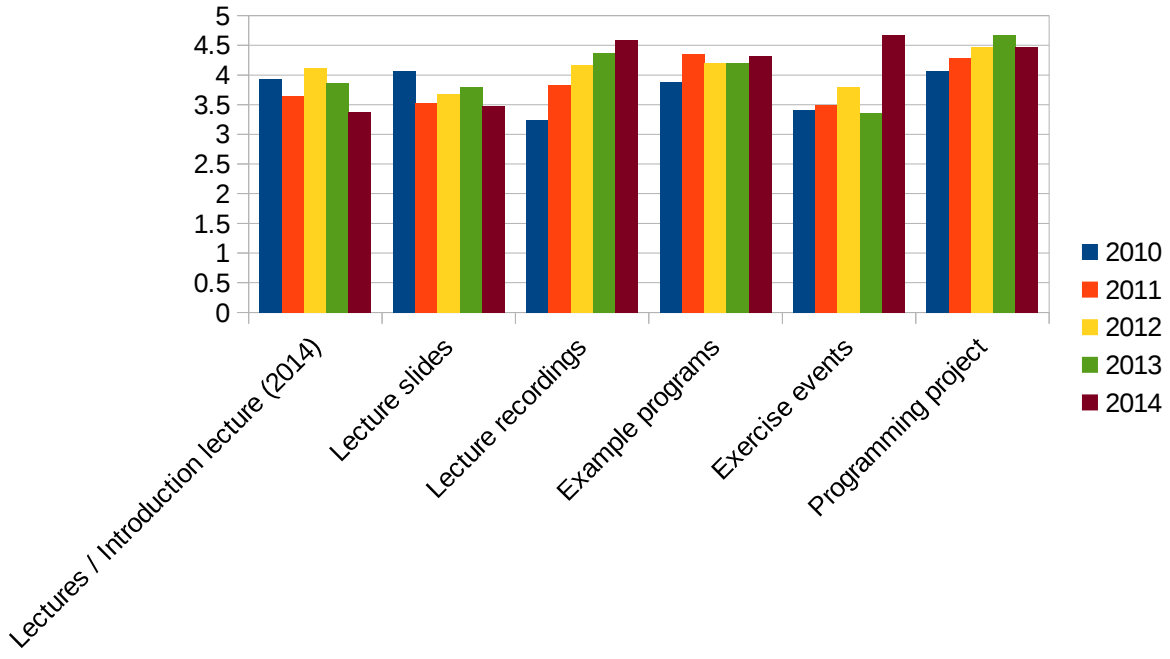

Figure 5. The changes in usefulness of the major components of the course.

Since the lecture videos were provided through YouTube, it has become possible to provide statistics about how much students watch the provided videos. The statistics are presented in Table 24. Since the physical lectures were removed, the number of watched videos has increased significantly. As reference, the standard lecture takes maximum of 90 minutes and the course was 14 weeks long in 2013. The maximum time used to lecture was 1260 minutes (in addition to lecture videos in Table 24), while the number of participating students was varying from 2 to 20.

Table 24. Use of lecture videos.

Year	Views	Watched minutes in the course	Most popular topics
2013	575	6885	Introduction to the course, Basics about objects, Inheritance
2014	2361	18983	First introductions to course, UML and GUI

The students expectedly used the video lectures more in 2014 than in 2013. In Figure 6 the daily views of the videos is presented and it shows, how much students used the videos. The graph shows clear spikes weekly, on the same day when the lectures were uploaded and published. The exercises were held on Monday and the spikes indicate, that the videos were used the most before exercises but also in the middle of them. The drop in October is the exam week, when there were no exercises. The last exercises were held in 1st of December, which indicates that students used the videos for the course project and recap material for the exam.

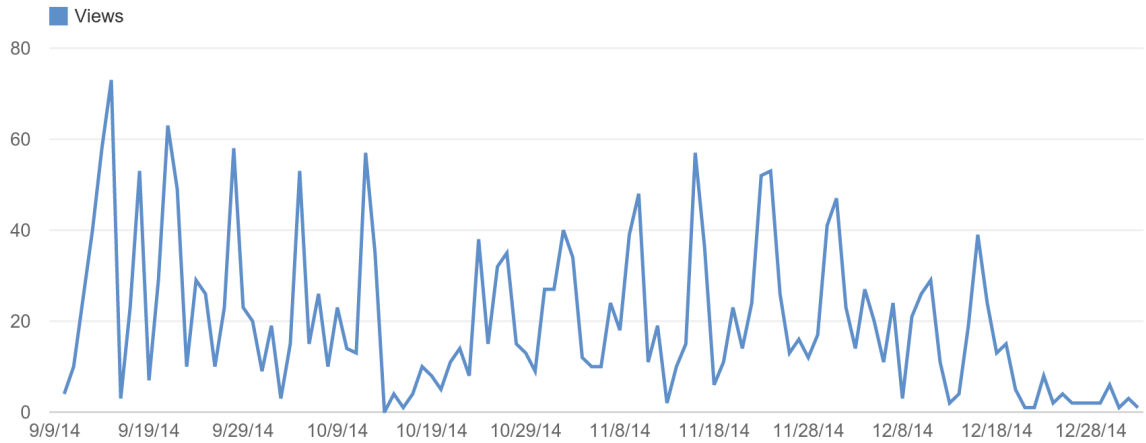

Figure 6. Daily views of the videos.

Figure 6 only presents all the views of the videos but it does not indicate which videos were watched. In Figure 7 the five most viewed videos and their views are presented. The five most popular video titles and their colours are:

1. Java + JavaFX = GUI (purple)
2. JavaFX and WebView (dark red)
3. Java + JavaFX = GUI #2 (green)
4. Object-oriented programming in Java – Intro (lighter red)
5. Class diagrams with Umbrello (yellow)

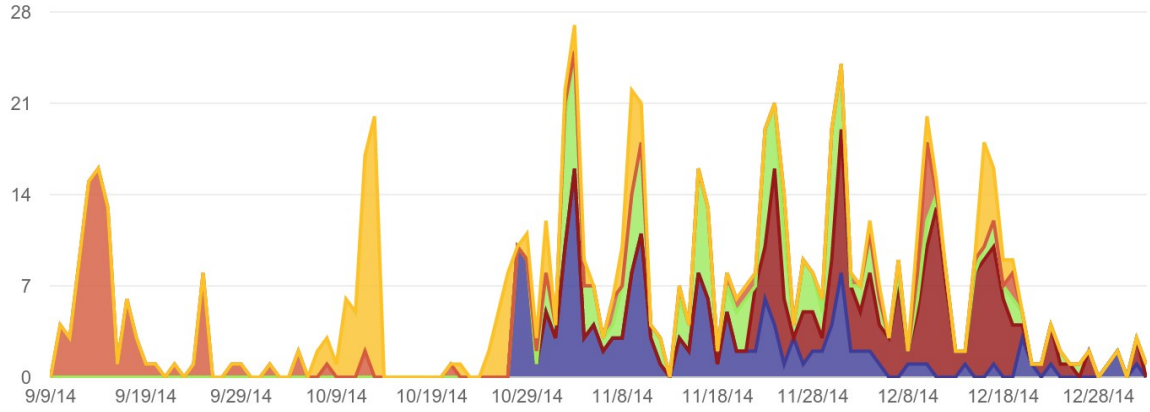

Figure 7. Five the most viewed lecture videos. JavaFX GUI (purple), JavaFX WebView (dark red), JavaFX GUI #2 (green), Intro to Java (lighter red), Class diagrams (yellow).

The figure shows, that the video about the basics of object-orientation were watched in the beginning but also later in the course, especially in December. This holds true for the video about the UML as well. Students used the videos to refresh their memories about the basics, when they had to use them again. By repeating the subjects, students can form a clearer image about the concepts of the course.

The students were asked for open feedback about positive and negative aspects of the course and ideas for improvements. The positive feedback concentrates mostly on the exercises and project: "*Project and exercises were successful, one could actually do something else than calculators (i.a. exercises with graphical user interface).*" and the video lectures: "*The video lectures were a great invention, especially in teaching programming. They made the schedule adjusting possible*".

The negative feedback focuses on the content of the course, such as the exercises in the first period not covering enough material: "*Why can't there be any useful programs from the beginning? The UML diagrams were taught too fast, I don't think I understood them well enough*". The course also concentrated more on the time used than the overall know-how: "*Currently the course grading measures the amount of time used.*", which can be considered both positive and negative since repeating increases the learning result [87].

The student ideas addressed multiple subjects. One feedback asked, if the course could use more open data as teaching material: "*The open data services should be utilised more (xml and especially other formats). In overall this course is very good as a whole.*". Students seemed to adopt the use of open data without any major problems.

During the course the lecturer made observations about the exercises and overall atmosphere of the course. One of the course's problems was that the exercises mismatched the theory in some parts and some students were confused by it. The observations suggest that the manuals have to be updated to match the course structure better and the manuals should be integrated to the course page, tying the literature closer to the exercises and lecture videos. The material lacked mostly the UML that was actually used in one week only and it should be added to both lectures and exercises. A positive surprise to the lecturer was the fast pace with the course projects; the evaluation process was much faster than in previous years. There were more projects, that were let through without the need of fixing, presented in Table 25 and visualised in Figure 8. One of the reasons was the lack of pointers and other more complicated features.

Table 25. Returned and redone projects.

Year	Projects returned	Redone projects	Redone percentage [%]
2010	30	14	46.67
2011	33	28	84.85
2012	24	17	70.83
2013	22	14	63.64
2014	18	8	44.44

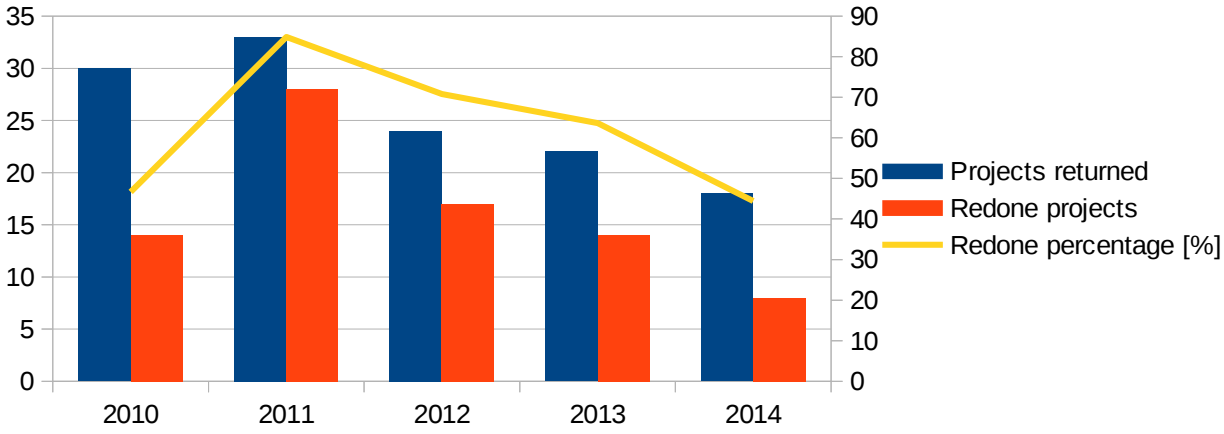

Figure 8. Projects and redoes.

One suggested improvement was to use mandatory quizzes, that have to be completed before the exercises. This was also supported by Zappe et al [37] and mentioned in Section 2. During the course students had not always used the available material and were not familiar with the theory before the exercises. Adding mandatory quizzes before exercises aims to increase the theory review by students. Another improvement would be that the exercises should contain some alternatives, offering multiple, parallel tasks, enabling students to pick the tasks that are the most interesting to them.

One of the improvements was that the teacher did not have to spend as much time teaching and administrating the course as previous years. In Table 26, the time that the course took is presented in comparison to previous year and the hours are the concrete, actual hours, not the budgeted hours. In Table 26, only the teaching hours are calculated, the set up time – when the lecture videos and materials were created – is not included. The total number of hours dropped by 16 hours and the focus of teaching evidently moved towards interaction in exercises. The project evaluation also took less time, while the number of returned projects were nearly the same as can be seen from Table 25. This can be explained by the lack of redoes and their evaluations in the new course.

Table 26. Overall time in hours used by the teacher.

Year	Exercises	Evaluating projects	Lecturing	Whole course
2013	12.5	23.5	17.2	88.5
2014	29.5	18	0.5	72.5

One of the results that were outside the hypothesis was the percentage of students, that completed both the project and exam and received a grade. The numbers can be seen in Table 27. In previous years some students have returned the project and failed to redo it but still participated and completed the exam. The new course was the first version in this time frame, where all students who did the project and completed the exam received a grade.

Table 27. Students enrolled and completed the course.

	2010	2011	2012	2013	2014
Enrolled to the course	69	68	56	54	54
Started the course (achieved at least 1 point)	46	51	42	38	42
Completed the project	29	41	31	28	26
Completed the exam	28	41	31	28	26
Got a grade	28	39	30	26	26

Figure 9 is drawn with the data from Table 27. The figure shows clearer indications that there were no students, who left the course unfinished by not participating into the exam or not finishing the project.

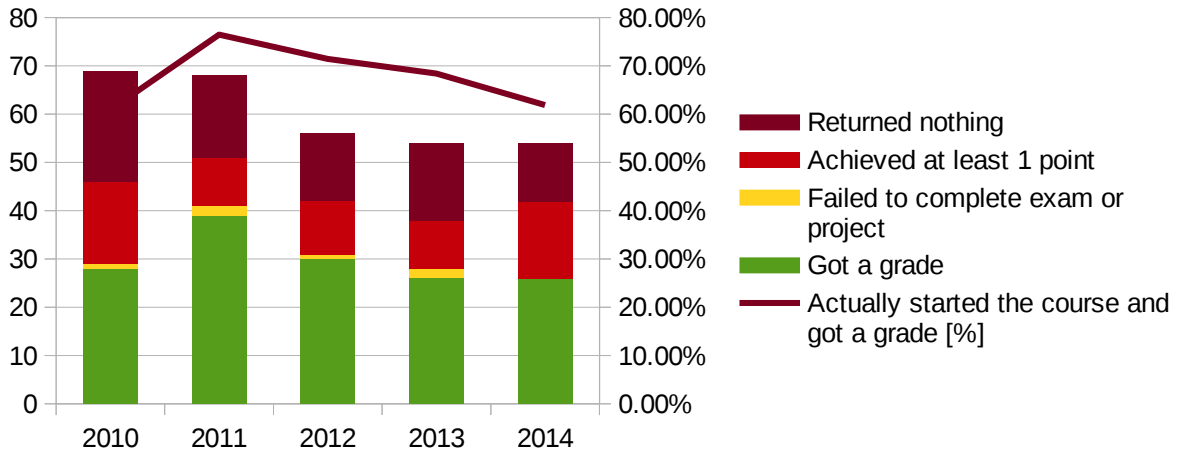

Figure 9. Percentage and number of students completing various course elements.

One aspect in this study that failed was the grading. During the course it was noticed, that students get too many points from exercises and participation when compared to the maximum points of the course. The grading scale was made too easy and students received better grades than earlier years. The number of grades can be seen in Table 28.

Table 28. Number of student grades.

Grade	**2010**	**2011**	**2012**	**2013**	**2014**
1	2	5	6	3	0
2	11	19	9	14	1
3	11	7	5	4	3
4	3	6	8	4	10
5	1	2	2	1	12
Average	2.6	2.5	2.7	2.5	4.3

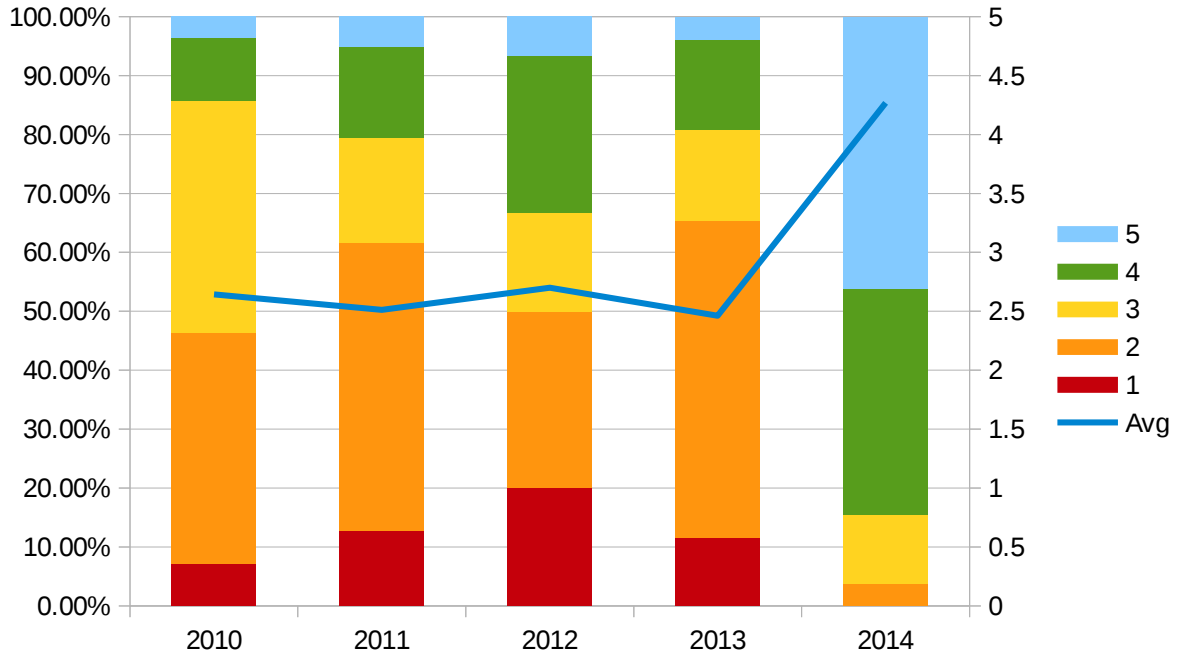

Figure 10. Grades and average.

Overall the new course can be considered a success that can be seen from the overall course grade and the separate feedback grades for the components of the course. The open feedback from students was also mostly positive even though some aspects of the course should be remodelled. The definite improvements on learning results cannot be assessed

because of the change in grading, but the number of redone projects show some evidence about better learning.

6.3 Summarising the results

There were multiple aspects of the course and the results were measured. Most of the results met the criteria of the research questions, while there were other, unexpected results as well.

- The course and tool average grade from feedback was better than before
- Students used the allocated time for the course
- There were no radical changes in the perceived easiness or usefulness of the course
- The adoption of reverse classroom was successful based on the student feedback
    - Students felt that the method gave them more freedom to learn
- The number of redoes in course project decreased
- The lecturer used less time in overall for the course
- All the students who completed the project and exam got a grade
- The grading was not successful because, for example, students got too many points from exercise participation

The study did not fail in any other area but the grading. There were challenges to provide a relevant grading, because of the multiple changes and the predictability of student reactions was difficult.

7 DISCUSSION

In this thesis the major research questions were "*What is the current status of object-oriented programming and open data in teaching*", "*How shoud a new course on OOP be constructed?*" and "*How does the material and course compare to its previous implementation?*".

Currently object-oriented programming is taught in universities which offer degree in computer science. Object-orientation has propelled itself into the major skills that any software expert should have at least adequate knowledge about [12]. Most of the discussion focuses on whether objects should be taught early or later and at the same time the objects-first method of teaching is steadily gaining a larger foothold. Most of the open data in teaching deals with open educational resources and it is not used in teaching widely, but when it is used, it enhances teaching and brings real life data into the classroom.

The new course was constructed around the reverse classroom method, where the traditional physical lectures are removed from the course and they are replaced by video lectures, that students are recommended to watch before exercises. The only in-class activities are the exercises, where students apply the theory they have learned from the lectures and worksheets. Students are encouraged to participate in the educational events by rewarding the participation and required to complete a necessary number of assignments. The new course removed C++ as the programming language and substituted it with Java. The objective of the new course was to emphasise the object-oriented aspects of programming and highlight design over implementation. The participants were expected to follow the best practises of object-orientation.

The new course's feedback was mostly positive and it received higher overall grade from the students than the previous course. The exercises and project were regarded easier than in previous years, which may be a direct consequence from the higher attendance in the exercises or better know-how. The exercises and project might have felt easier, because the students had more time to spend with the materials and the teaching focused more on the interaction than lecturing. The grade for the material did not change from previous course

even though the new course had new literature material in Finnish. The material was not as structured as it could have been for the course and students probably did not find the material as relevant as it should have been. As can be seen from Tables 16, 17 and 18, the sections in the manual and lecture did not always match, causing confusion about what the weekly exercises might contain. After the course is revised with the feedback, the order of sections and lectures should match better.

7.1 Improvement ideas

In the future the course could be improved by adding mandatory quizzes before exercises [88] and perhaps replace the exam from the course, if the quizzes contain the same level of evaluation as the exam does. Based on the feedback, the exam can be argued to be unnecessary in programming courses, where the main focus is to practice programming. Programming can be learned by doing and the most effective manner to evaluate the student's know-how is to evaluate the program and the process, not the amount of syntax student can memorise. If the quizzes provide satisfactory evaluation of the necessary know-how about object-orientation, the exam could be eliminated. The weekly quizzes would be made by the course teacher by focusing on the weekly topics.

The quizzes would possibly serve the course better than the exam based on the student feedback. The quizzes are not as taxing to students as exam is, since he quizzes reduce the amount of theory to memorise at once. By using quizzes the students can test, if they understood the theory when it is still fresh in their memory, avoiding the misunderstandings as early as possible. The course should be ran again as a self-study course during summer to re-evaluate the effect of the changes in the test phase.

7.2 Limitations

The limitations in this study include the lack of control group for direct comparison and multiple major, concurrent changes, that make it impossible to segregate the cause for results. The changes caused different responses in students and it cannot be differentiated, which change caused which reaction. To separate the causes for results a comparison would be necessary and to enable a comparison, a control group would be mandatory to prevent the annual changes in the participating students.

Another limitation of this study is the lecturer and students. Every lecturer and student are different individuals and everyone perceives the materials and methods differently. It is difficult to definitely state, that this change would have been successful a year before or if it will be successful a year after. The students are also affected by the personality of the lecturer, which can make the course easier or harder to complete.

7.3 Generalisability

Generalisability states that is it possible to use the used methods outside the context of this study [89]. The reverse classroom can be used to teach programming, based on this research and other studies, that indicate the successful use of reverse classroom. The use does not limit itself to the number of students, as long as each student is given adequate opportunities to participate to the in-class activities. This allows the use of reverse classroom in large programming courses, as long as there are enough exercise sessions, so every student can participate. In smaller courses the reverse classroom can be used as is indicated in this study.

The previous research about the use of reverse classroom suggests, that the use of the method increases the students learning results and thus the grades. Students also regard the method more effective and sufficient to each individual than traditional lecturing. While the previous research supports the results in this study, the amount of research about this topic is relatively minor. Based on the research by Lee and Baskerville [90], this offers only limited support to the generalisation and additional evidence is required.

This study does not state that Java or Netbeans is the optimal result for every object-oriented course. The previous programming courses in LUT support the use of these particular tools in this context. However, the results should not vary too much, if the used language and environment were changed as long as the staff use enough time and care to develop the course using this study as the guideline.

The use of open data in any university level courses has been minor and non-existing in programming courses. This study indicates, that students do not find open data as hindrance and use it as they would use any other data set. The use of open data gave

students the feeling of creating actual programs, since the data was from actual, real-life sources that only required visualising. By visualising the data by using graphical programs, students were given more motivation to complete the tasks, since there were more to do than just playing around in a text-based program in the IDE console. Students use graphical programs in their everyday life and by allowing them to create similar programs using real-life data gives students more motivation.

7.4 Future work

Since the reverse classroom in computer science and especially in programming is a relatively new method, its influence students should be researched. In LUT the CS1 course could also be inverted and the influence of the method to students could be evaluated in this course. This would provide insight, whether the reverse classroom causes problems in practices or methods the students use to learn programming.

8 CONCLUSIONS

This thesis was done in Lappeenranta University of Technology as a study to improve an existing object-oriented programming course with new methods. In the course the reverse classroom teaching method was used to see how well the method works in a programming course. Other changes were: the use of open data is presented to students as a new possibility to collect data for software and the course was reformed to use Java instead of C++.

This thesis addresses the current status of teaching introduction level programming in Finnish universities as well as in international universities. The introduction level programming in Finland follows basically the same pattern in every university, where the programming is taught with Java or Python, with small variations in order and content of the course. The international universities were found to have more variations in basic programming but still teaching object-orientation. No university was found that did not teach object-oriented programming in some part of its curriculum.

The previous object-oriented programming course in LUT had been used for multiple years and the student feedback motivated a larger change. The course had multiple problems from the course focus to students participation and the goal of this study was to remove all perceived disadvantages. The change contains the adoption of reverse classroom, the removal of physical lectures and replacing them with video lectures to increase the student-instructor interaction. Mapping the existing practices for programming courses and using literature to compare the different possibilities, a new course was decided to be taught with Java in Netbeans environment.

The results for the new course were promising. The student feedback graded the course average higher than in earlier years and the used methods received positive feedback. Some of the components, such as the literature, did not receive the expected feedback but the material will be improved from the first trial based on the feedback. Overall this study can be seen as highly successful.

REFERENCES

[1] A. Dingle and C. Zander, “Assessing the ripple effect of CS1 language choice,” *J Comput Sci Coll*, vol. 16, no. 2, pp. 85–93, 2000.

[2] M. de Raadt, R. Watson, and M. Toleman, “Introductory programming: what’s happening today and will there be any students to teach tomorrow?,” in *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, Dunedin, New Zealand, 2004, pp. 277–282.

[3] M. de Raadt and M. Toleman, “Language Trends in Introductory Programming Courses,” *Proc Informing Sci. IT Educ. Conf.*, 2002.

[4] C. Stephenson and T. West, “Language Choice and Key Concepts in Introductory Computer Science Courses,” *J. Res. Comput. Educ.*, vol. 31, no. 1, pp. 89–95, 1998.

[5] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, “A survey of literature on the teaching of introductory programming,” 2007, p. 204.

[6] TIOBE.com, “TIOBE Software: TIOBE Index,” 27-Aug-2014. [Online]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Accessed: 27-Aug-2014].

[7] LangPop.com, “Programming Language Popularity,” 27-Aug-2014. [Online]. Available: http://langpop.com/. [Accessed: 27-Aug-2014].

[8] G. de Montmollin, “The Transparent Language Popularity Index,” 27-Aug-2014. [Online]. Available: http://lang-index.sourceforge.net/. [Accessed: 27-Aug-2014].

[9] M. de Raadt, R. Watson, and M. Toleman, “Language tug-of-war: industry demand and academic choice,” in *Proceedings of the fifth Australasian conference on Computing education - Volume 20*, Adelaide, Australia, 2003, pp. 137–142.

[10] J. Lewis, “Myths about object-orientation and its pedagogy,” *ACM SIGCSE Bull.*, vol. 32, no. 1, pp. 245–249, Mar. 2000.

[11] L. F. Capretz, “A brief history of the object-oriented approach,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, p. 6, Mar. 2003.

[12] C. T. J. T. F. on C. Curricula, Ed., “Computing curricula 2001,” *J Educ Resour Comput*, vol. 1, no. 3es, p. 1, 2001.

[13] K. B. Bruce, “Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list,” *ACM SIGCSE Bull.*, vol. 37, no. 2, p. 111, Jun. 2005.

[14] S. Cooper, W. Dann, and R. Pausch, “Teaching objects-first in introductory computer science,” in *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, New York, NY, USA, 2003, pp. 191–195.

[15] V. K. Proulx, J. Raab, and R. Rasala, “Objects from the beginning - with GUIs,” in *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, New York, NY, USA, 2002, pp. 65–69.

[16] A. Ehlert and C. Schulte, “Empirical comparison of objects-first and objects-later,” in *Proceedings of the fifth international workshop on Computing education research workshop*, Berkeley, CA, USA, 2009, pp. 15–26.

[17] A. Ehlert and C. Schulte, “Comparison of OOP first and OOP later: first results regarding the role of comfort level,” in *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, New York, NY, USA, 2010, pp. 108–112.

[18] P. J. Burton and R. E. Bruhn, “Teaching programming in the OOP era,” *ACM SIGCSE Bull.*, vol. 35, no. 2, p. 111, Jun. 2003.

[19] S. Reges, “Back to basics in CS1 and CS2,” in *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, Houston, Texas, USA, 2006, pp. 293–297.

[20] M. Kölling and J. Rosenberg, “Guidelines for teaching object orientation with Java,” *SIGCSE Bull*, vol. 33, no. 3, pp. 33–36, 2001.

[21] Jan Hylén, “Open Educational Resources: Opportunities and Challenges,” Organisation for Economic Co-operation and Development, 2006.

[22] V. Ferrara, A. Macchia, S. Sapia, and F. Lella, “Cultural heritage open data to develop an educational framework,” in *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*, Chania Crete, Greece, 2014, pp. 166–170.

[23] R. Buckland, “Open teaching a case study on publishing lecture videos publicly,” in *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*, Perth, Australia, 2011, pp. 19–28.

[24] P. J. Guo and K. Reinecke, “Demographic differences in how students navigate through MOOCs,” in *Proceedings of the first ACM conference on Learning @ scale conference*, New York, NY, USA, 2014, pp. 21–30.

[25] “Case Western Reserve University - MOOCs/free online courses list | Class Central.” [Online]. Available: https://www.class-central.com/university/casewestern. [Accessed: 29-Jan-2015].

[26] “Open Course Library – Home.” [Online]. Available: http://opencourselibrary.org/. [Accessed: 29-Jan-2015].

[27] Tiago Cinto, Harlei M. A. Leite, Cecilia S. A. Peixoto, and Dalton S. Arantes, “Virtual Learning Environments: Proposals for Authoring and Visualization of Educational Content,” *Int. J. Digit. Inf. Wirel. Commun. IJDIWC*, vol. 4, no. 3, pp. 387–400, 2014.

[28] M. J. Lage, G. J. Platt, and M. Treglia, “Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment,” *J. Econ. Educ.*, vol. 31, no. 1, pp. 30–43, Jan. 2000.

[29] J. L. Bishop and M. A. Verleger, “The flipped classroom: A survey of the research,” in *ASEE National Conference Proceedings, Atlanta, GA*, 2013.

[30] “MIT OpenCourseWare | Free Online Course Materials.” [Online]. Available: http://ocw.mit.edu/index.htm. [Accessed: 08-Jan-2015].

[31] “Khan Academy,” *Khan Academy*. [Online]. Available: http://www.khanacademy.org. [Accessed: 08-Jan-2015].

[32] “Advance Your Career Through Project-Based Online Classes - Udacity.” [Online]. Available: https://www.udacity.com/. [Accessed: 08-Jan-2015].

[33] “Coursera,” *Coursera*. [Online]. Available: https://www.coursera.org/. [Accessed: 08-Jan-2015].

[34] “edX,” *edX*. [Online]. Available: https://www.edx.org/. [Accessed: 08-Jan-2015].

[35] J. A. Day and J. D. Foley, “Evaluating a Web Lecture Intervention in a Human-Computer Interaction Course,” *IEEE Trans. Educ.*, vol. 49, no. 4, pp. 420–431, Nov. 2006.

[36] G. S. Mason, T. R. Shuman, and K. E. Cook, “Comparing the Effectiveness of an Inverted Classroom to a Traditional Classroom in an Upper-Division Engineering Course,” *IEEE Trans. Educ.*, vol. 56, no. 4, pp. 430–435, Nov. 2013.

[37] S. Zappe, R. Leicht, J. Messner, T. Litzinger, and H. W. Lee, "'Flipping' the classroom to explore active learning in a large undergraduate course," in *American Society for Engineering Education*, 2009.

[38] D. Horton, M. Craig, J. Campbell, P. Gries, and D. Zingaro, "Comparing Outcomes in Inverted and Traditional CS1," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, New York, NY, USA, 2014, pp. 261–266.

[39] K. Lockwood and R. Esselstein, "The Inverted Classroom and the CS Curriculum," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2013, pp. 113–118.

[40] P. Järvinen, *On research methods*. Tampere, Finland: Opinpajan Kirja, 2004.

[41] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, Dec. 2007.

[42] E. Vanhala, M. Määttänen, and K. Smolander, "In-service Promotion as a Business Model for Social Web Applications," presented at the Advances in Business-Related Scientific Research Conference 2013, Venice, Italy, 2013.

[43] T. Metsä-Tokila, "Näkemyksestä menestystä: Ohjelmistoala," MEE Business Sector Services, Jun. 2014.

[44] M. Kölling, "The problem of teaching object-oriented programming, Part I: Languages," *J. Object-Oriented Program.*, vol. 11, no. 8, pp. 8–15, 1999.

[45] QS, "QS World University Rankings 2013," 09-Feb-2014. [Online]. Available: http://www.topuniversities.com/university-rankings/world-university-rankings/2013#sorting=rank+region=+country=+faculty=+stars=false+search=. [Accessed: 09-Feb-2014].

[46] The Times, "World University Rankings 2013-2014." [Online]. Available: http://www.timeshighereducation.co.uk/world-university-rankings/2013-14/world-ranking. [Accessed: 09-Feb-2014].

[47] C. Schulte and J. Bennedsen, "What do teachers teach in introductory programming?," in *Proceedings of the second international workshop on Computing education research*, New York, NY, USA, 2006, pp. 17–28.

[48] Tuotantotalouden tiedekunta, "Opinto-opas 2013-2014." Lappeenranta University of Technology, Lappeenranta, Finland.

[49] Tuotantotalouden tiedekunta, “Opinto-opas 2014-2015.” Lappeenranta University of Technology, Lappeenranta, Finland.

[50] J. Howatt, “A project-based approach to programming language evaluation,” *ACM SIGPLAN Not.*, vol. 30, no. 7, pp. 37–40, Jul. 1995.

[51] C. Sadowski and S. Kurniawan, “Heuristic evaluation of programming language features: two parallel programming case studies,” in *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*, 2011, pp. 9–14.

[52] M. Schatten, “Reasonable Python or how to Integrate F-Logic into an Object-Oriented Scripting Language,” in *Intelligent Engineering Systems, 2007. INES 2007. 11th International Conference on*, Budapest, Hungary, 2007, pp. 297–300.

[53] F. Culwin, “Object imperatives!,” *SIGCSE Bull*, vol. 31, no. 1, pp. 31–36, 1999.

[54] D. Ruiqing and L. Xiaohui, “Discussions on the teaching of C++ Programming Language in colleges,” in *Artificial Intelligence and Education (ICAIE), 2010 International Conference on*, Hangzhou, China, 2010, pp. 579–581.

[55] J. Gosling and H. McGilton, “The Java Language Environment.” Oracle, 1997.

[56] J. Liberty, *Programming C#: Building. NET Applications with C#.* O’Reilly Media, Inc., 2005.

[57] S. Wiltamuth and A. Hejlsberg, “C# Language Specification.” [Online]. Available: http://msdn.microsoft.com/en-us/library/aa645596(v=vs.71).aspx. [Accessed: 27-Oct-2014].

[58] S. Reges, “Can C# replace java in CS1 and CS2?,” in *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, New York, NY, USA, 2002, pp. 4–8.

[59] A. C. Kay, “The early history of Smalltalk,” in *The second ACM SIGPLAN conference on History of programming languages*, New York, NY, USA, 1993, pp. 69–95.

[60] P. W. Lount, “Smalltalk.org | versions | ANSIStandardSmalltalk.html.” [Online]. Available: http://www.smalltalk.org/versions/ANSIStandardSmalltalk.html. [Accessed: 27-Oct-2014].

[61] J. A. Gallud, R. Tesoriero, and P. Gonzalez, “Smalltalk: The leading language to learn object-oriented programming,” in *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, 2012, pp. 839–840.

[62] C. Severance, “Bertrand Meyer: Software Engineering and the Eiffel Programming Language,” *Computer*, vol. 45, no. 9, pp. 6–8, Sep. 2012.

[63] B. Meyer, *Touch of class: learning to program well with objects*. New York: Springer, 2013.

[64] M. Kölling, “The problem of teaching object-oriented programming, Part II: Environments,” *J. Object-Oriented Program.*, vol. 8, 1999.

[65] L. Nackman, “A Brief History of Eclipse.” [Online]. Available: http://www.ibm.com/developerworks/rational/library/nov05/cernosek/. [Accessed: 31-Oct-2014].

[66] J. McAffer, *Eclipse Rich Client Platform*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2010.

[67] netbeans.org, “A Brief History of NetBeans.” [Online]. Available: https://netbeans.org/about/history.html. [Accessed: 31-Oct-2014].

[68] netbeans.org, “NetBeans IDE - Overview.” [Online]. Available: https://netbeans.org/features/index.html. [Accessed: 31-Oct-2014].

[69] jetbrains.com, “JetBrains :: Company History and Timeline.” [Online]. Available: https://www.jetbrains.com/company/history.jsp. [Accessed: 31-Oct-2014].

[70] jetbrains.com, “JetBrains IntelliJ IDEA Plugin Repository.” [Online]. Available: https://plugins.jetbrains.com/?idea. [Accessed: 31-Oct-2014].

[71] jetbrains.com, “IntelliJ IDEA :: Features.” [Online]. Available: https://www.jetbrains.com/idea/features/. [Accessed: 31-Oct-2014].

[72] “GtkSharp | Mono.” [Online]. Available: http://www.mono-project.com/docs/gui/gtksharp/. [Accessed: 30-Jan-2015].

[73] Mono Project, “MonoDevelop - Mono Wiki.” [Online]. Available: http://mono.wikia.com/wiki/MonoDevelop. [Accessed: 31-Oct-2014].

[74] Mono Project, “Feature List - MonoDevelop.” [Online]. Available: http://monodevelop.com/Documentation/Feature_List. [Accessed: 31-Oct-2014].

[75] D. Teske, “Qt Creator 1.2 released | Qt Blog.” [Online]. Available: http://blog.qt.digia.com/blog/2009/06/25/qt-creator-12-released/. [Accessed: 11-Mar-2014].

[76] Qt Project, “Category:Tools -> QtCreator | Qt Wiki | Qt Project.” [Online]. Available: http://qt-project.org/wiki/Category:Tools::QtCreator. [Accessed: 11-Mar-2014].

[77] Qt Project, “QtCreatorWhitepaper | Qt Wiki | Qt Project.” [Online]. Available: http://qt-project.org/wiki/QtCreatorWhitepaper. [Accessed: 11-Mar-2014].

[78] M. Ronchetti, “Using video lectures to make teaching more interactive,” *Int. J. Emerg. Technol. Learn. IJET*, vol. 5, no. 2, 2010.

[79] P. Hubwieser and M. Berges, “Minimally invasive programming courses: learning OOP with(out) instruction,” 2011, p. 87.

[80] A. Herala, E. Vanhala, and U. Nikula, *Olio-ohjelmointi Javalla, versio 1.0*. Lappeenranta: LUT, 2015.

[81] A. Herala, E. Vanhala, and U. Nikula, *Olio-ohjelmointi käytännössä käyttäen hyväksi avointa tietoa, graafista käyttöliittymää ja karttaviitekehystä, versio 1.0*. Lappeenranta: LUT, 2015.

[82] A. Bieniusa, E. Knauel, M. Degen, P. Heidegger, P. Thiemann, S. Wehr, M. Gasbichler, M. Sperber, M. Crestani, and H. Klaeren, “Htdp and dmda in the battlefield: a case study in first-year programming instruction,” 2008, p. 1.

[83] A. Begel and N. Nagappan, “Pair programming: what’s in it for me?,” in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, Kaiserslautern, Germany, 2008, pp. 120–128.

[84] J. Bennedsen and M. E. Caspersen, “Teaching object-oriented programming-Towards teaching a systematic programming process,” in *Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts. Affiliated with 18th European Conference on Object-Oriented Programming (ECOOP 2004)*, 2004.

[85] W.-K. Chen and Y. C. Cheng, “Teaching Object-Oriented Programming Laboratory With Computer Game Programming,” *IEEE Trans. Educ.*, vol. 50, no. 3, pp. 197–203, Aug. 2007.

[86] K. Sanders and L. Thomas, “Checklists for grading object-oriented CS1 programs: concepts and misconceptions,” in *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY, USA, 2007, pp. 166–170.

[87] W. B. Hirschmann, “Profit From the Learning Curve,” *Harv. Bus. Rev.*, vol. 42, no. 1, pp. 125–139, 1964.

[88] V. A. Cicirello, “On the Role and Effectiveness of Pop Quizzes in CS1,” in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2009, pp. 286–290.

[89] J. Maxwell, “Understanding and Validity in Qualitative Research,” *Harv. Educ. Rev.*, vol. 62, no. 3, pp. 279–301, Sep. 1992.

[90] A. S. Lee and R. L. Baskerville, “Generalizing Generalizability in Information Systems Research,” *Inf. Syst. Res.*, vol. 14, no. 3, pp. 221–243, Sep. 2003.

APPENDIX 1. Finnish universities

University	Language		Objects?		Scheduled in curriculum		Additional object-oriented course
	CS1	CS2	CS1	CS2	CS1	CS2	
Aalto university[1] [2]	Scala	Scala	Yes	No	1. year, 1.-2. period	1. year, 3.-4. period	No
Lappeenranta University of Technology [3] [4]	Python	C	No	No	1. year, 1.-2. period	1. year, 3.-4. period	Yes / C++
Tampere University of Technology [5] [6]	Python	C++	Yes	Yes	1. year, 3.-4. period	2. year, 1.-2. period	No
University of Helsinki [7] [8]	Java	Java	Yes	Yes	1. year, 1. period	1. year, 2. period	No
University of Eastern Finland [9] [10]	Python	Java	Yes	Yes	1. year, 1.-2. period	1. year, 4. period	No
University of Jyväskylä [11] [12]	C#	Java	No	Yes	1. year, 1.-2. period	1. year, 3.-4. period	No
University of Oulu [13] [14]	Python	C	No	No	1. year, 1.-3. period	1. year, 4.-6. period	Yes / C++
University of Tampere [15] [16]	Java	Java	No	Yes	1. year, 1.-2. period	1. year, 3. period	No
University of Turku [17] [18]	Java	Java	No	Yes	1. year, 2. period	1. year, 3. period	No
University of Vaasa [19]	Java	Java	No	Yes	1. year, 1. period	1. year, 3. period	No
Åbo Akademi University [20]	Java	Java	No	Yes	1. year, 1.-2. period	1. year, 3.-4. period	No

1 https://noppa.aalto.fi/noppa/kurssi/cse-a1110/esite
2 https://noppa.aalto.fi/noppa/kurssi/ics-a1120/esite
3 https://noppa.lut.fi/noppa/opintojakso/ct60a0200/etusivu
4 https://noppa.lut.fi/noppa/opintojakso/ct60a0210/etusivu
5 http://www.tut.fi/wwwoppaat/opas2014-2015/perus/laitokset/Tietotekniikka/TIE-02100.html
6 http://www.tut.fi/wwwoppaat/opas2014-2015/perus/laitokset/Tietotekniikka/TIE-02200-1.html
7 http://mooc.cs.helsinki.fi/ohjelmoinnin-perusteet
8 http://www.cs.helsinki.fi/group/java/k13/ohja/
9 https://weboodi.uef.fi/weboodi/opintjakstied.jsp?html=1&Kieli=1&Tunniste=3621217
10 https://weboodi.uef.fi/weboodi/opintjakstied.jsp?html=1&Kieli=1&Tunniste=3621367
11 https://korppi.jyu.fi/kotka/course/student/generalCourseInfo.jsp?course=148505&javascript-enabled=true
12 https://korppi.jyu.fi/kotka/course/student/generalCourseInfo.jsp?course=148492
13 https://noppa.oulu.fi/noppa/kurssi/521141p
14 https://noppa.oulu.fi/noppa/kurssi/521142a/esite
15 http://www.uta.fi/sis/tie/laki/opetussuunnitelma.html
16 http://www.uta.fi/sis/tie/oope/opetussuunnitelma.html
17 https://nettiopsu.utu.fi/opas/opintojakso.htm?rid=22978&idx=2&uiLang=fi&lang=fi&lvv=2014
18 https://nettiopsu.utu.fi/opas/opintojakso.htm?rid=19164&idx=0&uiLang=fi&lang=fi&lvv=2014
19 http://lipas.uwasa.fi/opinto-opas/di/tite.html
20 http://www.abo.fi/student/Content/Document/document/29686

APPENDIX 2. International universities

University	Language		Objects?		Additional object-oriented course
	CS1	CS2	CS1	CS2	
Columbia University [21] [22]	Java	C/ C++/Perl	Yes	No	No
Cornell University [23] [24]	Python	Java	Yes	Yes	Yes / Java
California Institute of Technology (Caltech) [25] [26]	Python	Scheme	Yes	No	No
ETH Zürich – Swiss Federal Institute of Technology Zürich [27] [28]	Eiffel	Java/C#	Yes	No	No
Harvard University [29] [30]	C, PHP, JavaScript	OCaml	No	Yes	No
Imperial College London [31] [32]	Haskell	Java	No	Yes	No
Johns Hopkins University [33]	Java	C/C++	No	Yes	Yes / C++
Massachusetts Institute of Technology (MIT) [34] [35]	Python	Python	No	Yes	No
Princeton University [36] [37]	Java	C	No	No	Partial / C++ [38]
Stanford University [39] [40]	Java	C++	No	No	Partial [41]
University of Cambridge [42]	ML	Java	No	Yes	Parallel with CS2
University of Chicago [43] [44]	Racket	C++	No	Yes	No
University of Oxford [45] [46]	Haskell	Scala	No	No	Yes / Java
University of Pennsylvania [47] [48]	Java	Ocaml/ Java	No	Yes	No
University of Toronto [49] [50]	Python	Python	No	Yes	No
Yale University [51] [52]	Java	Racket	Yes	No	No

21 http://www.cs.columbia.edu/mice/classes/showSessions.php?courseID=5&term=fall&schoolyear=2014&public=1&base=%2Fmice%2Fclasses%2F&
22 http://www.cs.columbia.edu/mice/classes/showSessions.php?courseID=86&term=fall&schoolyear=2014&public=1&base=%2Fmice%2Fclasses%2F&
23 http://courses.cornell.edu/preview_course_nopop.php?catoid=22&coid=336829
24 http://courses.cornell.edu/preview_course_nopop.php?catoid=22&coid=336841
25 http://catalog.caltech.edu/courses/listing/cs.html
26 http://www.scribd.com/doc/206357076/Lecture-01
27 http://www.vvz.ethz.ch/Vorlesungsverzeichnis/lerneinheitPre.do?lerneinheitId=93673&semkez=2014W&lang=en
28 http://www.vvz.ethz.ch/Vorlesungsverzeichnis/lerneinheitPre.do?lerneinheitId=89978&semkez=2014S&lang=en
29 http://d2o9nyf4hwsci4.cloudfront.net/2014/spring/lectures/0/w/syllabus/syllabus.html
30 https://docs.google.com/document/d/1Y1n-RR5j_FQ9WFMG8E_ajO0OpWLNANRu4lQCxTw9T5g/edit
31 http://www3.imperial.ac.uk/computing/teaching/courses/120_1
32 http://www3.imperial.ac.uk/computing/teaching/courses/120_2
33 http://e-catalog.jhu.edu/departments-program-requirements-and-courses/engineering/computer-science/#undergraduatetext
34 http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00sc-introduction-to-computer-science-and-programming-spring-2011/
35 http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011/
36 http://www.cs.princeton.edu/courses/archive/fall14/cos126/info.html
37 https://www.cs.princeton.edu/courses/archive/spring14/cos217/index.html
38 http://www.cs.princeton.edu/courses/archive/spring14/cos333/
39 http://web.stanford.edu/class/cs106a/
40 http://see.stanford.edu/see/courseinfo.aspx?coll=11f4f422-5670-4b4c-889c-008262e09e4e
41 http://see.stanford.edu/see/courseinfo.aspx?coll=2d712634-2bf1-4b55-9a3a-ca9d470755ee
42 http://www.cl.cam.ac.uk/teaching/1415/CST.pdf
43 http://www.classes.cs.uchicago.edu/archive/2013/fall/10500-1/
44 http://www.classes.cs.uchicago.edu/archive/2014/spring/10600-1/
45 http://www.cs.ox.ac.uk/teaching/courses/2014-2015/fp/
46 http://www.cs.ox.ac.uk/teaching/courses/2014-2015/imperativeprogramming1/
47 http://www.cis.upenn.edu/~cis110/14fa/syllabus.html
48 http://www.seas.upenn.edu/~cis120/current/syllabus.shtml
49 http://www.cdf.utoronto.ca/~csc108h/winter/
50 http://www.cs.toronto.edu/~fpitt/20131/CSC148/index.html
51 http://zoo.cs.yale.edu/classes/cs112/cs112-2014-spring/course_info.shtml
52 http://zoo.cs.yale.edu/classes/cs201/