Object-oriented programming course revisited

Antti Herala Erno Vanhala Uolevi Nikula
Lappeenranta University of Technology
Department of Innovation and Software
P.O.Box 20
FI-53851 Lappeenranta, Finland
antti.herala@lut.fi erno.vanhala@lut.fi uolevi.nikula@lut.fi

ABSTRACT
Teaching has faced challenges over the latest decades. It is easier than ever to provide material for students and get returned exercises and hold exams online. Besides technology and platforms, also teaching methods need to adapt to the Internet-age and its generation of people. In this article we present a case study where an university level programming course was upgraded to fit the needs of 2010s by introducing Java as the predominate language and utilizing available technologies to enhance teaching. This was done on both technological and pedagogical level, introducing open data and flipped classroom to programming education while the scope remained unchanged. This article presents the first results of the new course. Based on the collected student feedback, the use of Java and open data and implementations of the flipped classroom teaching method are all considered as a success.

Categories and Subject Descriptors
K.3.2 [**Computer and Education**]: Computer and Information Science Education – *computer science education.*

General Terms
Management, Measurement, Documentation, Experimentation

Keywords
Flipped classroom, reversed classroom, programming, teaching, Java, open data

1. INTRODUCTION
Teaching programming has long had two sides in sense that besides theoretical knowledge learning programming requires getting hands dirty with the actual programming. Teaching in universities has met paradigm shifts over the decades and the latest occurred when lecturing and providing material became available via the Internet. Besides pedagogical issues there is always the question what programming languages and integrated development environment one should use [6, 34]. Furthermore, the industry has its own special expectations, and universities teaching engineers are keen on meeting these expectations [13].

Koli Calling 2015, November 19-22, 2015, Koli, Finland
 ISBN 978-1-4503-4020-5/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2828959.2828974

In addition to technical selection there is also the pedagogical aspect and although lecturing has been effective way to teach students since the dawn of universities it does not mean that there would not be better ways. The flipped classroom method where lectures are replaced with self-study material has been gaining popularity in several areas of education [42, 57] including the teaching of programming [27, 36].

As our university in Finland decided to change the length of a semester from fourteen to twelve regular teaching weeks and the shift from a Nokia-driven industry to a land of computer game companies brought us the option to change the programming language and improve the course. We decided to re-evaluate all the fundamental elements of the course, including language, tools and teaching methods.

This study focuses on the context of programming, especially experiences of running a programming course, where the teaching focuses on practical education instead of delivering the theory. Students with adequate knowledge about basic programming should understand the theory mostly by themselves but require guidance with practical implementations, regardless of the language used.

In the end this case study is looking for an answer to the questions *what are the existing recommendations to teach fundamentals of object-oriented programming and how they suited our course?* This article also presents the results of the first implementation of the course.

2. RELATED RESEARCH

2.1 Object-oriented versus other paradigms
Object-oriented programming (OOP) paradigm builds on objects [7]. The complexity to teach object-orientation is emphasized by the concepts of objects-first and objects-later, a division started by the publication of Computer Curricula 2001 [53], where objects-first was officially introduced as a formal method for education. The approach to object-oriented education has been studied a lot [6, 10], yet studies [14, 15] argue how the difficult concepts are difficult despite the method.

While the object-oriented paradigm was introduced as early as in 1970s [7], it started to make its way to programming education as late as in the 90s [11]. In the early 21st century and later, the researchers around programming education have developed models and guidelines for the course structure and topics. Based on the previous research, multiple tools have been considered for the new course, such as a model-driven approach [3], game-based design [8] and a checklist for grading [51]. In this study the guidelines for an object-oriented programming course [34] are considered to serve as the basic structure, because they give the freedom to select components of the course independently without the need to follow a strict set of rules. The guidelines and their descriptions are presented in Table 1.

2.2 Programming language and IDE for object-oriented programming

Needs from the industry, pedagogical guidelines, available teaching material and teachers' own experiences and skills make a mix that leads to variety of courses and programming languages [44]. One of the most heated questions in computer science education is the selection of a programming language for the course, whether it is a basic CS1 course or more advanced programming course. The programming languages used in programming courses can have an impact to the programmers career and especially their advancement while studying [13]. There exists also global (e.g. the increase of web and mobile apps [20, 25]) and local factors (e.g. presence of a major employer, such as Nokia in Finland or Samsung in South-Korea) that affect the decision of which programming languages and tools are taught [49]. The balance that the educators strive to find is between the demand from the industry and the necessary functionalities for academic purposes. While the educators would rather see a programming language that is pedagogically the most suitable for learning [13], the language is usually selected based on the demand from the industry [49]. Various surveys point out how C, C++ and Java have been the most widely used programming languages in the educational landscape [44, 47, 48], similarly to the industrial preferences as statistics show [46, 54, 56].

Just like for the methods to teach object-orientation, the language for the course has been discussed in many publications. The selection criteria vary from a specific, heuristic model [50] to criteria based evaluation [28]. The requirements are presented in Table 2 with explanations.

2.3 Flipped classroom method

The technological development has allowed the education methods to improve. One of such methods is the flipped classroom (a.k.a. reversed classroom), that was initially developed and used in economics [35]. The gist of the method is to allow students to study the theory outside class with computer-assisted tools and concentrate on practicing in class with the instructor. The method is argued to allow students to adapt their individual study patterns to the material and they are not required to be present at traditional one-to-many lectures [4].

In recent years the flipped classroom has acquired more visibility in the educational landscape as it has been used to teach topics in mathematics [1, 37], biology [42], introductory business [52], industrial engineering [57], mechanical engineering [40] and computer science both in introductory [27, 36] and advanced levels [21, 38].

Table 1. Guidelines for OOP course [34]

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 existing code 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.
Don't start with "main"	The main()-function has no relation to object-oriented programming, it is only a point of communication from Java to operating system.
Don't use "Hello World"	"Hello World" does not present OOP and the use of objects is not clear for students.
Show program structure	The relation between objects, classes, and the structure should be visually presented.
Be careful with the UI	GUI is an easily distracting component of a program and not relevant to OOP.

The use of flipped classroom has been experienced positively by students, giving positive feedback about the method used [38]; at the same time the learning results have also improved [40]. While the results have been mostly positive, studies show that the acceptance-level from students is not unanimous [19] and the learning results do not always improve [21].

2.4 Open data in education

Openness is defined by Open Definition [43] as "*Open data and content can be freely used, modified, and shared by anyone for any purpose*". Following the definition of openness, open data is defined by Open Data Institute [58] as "*Open data is data that anyone can access, use and share*".

Table 2. Requirements for programming language in object-oriented education [32]

Requirement	Explanation
Clear concepts	The language should be implemented with enough abstraction so it does not contradict with the teaching.
Transition to other languages	The language enables students to understand programming concepts instead of an exact programming language.
High level	Tasks available to be executed by the compiler or the runtime environment should be automated.
No redundancy	The language should have one solution per problem. Different tools to solve same problems adds confusion.
Pure object-orientation	The language should only support object-orientation and should not permit other programming paradigms.
Readable syntax	The syntax should be easy to read and comprehend.
Safety	Errors can be easily and automatically detected. The error messages should be clear and easy to interpret.
Simple object model	The execution and object allocation should be easy to understand step by step, even for a beginner.
Small	The language should be as compact as possible, but still powerful enough to be usable in education.
Suitable environment	The environment should support multiple operating systems and the object-oriented paradigm.
Correctness assurance	Students should be aware of the software engineering principles and the language should support them.

The link between data and research has been noticed in industry and open data has had a large impact on university research [45]. This has led the researchers to understand, that the open data may not be only relevant to research but to education as well [5, 16].

The connection of open data and education is usually understood as open educational resources (OER) and rarely as a tool to enhance learning [29]. However, some definitions of OER contain all of the open materials used in classes, which could technically cover open data as well [22]. Following the idea behind OER, open data can be used to enhance learning through a common platform, that connects public bodies with universities, such as museums to cultural heritage studies [16]. The use of open data is a two-way symbiosis: the data is used to educate students about the real life situations they require for their studies but at the same time they validate and curate the existing data [5].

3. RESEARCH PROCESS

The data used in the study comes from the third programming course (CS3) held at Lapppeenranta University of Technology (LUT), Finland, between 2010 and 2014. In this paper the results focus on the differences between years 2010-2013 and 2014, where the major change was made. Data was collected from various sources: each course included a final survey that was distributed to all students enrolled in the course, approximately half of the programs the students developed during these courses have been stored in the database of a virtual learning environment (VLE), and teacher also kept statistics on course projects and their problems. The final surveys have included quantitative questions such as the difficulty and usefulness of different course components (e.g. lectures and course project) but also open questions were available to enable students to give both positive and negative feedback on the topics they saw important.

In general the study had three objectives: First, to report internal and external reasons to reconstruct the course. Second, to demonstrate how the chosen tools and methods can be used to improve the course. And third, to report how the course was remodeled in 2014 and how it was received by students. Thus the study has the characteristics of both natural and design science [39] but it is reported as a case study since the case study methods [59] fit well the study. Case studies aim to understand the problem under investigation thus the repeatability and generalizability can be low but discoverability and representability high [18].

Meyer (2001) present choices (selection of cases, sampling of cases, unit of analysis, sampling time, sampling business areas, and divisions, and sites) that need to be considered when designing a case study [41]. In addition Meyer (2001) also presents options for data collection procedures, data analysis, and validity and reliability. These are discussed here shortly.

As this is a single case study the selection of cases choice is considered straightforward as we are reporting the object-oriented programming course held in Lappeenranta University of Technology during the years 2010 - 2014. The same happens with sampling of cases as we have only one case and it presents a case that could be replicated all over the world thus the sampling is also straightforward. Meyer (2001) continues pointing out how the unit of analysis can be selected so that there are various units under observations thus controlling the researchers' and respondents' bias. In our study we are not focusing only on topic of interest but researching out several key issues. To issue sampling time we utilized approach where we used data collected after every iteration of the course. Sampling business areas, divisions, and sites is not considered relevant in our study as we are studying only one course with 5 iterations, so these samplings would be useful if we had wider data sources including for example geographical, time, or size variations.

For *selection of data collection procedures* we utilized course feedback survey, which gave us both quantitative and qualitative data. In addition teacher gave us his data on how much time he had spend on various teaching tasks and how he saw the course evolution. For *data analysis* we used spreadsheet program to calculate averages and other key figures as suggested by Fink (2013) [17]. *Validity and reliability* presented by Meyer (2001) is discussed in the end of Discussion section.

4. CASE STUDY

In our university the curriculum has been that CS1 and CS2 are taught with Python and C respectively and both courses use procedural paradigm. The first programming course in the second year is taught completely with objects and this case study presents how the new version of the course was constructed and what kind finding did we recognize when running the first implementation of the course. Students do not have any previous knowledge about objects and their interactions; they've used objects only as data storages with Python in CS1. Similarly methods are only known from use of methods of existing objects, such as string methods in Python.

4.1 The previous version of the course

The Object-Oriented Programming course has had multiple lectures and assistants over the years and they have emphasized different topics and tools in the course. The focus has still always been to teach object-orientation with C++ for the second year students who have the basic knowledge and understanding of programming as they have passed CS1 and CS2. The goals of the course 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*".

The course syllabus consisted of three main parts: object-oriented lectures, C++ lectures and their hybrid. The pure object-oriented lectures focused on objects and classes, inheritance, interfaces and object-oriented design with programming examples. The C++ lectures focused on variables, pointers, references, functions, namespaces, and C++ models with standard template library (STL). The hybrid lectures were focused on copying and assignment, exceptions, and error handling.

The course was built traditionally based on weekly lectures and exercises. In paper both sessions took up to 90 minutes, while lectures were usually shorter and exercises longer than allocated. In addition to physical lectures, the lectures were also recorded and shared as video via YouTube (in 2013) or as downloadable video and audio file (before 2013). The course also had a course book, that was not referenced in the lectures and was up to students to read. In addition to weekly exercises, the course had a larger project, that the students were expected to complete before the final exam. In the project, pair working was allowed and encouraged, leading to most of the students completing the project in pairs.

Since 2010, the lecturer had not changed and thus, the improvements and their possible impacts had been systematically documented. In Table 3 the previous, incremental changes made to the course are elaborated. The changes have always followed the student feedback and the lecturer's observations to make the course more and more interesting. However, the course has not been changed for the sole purpose of changing something and for example in 2012 there was no changes from the previous year.

4.2 Needs for the change

The object-oriented programming course had been taught with C++ for more than ten years and basically the only radical change happened in 2009 when the calendar length of the course was increased from seven to fourteen weeks. The external factors had changed since the beginning of this millennium and for example the shutdown of Symbian development in Nokia and growing computer game industry [25] changed the need of programming skills from C++ to include various other languages.

The original C++ course did not have any mandatory GUI design part, but today major of the software is used through GUI. Students got extra points when they returned course projects with GUI but they needed to learn the tools and techniques by themselves. Besides a clear need for a graphical user interface the course required some connections to real life situations. Open data has been lately introduced to software industry [12], when governments have been starting to publish data sets they have collected over the decades. This allow software developers to get the data for free and build their applications on the base that already exists instead of starting to collect data by themselves. It was considered to be a relevant improvement to the course's connection to the real life software engineering work and thus we decided to utilize various freely available data sources.

At the same time the university policy-makers decided to rethink teaching periods in 2013 and since 2014 the course had calendar time twelve standard teaching weeks with the option to build special events in intensive weeks where teaching can be out of normal schedule.

Although the course had gained a lot of positive feedback, it was clear that it had many outdated aspects that could be improved, one being the fact that some lectures were focusing purely on the programming language, not object-orientation. While the feedback was positive, students rarely participated in lectures and they also skipped exercises, since they were not mandatory and did not accumulate any points for them. All the issues lead to decision to completely reconstruct the course.

Table 3. Development of OOP course 2010-2013

Year	Changes
2010	New lecture slides and programming examples were created and the lectures were video recorded. The lecturer used Linux operating system with a simple text editor, while students made tasks in Windows.
2011	The tasks were done using Linux and the text editor was replaced with NetBeans IDE in both lectures and exercises. Students were also given the freedom to choose different IDE. A graphical user interface (GUI) example with Qt was included into the course. The lectures were offered in video format and in mp3-format. Non-mandatory essay was added as a bonus.
2012	The course remained the same from the previous year.
2013	The recordings of the lectures were uploaded into YouTube, so students could watch the lectures with any device. Students were introduced to a lecture bingo [55] that students could play while listening the lectures. The GUI example was expanded from previous years and students were encouraged to use version control, introduced in CS2. It was not mandatory, but was awarded with extra points. The project could be returned by sharing a repository with the lecturer.

4.3 Selection of new tools, techniques and methods

In the very beginning of the upgrade process people responsible of the upgraded course discussed that flipped classroom method might be a valid method to try out, especially since the existing research supports utilizing the method in programming courses [27, 36]. It was decided that traditional lectures would be replaced with short video recordings and the effort would be focused on exercises. It was realized that it might bring problems and resistance might rise. Another point to discuss with the classroom flip was the possibility to use external, existing lecture videos instead of creating all from scratch, a strategy presented by Maher et al. (2015) [38]. Baldwin (2015) suggests lecture videos should be custom made to provide as much aid to students as possible.

The second issue to consider was the programming language. As the emphasizes of the industry had shifted in the local area there was also possibility to change programming language from C++ to some other. To select a language for the new course in LUT, a set of 11 requirements [32] were used to evaluate multiple languages. The languages had to fill at least one condition to be evaluated: the language was used by students before this course, it should support object-orientation as much as possible or it should have a considerable market segment. The selected languages and the review can be seen in Table 4.

Students have been familiarized to Python in the first programming course from the procedural point of view. While being the best language for this course in the mapping, it is a risky choice to consider, since the language does not enforce object-orientation. The second place in the Table 4 is taken by Java, which is a language designed for object-orientation and is also relevant to the software industry.

Java is currently a popular language and it is supported by multiple popular IDE's. There are currently only two relevant free and open source programming environments to be considered, that are in wide use in the industry: Eclipse and Netbeans. Netbeans is supported by Oracle, which releases versions of Java

Table 4. Our survey of the languages based on framework presented in [32] (x=covers)

	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

and Netbeans simultaneously, ensuring continuous support. This simultaneous release and easiness to install at the same time makes Netbeans favorable over Eclipse, allowing students an easier setup by themselves.

4.4 Created materials

In support of the flipped classroom method, the course had multiple lecture videos. The videos were constructed to present one topic per video and each lecture video was constructed to last only 15-30 minutes, as advised in [21, 60]. However, some lectures were longer. This happened with for example UML, since there was no reason to divide one topic into multiple parts.

To provide students text-based material, custom made manuals in Finnish were added to the course. CS1 and CS2 in LUT have custom made manuals to support learning and they have been found useful [30], so it was reasonable to build manuals for the third course too. The manuals were divided into two major topics: object-orientation and practical software development.

The first manual presented object-orientation as objectively as possible, providing examples with Java [23]. The language constraints were ignored, the manual presents, for example, how and why destructors are made, while Java does not support destructors. Each section in the manual had the object-oriented concepts presented, if necessary, with code and each section ended with Java tutorials about the week's main topics. Some of the sections were more about programming than object-orientation, such as the section about libraries and data streams or reading data from the Internet, but they were necessary for basic object-orientation and therefore included in the first manual.

The second manual concentrated more on the software development, spanning over topics like version control, XML and GUI development [24]. In the manual were also sections about open data and developing map-based programs. Practically the second manual consisted of topics that were used in the new course project: open data as the data source, XML as the data structure, GUI as the interface, programs with geolocation data, and version control as the teamwork tool used also to return the project to the teacher.

4.5 Syllabus for the new course

In the new syllabus, the focus is being kept in object-orientation and Java is taught only when necessary. The advantage of short video lectures is the possibility to concentrate on one topic at a time. This way the teaching about the language can be separated from the more theoretical lectures, leaving more room to the theory. In Table 5 the new syllabus is presented with additional information about the length and number of the lecture videos.

The new course was built using the set of 11 guidelines presented in Section 2. In Table 6 the guidelines are reported and their effects on the new course are described. Some guidelines are followed more strictly than others, because of some limiting factors and necessary tool initializations.

Table 7 presents a comparison of the old course and the new version. The switch to flipped classroom teaching method removed the physical lectures and introduced shorter educational videos and two programming manuals instead. The voluntary exercise tasks and VLE tasks were replaced with compulsory weekly tasks from which student must get at least 10 points out of 22. Where previous years weekly exercises were composed of one big task the new course had five smaller tasks every week. The main idea was to first build simple application and then increase its features. For example, in week 10 the first task was to build a program that shows the university web page in a webview component. Then it was improved and in the task five student had build a web browser with support for local and Internet files featuring back, forward and reload buttons. As there was programming tasks that included designing and implementing graphical user interface, no suitable automatic inspection method was found but instead it was decided that students need to present their solution in the exercises to earn points. This solution also reduced the risk of plagiarism as students really needed to understand their programs to demonstrate the functionalities and features.

Table 5. New course syllabus

Content	Videos (sum)
1 Introduction to OOP and course's tools	27:06, 12:19, 2:54, 18:38 (1:00:57)
2 Java and objects, user I/O and methods in Java	39:57, 22:52, 17:40 (1:20:29)
3 Data structures in Java, more about OOP, variables and constants, operations	24:58, 10:35, 11:41, 28:36 (1:15:50)
4 File I/O, serialization and libraries	10:49, 19:50, 16:28 (47:07)
5 Object-based design with UML and unit testing	51:48, 20:31, 34:53 (1:47:12)
6 Inheritance, abstraction (abstract classes/interfaces) and polymorphism	33:59, 16:20, 30:41 (1:21:00)
7 Basics of building a GUI	24:43, 39:04 (1:03:47)
8 Class variables and methods	27:48
9 Object-based design philosophy, errors and exceptions, Java, Internet, and XML	9:32, 38:30, 40:40 (1:22:42)
10 Generics in Java, iterators and their uses, connecting Java-program to Internet	13:01, 30:25 (26:19)
11 Basics about inner and anonymous classes, theory of copy and assignment	5:38, 37:24 (43:02)
12 Going over the material for the exam.	0 / 0:00

The previous version of the course did not include mandatory GUI tasks. Qt was demonstrated shortly as a recommended GUI library and some students added GUI to their course project. The first half of the course contained material for learning basic object-oriented concepts. The purpose of the second half was to enhance the basics with GUI. For example, in week 3 the task was to construct soda machine simulator from where one could buy soft drinks. In week 8 the program logic remained the same but the task was to build a new graphical user interface with JavaFX.

The larger programming project concentrated to the use of open data. Using open data set borders for the topic. The main concept was to apply maps as the basic functionality. To reach the goal of course project, the data had to be specific and useful enough. The project materialized as a postal office simulator, that relied to self-service machines. To motivate the students, the project concentrated on the map of Finland. To this map, students added visualization about the self-service machines available. The data source is open and accessible through Internet, provided by the manufacturer. To add functionality, the user could send objects in different postal packages between machines.

Table 6. New course in comparison to the guidelines

Guideline	Implementation in the new course
Objects first	Objects are taught right from the beginning by presenting commonly used paradigms and then moving into objects.
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 shown as code and explained in detail.
Use "large" projects	The first example that students see contains a combination of 4 classes. Students begin by building a program with 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 emphasized 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 a test that students can log into the visual learning environment (VLE) system and can run their project. Printing one line is the easiest method of testing 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 pre or post programming. In the fifth week, students have to construct their own UML class diagrams to complete the weekly exercise.
Be careful with the user interface	The graphical interface is presented and taken into use only after the students have adequate understanding of objects. The emphasis remains in the program logic, a satisfactory application is required before extra points from GUI are credited. The restriction aims to minimize the GUI polishing in the expense of application logic.

The previous version of the course provided all the materials in static web pages. The lecture slides, demo examples and videos were links to files in university servers. The new course changed this by publishing the material through external services. The slides were in Google Drive, demo codes in a Git repository and videos in YouTube. Exam on paper was replaced by an online exam but the traditional paper exam was also an option, if students preferred it. The paper exam followed a strict schedule while the online exam gave more freedom. The online exam could be done with any computer on a given time interval. The students who took the online exam had a two hour time slot to finish the exam. The time restriction was necessary to discourage plagiarism and group work in individual exams.

Table 7. Old version and new version of the course

	Old course	New course
Lectures	Physical and video recorded lectures, 60-90 min / week	31 short educational videos
Exercises	90 min	90 min
Exercise tasks	Voluntary, no points earned	Compulsory to achieve 10/22 points
Data sources	Self-generated	Self-generated and open data
Returning method	VLE	VLE and physical demonstration
GUI	No, Qt was only shortly demonstrated	JavaFX
Use of version control (VC)	Use of VC provided extra points	Mandatory when returning the course project
Supplementary material	Lecture slides	Two manuals and lecture slides
Example code	From university VLE as a zip-file	From a git repository in Bitbucket
Exam	On paper	Online (or on paper)

4.6 Results

The university requires mandatory feedback collection from every course. The Object-oriented programming course had earned quite high average grade as Table 8 illustrates. Although everything was changed the total grade was still the highest ever. The measurements focus on the student satisfaction and the learning environment, both quantitatively and qualitatively.

Table 8. Course average grade from students (1=lowest, 5=highest)

	2010 (N=19)	2011 (N=18)	2012 (N=19)	2013 (N=14)	2014 (N=19)
Average grade for course	4.00	3.94	4.37	4.36	4.47

Students opinions on all the components in the course were asked quantitatively. Their graded usefulness are presented in Figure 1. By offering the exercises as the only in-class activity, their reported usefulness increased significantly.

The course started with a short lecture in September 12th, introducing the schedule and methods. The first exercises were held on 15th and every Monday after that with the exception of an exam week in mid October. Students were able to come in the exercise sessions to show their weekly progress. The new lecture

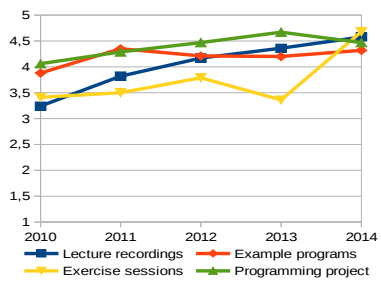

Figure 1. The usefulness of course's components (1=useless, 5=useful)

Figure 2. Number of daily views of course videos in YouTube.

videos were uploaded to YouTube in the preceding week. In Figure 2 one can note how students were watching videos on few days before exercises on the first period. After the exam week, in the second period the videos were watched several days before exercise session. The last exercises were held on December 1st but the students used the videos even after that. The statistics on YouTube illustrate how videos were watched while the students were finishing the course. The videos helped students to complete the course project and revise material for the exam on December 17th.

Figure 3 presents the number and percentage of redone course programming projects. The redone projects were not accepted by the teacher and needed revision. The current teacher started to lead the course in 2010, and the programming projects were only checked by running the program and reviewing the code. In 2011 the memory leaks were also checked and thus the redone percentage skyrocketed. In 2012 and 2013 students were taught throughly to check for memory leaks, decreasing the number of redoes. The change of programming language from C++ to Java in 2014 removed the memory management problems. However, the use of GUI and the exception handling of Java introduced new difficulties to students. Besides the challenges set by the technology, the teacher added structural requirements. The required program structure had the source code divided into reasonable classes and modules and the documentation needed to be in line with the code. With all these changes the redone percentage was still lower than in previous years.

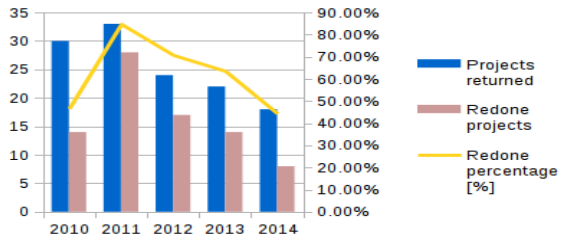

Figure 3. Number of returned and redone projects. Projects can be done by one or two students.

While some students completed the course project, there still remained problems with final grades. A major problem in the course was that not all enrolled students got a grade (Figure 4). The 100% grading is not possible but after the revision approximately half of the enrolled students got a grade. The number of students who enroll but do not start the course has decreased over the years. 2014 was the first year when all the students that returned the project or exam also got a grade. On previous years some students quit the course after the completion of programming project or exam.

The creation of new course materials required time as the manuals were written from scratch and as the programming language was changed no parts from old videos could be used and also new videos were made from scratch. Table 9 presents the teaching hours that were required from the teacher. On one hand

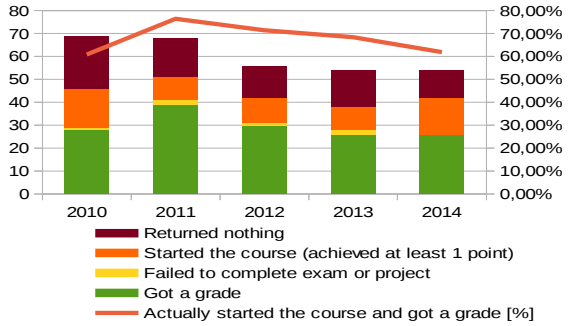

Figure 4. Statistics on the course.

time spent in exercises increased as was predicted when changed to flipped classroom method, but on the other hand lecturing took no time and evaluating course projects was faster. The overall time usage was 16 hours less than on previous year – when considering only teaching hours, not the time spend on creating new materials, which can be then used next years. The creation of 12.3 hours of video material took 36.5 hours of work.

Table 9. Overall time in hours used by the teacher

Year	Exercises	Evaluating projects	Lecturing	Whole course
2013	12.5	23.5	17.5	88.5
2014	29.5	18	0.5	72.5

From the student's point of view, the course had some successful parts but there still remained aspects to improve. The incremental exercises, the project and video lectures got positive feedback, as noted by a student in the official course feedback "*Project and exercises were successful, one could actually do something else than calculators (i.a. exercises with graphical user interface)*" and "*The video lectures were a great invention, especially in teaching programming. They made the schedule adjusting possible*". The inquiry about improvements provided a wish, that "*The open data services should be utilized more (XML and especially other formats). In overall this course is very good as a whole*". There was no negative feedback about using open data as a data source for the programs.

The negative feedback did not emphasize the course's aspect but mainly the pacing "*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*", which suggests that the parts that were considered simpler and not in the main focus were skimmed through too fast. The grading also received some feedback, that "*Currently the course grading measures the amount of time used*". This feedback can be seen in both positive and negative light, since Hirschmann [26] states, that repeating increases the learning results. In this context the feedback is negative, since a course grade should indicate how much the student learned and not how much student used time to learn the topics.

5. DISCUSSION

5.1 Research questions

In the beginning we set a research question *what are the existing recommendations to teach fundamentals of object-oriented programming and how they suited our course?* We argue that we got a vision of how to construct this kind of course. When evaluating the results, we concentrated more on the feedback

about if the course was meaningful and stimulating to students instead of the learning results.

Based on the success of the implemented course we can recommend, that flipped classroom method suits well to teach programming. Students felt comfortable when they were familiarizing theory before coming to classroom to test newly learned theories in practice. Also Java and JavaFX-based GUI programs were a working combination to teach object-oriented programming. Besides excellent student feedback, the responsible teacher was satisfied with the new teaching methods, learning outcomes and the overall setup, while there were few problems in pacing and course grading.

The use of open data has been limited in the university level and programming courses and we could not find much research about the topic. Students found the use of open data similar to any other dataset and it was not found as a hindrance and in the future students can be used to curate and validate existing data as well, providing a meaningful symbiosis as suggested in [5]. The benefit of open data is that students felt doing something useful – as stated in the previous section – especially with GUI programming where they created programs they use in their everyday life with real life data, and it increased their motivation. Thus, we can argue that open data is useful in teaching.

In a nutshell the new course focused on three new key points:

- Flipped classroom method
- Java and JavaFX -based programs
- Utilization of real-life open data sources

In our experience all key points were successful and can be recommended to be used when developing an object-oriented programming course. The course grading and pacing were local problems, which do not affect these recommendations.

5.2 Improvements for the next year

The biggest problem we faced during this first round with the new course was the grading. As the average grade increased from 2.5 to 4.3 and although students liked the course and did their tasks well, the grading allowed them to have higher grades than their skills deserved. In the next year the requirements for the higher grades can be increased.

Figure 4 illustrated how the number of students who started the course but did not get grade had increased during the last three years. The reasons behind this trend are not clear. Some students have reported to find out that they do not need the course for their curricula and they drop it, some just vanish in the middle. Those students who start the course should be motivated to finish it too.

By observing the course and student's struggles, the teacher was able to point out three suggestions for improvements. The first problem was the disconnection between exercises and readable theory material, students were not able to find extensive theory package about the week's exercise problem. This suggests that the manuals should be integrated more closely to the exercises and videos. Second improvement is to increase the number of UML exercises. The students did not get much practice with the diagrams and some of the object-oriented understanding is easier to learn through visualization [33]. The third improvement is to add mandatory weekly quizzes (suggested by [60], see also [9]) before the exercises to ensure that students have studied the theory. The quizzing increases the probability, that students use time to prepare with the material and the quizzes can also serve as the point of origin and adjustments for the discussion in-class [60].

5.3 Limitations and validation of the study

Scientific studies have always some limitations [31]. Meyer [41] list objectivity as the first element of validity and reliability. To gain objective results we utilized three authors with different views on the data so that any biased construct could be identified and removed. To achieve the second element, construct validity, we utilized data gathered from 5 years so that we can identify changes in trends, not only yearly fluctuation. The utilization of several researcher and data from several years also helped to tackle the problem of internal validity, the third element in the list of Mayer's. To generalize, one has to remember that these findings are only relevant in the domain of teaching of programming in the university level and in other areas they are only as recommendations. What comes to the Meyer's reliability we can argue that our study is in line with the previous research.

The major limitations in this study were the lack of control group, multiple simultaneous changes and the individual preferences of teacher and students. While the lack of control group is natural to case study method, it would have been necessary to enable a comparison between the used methods, the comparison between years is not as extensive. The multiple simultaneous changes in the course did not produce negative feedback but it is difficult to state, which changes affected the most. The individual preferences of the teacher and students cause the study to be biased; it cannot be said if the changes would have been as successful a year after or a year before nor can be stated that a different teacher would have gotten the same results [2].

Multiple research papers suggest, that flipped classroom method influence the learning results in the course positively. When compared to the traditional lecturing, students found the flipped classroom to be more supportive to their individual learning styles. The quantitative data also suggests that students found the lecturing method more satisfying and the level of understanding was better, as is indicated by the success rate in course project.

6. CONCLUSION

This article studied how a university level object-oriented programming course can be constructed. The literature suggested that flipped classroom teaching method is suitable when teaching programming and the experience gained in this study supports these findings. Students gave very positive feedback to the flipped classroom.

Besides teaching methods the programming language was changed and all the material, weekly exercises, programming project, and exam were redesigned. Although basically everything was changed the course did not meet any major problems – though there is always room for improvements and we are going to continue to improve the course during the next semester. It seems that for example quizzes before classroom exercises would improve the going through of material as now some students came to classroom without beforehand preparations.

The utilization of open data in the exercise tasks and programming project got also positive feedback and we aim to increase the amount of data sources used in various programming courses. This ties teaching tighter to real-life material and thus reduces the gap between educational and industrial environments.

During the next semester we are going to replicate the study with another course to gain more experience on how repeatable are the results.

Acknowledgements

We would like to thank Liikennevirasto and FUUG for supporting this study.

7. REFERENCES

[1] Baldwin, D. 2015. Can We "Flip" Non-Major Programming Courses Yet? *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2015), 563–568.

[2] Bartlett, R.L. 1996. Discovering diversity in introductory economics. *The Journal of Economic Perspectives*. (1996), 141–153.

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

[4] Bishop, J.L. and Verleger, M.A. 2013. The flipped classroom: A survey of the research. *ASEE National Conference Proceedings, Atlanta, GA* (2013).

[5] Bradley, J.-C., Lancashire, R.J., Lang, A.S. and Williams, A.J. 2009. The Spectral Game: leveraging Open Data and crowdsourcing for education. *Journal of Cheminformatics*. 1, 1 (2009), 9.

[6] Burton, P.J. and Bruhn, R.E. 2003. Teaching programming in the OOP era. *ACM SIGCSE Bulletin*. 35, 2 (Jun. 2003), 111.

[7] Capretz, L.F. 2003. A brief history of the object-oriented approach. *ACM SIGSOFT Software Engineering Notes*. 28, 2 (Mar. 2003), 6.

[8] Chen, W.-K. and Cheng, Y.C. 2007. Teaching Object-Oriented Programming Laboratory With Computer Game Programming. *IEEE Transactions on Education*. 50, 3 (Aug. 2007), 197–203.

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

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

[11] Culwin, F. 1999. Object imperatives! *SIGCSE Bull.* 31, 1 (1999), 31–36.

[12] Desouza, K.C. and Bhagwatwar, A. 2012. Citizen Apps to Solve Complex Urban Problems. *Journal of Urban Technology*. 19, 3 (Jul. 2012), 107–136.

[13] Dingle, A. and Zander, C. 2000. Assessing the ripple effect of CS1 language choice. *J. Comput. Sci. Coll.* 16, 2 (2000), 85–93.

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

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

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

[17] Fink, A. 2013. *How to conduct surveys: a step-by-step guide*. SAGE.

[18] Gable, G.G. 1994. Integrating case study and survey research methods: an example in information systems. *European Journal of Information Systems*. 3, (1994), 112–126.

[19] Gannod, G.C., Burge, J.E. and Helmick, M.T. 2008. Using the Inverted Classroom to Teach Software Engineering. *Proceedings of the 30th International Conference on Software Engineering* (New York, NY, USA, 2008), 777–786.

[20] Gartner Says Worldwide Video Game Market to Total $93 Billion in 2013: 2013. *http://www.gartner.com/newsroom/id/2614915*. Accessed: 2015-03-30.

[21] Gehringer, E.F. and Peddycord, B.W., III 2013. The Inverted-lecture Model: A Case Study in Computer Architecture. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), 489–494.

[22] Geser, G. and Salzburg Research Forschungsgesellschaft 2007. *Open educational practices and resources: OLCOS Roadmap 2012*. Salzburg Research.

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

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

[25] Hiltunen, K., Latva, S. and Kaleva, J.-P. 2013. *Peliteollisuus – kehityspolku*. TEKES.

[26] Hirschmann, W.B. 1964. Profit From the Learning Curve. *Harvard Business Review*. 42, 1 (1964), 125–139.

[27] Horton, D. and Craig, M. 2015. Drop, Fail, Pass, Continue: Persistence in CS1 and Beyond in Traditional and Inverted Delivery. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2015), 235–240.

[28] Howatt, J. 1995. A project-based approach to programming language evaluation. *ACM SIGPLAN Notices*. 30, 7 (Jul. 1995), 37–40.

[29] Jan Hylén 2006. *Open Educational Resources: Opportunities and Challenges*. Organisation for Economic Co-operation and Development.

[30] Kasurinen, J. and Nikula, U. 2007. Lower dropout rates and better grades through revised course infrastructure. *Proceedings of the 10th IASTED International Conference on Computers and Advanced Technology in Education* (Beijing, China, 2007), 152–157.

[31] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., Emam, K. El and Rosenberg, J. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*. 28, 8 (Aug. 2002), 721–734.

[32] Kölling, M. 1999. The problem of teaching object-oriented programming, Part I: Languages. *Journal of Object-Oriented Programming*. 11, 8 (1999), 8–15.

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

[34] Kölling, M. and Rosenberg, J. 2001. Guidelines for teaching object orientation with Java. *SIGCSE Bull.* 33, 3 (2001), 33–36.

[35] Lage, M.J., Platt, G.J. and Treglia, M. 2000. Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment. *The Journal of Economic Education*. 31, 1 (Jan. 2000), 30–43.

[36] Latulipe, C., Long, N.B. and Seminario, C.E. 2015. Structuring Flipped Classes with Lightweight Teams and Gamification. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2015), 392–397.

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

[38] Maher, M.L., Latulipe, C., Lipford, H. and Rorrer, A. 2015. Flipped Classroom Strategies for CS Education. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2015), 218–223.

[39] March, S.T. and Smith, G.F. 1995. Design and Natural Science Research on Information Technology. *Decision Support Systems*. 15, 4 (1995), 251–266.

[40] Mason, G.S., Shuman, T.R. and Cook, K.E. 2013. Comparing the Effectiveness of an Inverted Classroom to a Traditional Classroom in an Upper-Division Engineering Course. *IEEE Transactions on Education*. 56, 4 (Nov. 2013), 430–435.

[41] Meyer, C.B. 2001. A Case in Case Study Methodology. *Field Methods*. 13, 4 (Nov. 2001), 329–352.

[42] Moravec, M., Williams, A., Aguilar-Roca, N. and O'Dowd, D.K. 2010. Learn before Lecture: A Strategy That Improves Learning Outcomes in a Large Introductory Biology Class. *CBE-Life Sciences Education*. 9, 4 (Dec. 2010), 473–481.

[43] Open Definition - Open Definition - Defining Open in Open Data, Open Content and Open Knowledge: *http://opendefinition.org/od/index.html*. Accessed: 2015-04-09.

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

[45] Perkmann, M. and Schildt, H. 2015. Open data partnerships between firms and universities: The role of boundary organizations. *Research Policy*. 44, 5 (Jun. 2015), 1133–1143.

[46] Programming Language Popularity: 2014. *http://langpop.com/*. Accessed: 2014-08-27.

[47] Raadt, M. de and Toleman, M. 2002. Language Trends in Introductory Programming Courses. *Proc. Informing Science and IT Education Conference* (2002).

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

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

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

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

[52] Schullery, N.M., Reck, R.F. and Schullery, S.E. 2011. Toward solving the high enrollment, low engagement dilemma: A case study in introductory business. *International Journal of Business, Humanities and Technology*. 1, 2 (2011), 1–9.

[53] The Joint Task Force on Computing Curricula ed. 2001. Computing curricula 2001. *J. Educ. Resour. Comput.* 1, 3es (2001), 1.

[54] The Transparent Language Popularity Index: 2014. *http://lang-index.sourceforge.net/*. Accessed: 2014-08-27.

[55] Tietze, K.J. 2007. A Bingo Game Motivates Students to Interact with Course Material. *American Journal of Pharmaceutical Education*. 71, 4 (Aug. 2007), 79.

[56] TIOBE Software: TIOBE Index: 2014. *http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html*. Accessed: 2014-08-27.

[57] Toto, R. and Nguyen, H. 2009. Flipping the work design in an industrial engineering course. *Frontiers in Education Conference, 2009. FIE'09. 39th IEEE* (2009), 1–4.

[58] What makes data open? | Guides | Open Data Institute: *http://theodi.org/guides/what-open-data*. Accessed: 2014-12-10.

[59] Yin, R.K. 2002. *Case Study Research: Design and Methods*. SAGE Publications.

[60] Zappe, S., Leicht, R., Messner, J., Litzinger, T. and Lee, H.W. 2009. "Flipping" the classroom to explore active learning in a large undergraduate course. *American Society for Engineering Education* (2009).