# CURRENT STATE OF PROGRAMMING IN HIGH SCHOOLS AND UNIVERSITIES - HORIZONTAL ANALYSIS IN CROATIA

Lidija Kozina

Davor Fodrek

Zlatko Stapić

Ivanec, July 2024

Co-funded by the Erasmus+ Programme of the European Union

Authors

Lidija Kozina, High School Ivanec

Davor Fodrek, High School Ivanec

Zlatko Stapić, Faculty of Organization and Informatics Varaždin

Project

Object Oriented Programming for Fun

Project acronym

OOP4FUN

Agreement number

2021-1-SK01-KA220-SCH-00027903

Project coordinator

Žilinska univerzita v Žiline (Slovakia)

Project partners

Sveučilište u Zagrebu (Croatia)

Srednja škola Ivanec (Croatia)

Univerzita Pardubice (Czech Republic)

Gymnazium Pardubice (Czech Republic)

Obchodna akademia Povazska Bystrica (Slovakia)

Hochschule fuer Technik und Wirtschaft Dresden (Germany)

Gymnasium Dresden-Plauen (Germany)

Univerzitet u Beogradu (Serbia)

Gimnazija Ivanjica (Serbia)

Year of publication

July, 2024

# Table of contents

# List of tables

# List of charts

# 1. Horizontal analysis of universities' data

To analyze gap in teaching of (object oriented) programming between high schools and universities firstly we have done analysis of the way of teaching of programming on universities. All partners identified relevant subjects related to teaching OOP in the bachelor studies and we performed horizontal analysis of these data. The methodology used to collect and analyze data was as follows:

1. Partners from universities identified subjects related to teaching of OOP. For every subject we collected these data (collected data are enclosed in attachment):
    a. Type of subject (mandatory/optional).
    b. Year of study.
    c. Total hours.
    d. Hours of teaching OOP.
    e. Hours of teaching topics related to OOP.
    f. Prior knowledge required to attend the subject.
    g. Prior skills required to attend the subject.
    h. Learning outcomes.
    i. Topics.
    j. Description of teaching methods.
    k. Type of activities (investigation, discussion, practical work, production, data acquisition, etc.).
    l. Use of technology.
2. Review of the data was performed. Partners from universities reviewed and discussed data entered from other partners in order to resolve inequalities.
3. The analysis of data was performed. We divided it into four areas:
    a. Analysis of OOP load. We focused on the year of study and hour dotation of every subject and categorized the subjects. Relevant subjects were filtered out for the other stages of analysis.
    b. Analysis of prior knowledge and prior skills. We compared these prerequisites to the current teaching practice of OOP.
    c. Analysis of learning outcomes. We identified outcomes and competencies that could be moved to high schools' syllabus.
    d. Analysis of methodologies used to teach OOP in universities.

We will focus on data collected in Croatia, from a Faculty of organization and informatics, Varaždin and High School, Ivanec.

## 1.1. Analysis of OOP load

The very first analysis of collected data was focused on identification of relevant subjects for latter parts of this analysis. Every partner was obliged to analyze relevant courses. This populated set $S$ of subjects taken into consideration. In this part we focused on following data for every subject $\sigma \in S$:

a. Type of subject (mandatory/optional): $\mathrm{T}_{\sigma} \in \{M, O\}$
b. Year of study: $\mathrm{y}_{\sigma} \in \{1,2,3\}$

c. Total hours: $TH_\sigma$
d. Total hours of teaching OOP: $TH_\sigma^{teaching}$
e. Total hours of teaching topics related to OOP: $TH_\sigma^{related}$

In order to perform the analysis we computed:

f. Relative hours of teaching OOP: $RH_\sigma^{teaching} = \frac{TH_\sigma^{teaching}}{TH_\sigma}$
g. Relative hours of teaching topics related to OOP: $RH_\sigma^{related} = \frac{TH_\sigma^{related}}{TH_\sigma}$
h. Total hors of teaching topics unrelated to OOP:
$$TH_\sigma^{unrelated} = TH_\sigma - TH_\sigma^{teaching} - TH_\sigma^{related}$$
i. Relative hours of teaching topics unrelated to OOP: $RH_\sigma^{unrelated} = \frac{TH_\sigma^{unrelated}}{TH_\sigma}$

To identify relevant courses we proposed three categories:

1. **OOP teaching course**: Courses in this category are primarily focused on teaching new concepts. These courses are oriented both on theoretical knowledge as well as on practical skills in using of these concepts. The criterion for the course to fall into this category is to teach topics directly related to the OOP at least in half of the hours:
$$crit_\sigma^{teaching} = RH_\sigma^{teaching} \geq 0.5.$$
2. **OOP practicing course**: Courses in this category are primarily focused on practical understanding of OOP concepts. These courses are not focused on teaching theoretical background – typically they rely on knowledge previously learned in courses from OOP teaching category and teach brand new concepts in smaller number of hours. The focus is placed on understanding of practical usage of the OOP in different scenarios. The criterion for the course to fall into this category is not to be an OOP teaching course and to teach topics related to the OOP at least in third of the hours:
$$crit_\sigma^{practicing} = \neg crit_\sigma^{teaching} \wedge RH_\sigma^{teaching} + RH_\sigma^{related} \geq 0.3.$$
3. **OOP using course**: Courses in this category are not focused on teaching OOP however are strongly dependent on understanding of OOP. These are typically courses focused on some technology/programming language. If new OOP concepts are discussed, these are typically strongly specific for used technology/programming language and may not be applicable in other technologies/programming languages. The criterion for the course to fit in this category is not to fit in any of previous two categories:
$$crit_\sigma^{using} = \neg crit_\sigma^{practicing}.$$

Note that fitting any course into forementioned categories does not mean, that the course itself its curriculum does not belong or is in opposition to the other two categories. We point out that any OOP teaching course can fit OOP practicing (without proper practice it is not possible to cover new topics) as well as OOP using (one has to use specific language with its specifics to learn it) category, however this does not work vice versa (OOP using course is not OOP teaching). In the following picture we define the hierarchy of the categories based on the areas that must be covered in respective lectures.

Categorization of all courses is presented in Table 1.

*Table 1 - University courses categorization*

| University | Subject name | Type of subject | Year | Hours | | | | | Category |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Total | Teaching OOP | | Related to OOP | | |
| | | | | | Total | Relative | Total | Relative | |
| FOI | Object-oriented programming | Mandatory | 1 | 60 | 60 | 100% | 0 | 0% | OOP Teaching |
| FOI | Windows Applications Development | Optional | 2 | 60 | 10 | 17% | 24 | 40% | OOP Practising |
| FOI | Programming in Python | Optional | 3 | 30 | 6 | 20% | 2 | 7% | OOP Using |
| FOI | Programming 2 | Mandatory | 1 | 60 | 20 | 33% | 10 | 17% | OOP Practising |
| FOI | Mobile applications and games development | Mandatory | 3 | 60 | 0 | 0% | 43 | 72% | OOP Practising |

Despite the necessity of courses from category OOP using on universities, we decided to filter them out and do the analysis taking into consideration only OOP teaching and OOP practicing courses, since these are focused on teaching the OOP concepts.

In order to identify the load of OOP in respective years, firstly the separate analysis of respective universities' courses was performed. If the load of the OOP will be high in the first years, that will support the assumption that universities invest a big amount of teaching hours to teach OOP from the beginning of studies. In the following charts and tables, we show data processed of every university as follows:

1. For every year of study, we present:
    a. Tables showing both total (TH) and related (RH) hours of teaching OOP, related to OOP and non related to OOP for both **course types** (mandatory and optional). These tables provide a data for following chart.
    b. Charts of RH of teaching OOP, related to OOP and non related to OOP. One can see the distribution of hours between subjects focused on OOP. One can see that hierarchy of courses is copied – in the first year of studies, the courses from the bottom of the hierarchy are present, latter the top courses are present.
    c. Tables showing both total (TH) and related (RH) hours of teaching OOP, related to OOP and non related to OOP for defined **course categories** (OOP teaching, OOP practicing and OOP using).
2. In the overall analysis of all years of study, we present:
    a. Chart of TH (teaching, related and unrelated) in every year of study (see sum row of respective tables as described in 1a). One can see total number of hours devoted to subjects focused on OOP. This justify the assumption about the focus of universities on OOP in first years of study.
    b. Chart of $TH_{\sigma}^{teaching}$ in every year of study distributed among course categories (see OOP teaching row of respective tables as described in 1c). One can observe the focus of OOP teaching subjects in different years of study. Note the significant number of OOP teaching courses in the first year of study.

The informatics related study program at FOI places courses that are relevant to OOP in all three years of study. Tables Table 2 and Table 3 summarize 1st year of study, Table 4 and

Table 5 summarize 2nd year of study Table 6 and Table 7 summarize 3rd year of study. Visualization of relative distributions of hours in respective years of study are presented in Chart 11, Chart 22 and Chart 33. Overall analysis of relevant FOI courses is presented in charts Chart 44 and Chart 55.

1st year of study

*Table 2 - Total OOP teaching/related/unrelated hours per mandatory and optinal subjects of 1st year of study on FOI*

| | Hours per year 1 | | | | | | |
|---|---|---|---|---|---|---|---|
| **Course type** | Σ Course total | Teaching OOP Σ | Teaching OOP % | Related to OOP Σ | Related to OOP % | Unrelated to OOP Σ | Unrelated to OOP % |
| **Mandatory** | 120 | 80 | 66,66667 | 10 | 8,333333 | 30 | 25 |
| **Optional** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Σ** | 120 | 80 | | 10 | | 30 | |

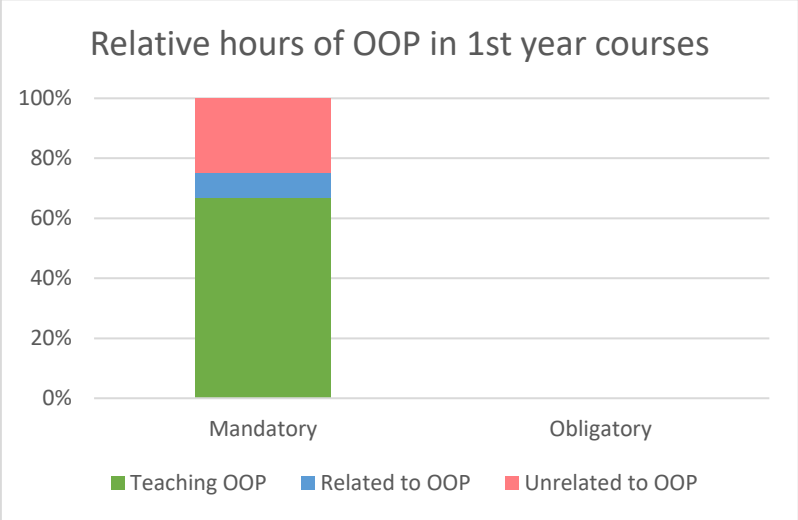


*Chart 1 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on FOI*

*Table 3 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 1st year of study on FOI*

| | **Course category** | Σ Course total | Teaching OOP Σ | Teaching OOP % | Related to OOP Σ | Related to OOP % | Unrelated to OOP Σ | Unrelated to OOP % |
|---|---|---|---|---|---|---|---|---|
| **OOP Teaching** | **OOP Teaching** | 60 | 60 | 100 | 0 | 0 | 0 | 0 |
| **OOP Practising** | **OOP Practising** | 60 | 20 | 33,33333 | 10 | 16,66667 | 30 | 50 |
| **OOP Using** | **OOP Using** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **Σ** | 120 | 80 | | 10 | | 30 | |

2nd year of study

*Table 4 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on FOI*

| | | Hours per year 2 | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Course type** | | Σ Course total | Teaching OOP | | Related to OOP | | Unrelated to OOP | |
| | | | Σ | % | Σ | % | Σ | % |
| **Mandatory** | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Optional** | | 60 | 10 | 16,66667 | 24 | 40 | 26 | 43,33333 |
| **Σ** | | 60 | 10 | | 24 | | 26 | |

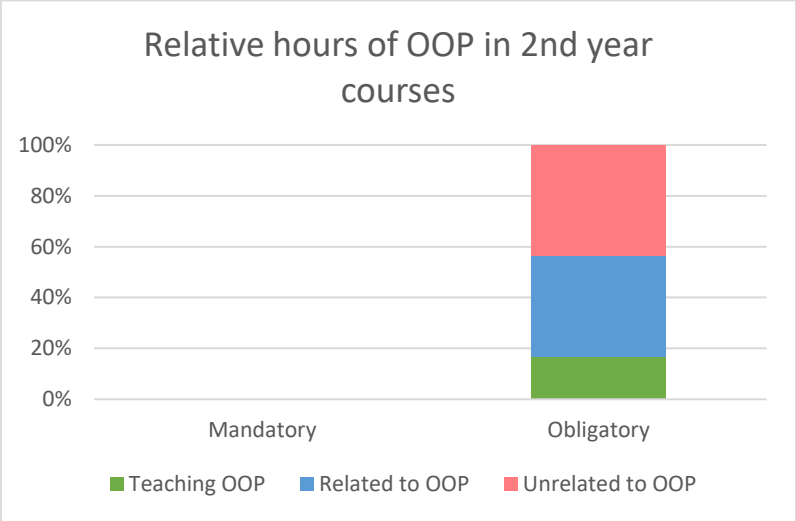


*Chart 2 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on FOI*

*Table 5 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on FOI*

| | Hours per year 2 | | | | | | |
|---|---|---|---|---|---|---|---|
| **Course category** | Σ Course total | Teaching OOP | | Related to OOP | | Unrelated to OOP | |
| | | Σ | % | Σ | % | Σ | % |
| **OOP Teaching** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **OOP Practising** | 60 | 10 | 16,66667 | 24 | 40 | 26 | 43,33333 |
| **OOP Using** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Σ** | 60 | 10 | | 24 | | 26 | |

3rd year of study

*Table 6 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 3rd year of study on FOI*

| | Hours per year 3 | | | | | | |
|---|---|---|---|---|---|---|---|
| **Course type** | Σ Course total | Teaching OOP | | Related to OOP | | Unrelated to OOP | |
| | | Σ | % | Σ | % | Σ | % |
| **Mandatory** | 60 | 0 | 0 | 43 | 71,66667 | 17 | 28,33333 |
| **Optional** | 30 | 6 | 20 | 2 | 6,666667 | 22 | 73,33333 |
| **Σ** | 90 | 6 | | 45 | | 39 | |

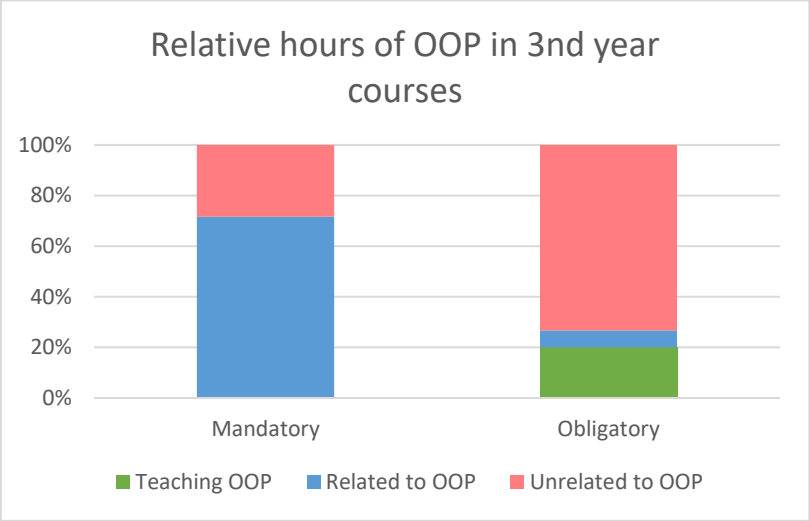


*Chart 3 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 3rd year of study on FOI*

*Table 7 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 3rd year of study on FOI*

| | Hours per year 3 | | | | | | |
|---|---|---|---|---|---|---|---|
| **Course category** | Σ Course total | Teaching OOP | | Related to OOP | | Unrelated to OOP | |
| | | Σ | % | Σ | % | Σ | % |
| **OOP Teaching** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **OOP Practising** | 60 | 0 | 0 | 43 | 71,66667 | 17 | 28,33333 |
| **OOP Using** | 30 | 6 | 20 | 2 | 6,666667 | 22 | 73,33333 |
| **Σ** | 90 | 6 | | 45 | | 39 | |

Overall

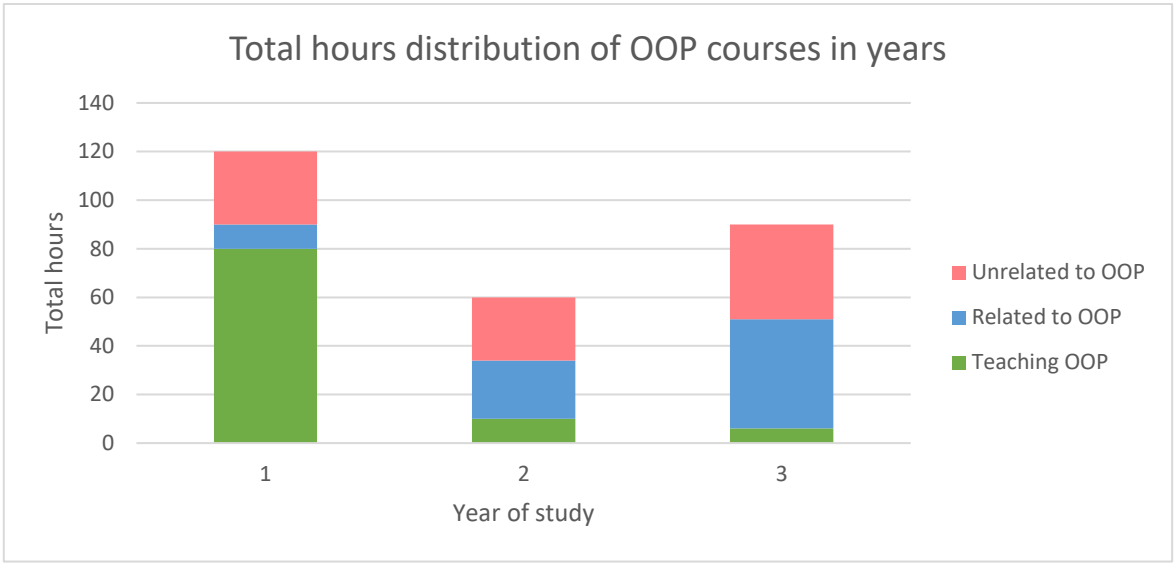


*Chart 4 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on FOI*

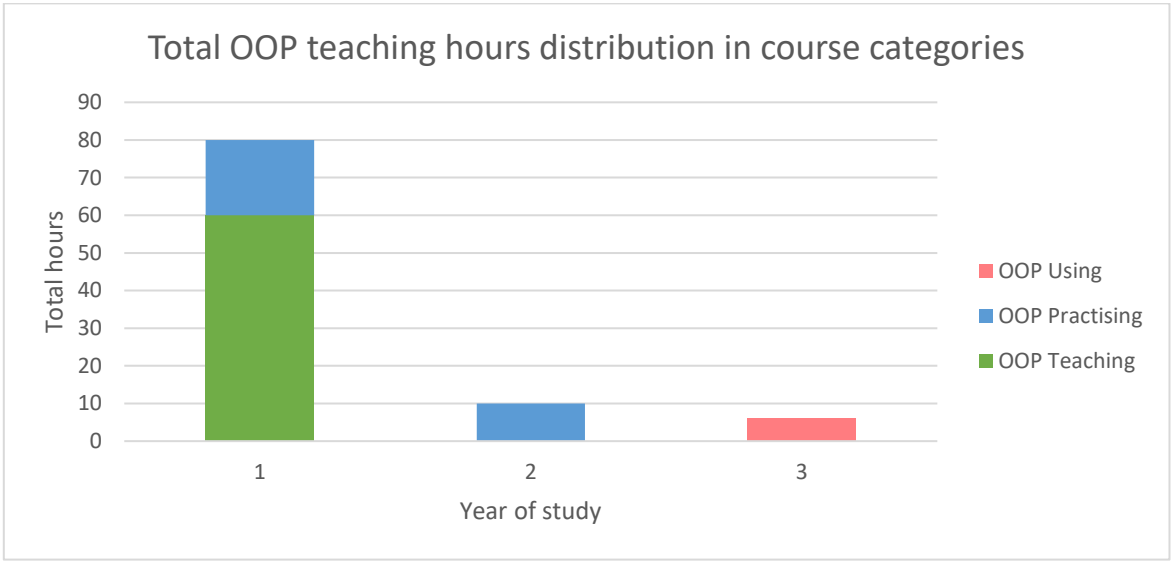


*Chart 5 - Total OOP teaching/practicing/using hours distribution between 3 years of study on FOI of courses teaching OOP*

## 1.2. Analysis of prior requirements of universities' OOP related courses

The gap between universities and high schools could lie in the different expectations of skills and knowledge of the universities and the real skills and knowledge of absolvents of high schools. In order to investigate this question, we performed analysis of prior skills and prior knowledge. The methodology was as follows:

1. We divided courses to OOP teaching, OOP practicing and OOP using.
2. From every subject we identified areas of prior requirements based on prior knowledge and prior skills provided by partners.
3. To identify overlaps between partners we created matrices of areas of prior requirements of subjects.
4. Based on experience and related work we interpreted data and formulated conclusion related to our project.

The **areas of prior requirements** were identified as follows.

- *None* (there is no specific prior knowledge or skill requested).
- *Code comprehension* (student has ability to understand code written in a programming language).
- *Algorithmization* (student can write an algorithm based on description of some process).
- *Structural programming* (student can write structured code using basic control structures).
- *Object programming* (student can write code following basic principles of OOP – objects, composition, association). This requirement is listed as well, since some courses from the analysis build upon knowledge of other courses, where such topic is covered. For the sake of consistency, we present the analysis of all university courses, however we will exclude such courses in conclusion.
- *Sophisticated programming* (student can write sophisticated code using proper paradigm in order to solve non-trivial problems).
- *Data structures* (student understands the philosophy and usage of fundamental data structures).
- *Mathematics HS* (student can solve math problems of the high school level).
- *Mathematics UNI* (student can solve math problems of the university level).
- *Programming language* (student can write algorithms in specified programming language).
- *UML* (student can create various UML diagrams with proper usage in particular situation).
- *Software architecture* (student is capable to create proper design of software following principles of selected architecture).
- *Computer networks* (student understands the principles of computer networks).
- *Use of IDE* (student knows how to use IDE and respective tools such as compiler, debugger, code editor).
- *Use of PC* (student knows how to install application, browse web, (un)pack files, work with office applications).

Areas of prior requirements for every subject are listed in separate tables with structure presented below. Tables include also relevant data (collected data are enclosed in attachment) without further content modifications, so the identification of respective areas is clear. For better orientation we highlighted relevant row. Every table contains:

- Subject name.

- Type of subject (mandatory/optional).
- Year of study.
- Prior knowledge required to attend the subject.
- Learning outcomes of related course - if analyzed course requests knowledge of some other subject, we put the learning outcome of that other subject. If there is no prerequisite, this row is not included in the table.
- Prior skills required to attend the subject.
- **Areas of prior requirements** – List of requirements of analyzed subject. These are derived from required prior knowledge, from learning outcomes of related course and from required prior skills.

### 1.2.1. OOP teaching courses

Following table summarizes OOP teaching courses as defined in the OOP load analysis on FOI.

*Table 8 - OOP teaching courses considered in the prior requirements analysis*

| University | Subject name | Type of subject | Year |
|---|---|---|---|
| FOI | Object-oriented programming | Mandatory | 1 |

On FOI, there is one mandatory OOP teaching course in the first year of study analyzed.

*Table 9 - Prior requirements analysis of course Object-oriented programming on FOI*

| **Subject name** | **Object-oriented programming** |
|---|---|
| **Type of subject** | Mandatory |
| **Year** | 1 |
| **Prior knowledge** | Algorithmic problem-solving, basics of structural programming, simple and complex data structures, control structures (sequence, selection, iteration, jump statements), functions and procedures. |
| **Prior skills** | The course requires basic skills in writing and understanding procedural code and the use of integrated development environments. |
| **Areas of prior requirements** | Algorithmization<br>Structural programming<br>Data structures<br>Code comprehension<br>Use of IDE |

### 1.2.2. OOP practicing courses

Following table summarizes OOP practicing courses as defined in the OOP load analysis on FOI.

*Table 10 - OOP practicing courses considered in the prior requirements analysis*

| **University** | **Subject name** | **Type of subject** | **Year** |
|---|---|---|---|
| FOI | Windows Applications Development | Optional | 2 |
| FOI | Programming 2 | Mandatory | 1 |
| FOI | Mobile applications and games development | Mandatory | 3 |

On FOI, there are three OOP practicing courses analyzed: Mandatory in first year of study, optional in second year of study and mandatory in third year of study.

*Table 11 - Prior requirements analysis of course Windows Applications Development on FOI*

| **Subject name** | **Windows Applications Development** |
|---|---|
| **Type of subject** | Optional |
| **Year** | 2 |
| **Prior knowledge** | The prerequisites include the successful completion of Object-oriented programming course. |
| Learning outcome of *Object-oriented programming* | After the course student is able to:<br>- Design an efficient software solution for a given algorithmic problem<br>- Create models of the software solution using standard UML diagrams<br>- Organize date in an efficient way for a given algorithmic problem.<br>- Create models of data structures using standard UML diagrams<br>- Develop a software solution for a given algorithmic problem using object-oriented programming language. |
| **Prior skills** | The course requires basic skills in writing and understanding procedural and object-oriented code and the use of integrated development environments. |
| **Areas of prior requirements** | Algorithmization<br>Object programming<br>UML<br>Code comprehension<br>Use of IDE |

*Table 12 - Prior requirements analysis of course Programming 2on FOI*

| **Subject name** | **Programming 2** |
|---|---|
| **Type of subject** | Mandatory |
| **Year** | 1 |
| **Prior knowledge** | Algorithmic problem-solving, basics of structural programming, simple and complex data structures, control structures (sequence, selection, iteration, jump statements), functions and procedures. |
| **Prior skills** | The course requires basic skills in writing and understanding procedural code and the use of integrated development environments. |
| **Areas of prior requirements** | Algorithmization<br>Structural programming<br>Data structures<br>Use of IDE |

*Table 13 - Prior requirements analysis of course Mobile applications and games development on FOI*

| **Subject name** | **Mobile applications and games development** |
|---|---|
| **Type of subject** | Mandatory |
| **Year** | 3 |
| **Prior knowledge** | The prerequisites include the successful completion of Object-oriented programming course and Mathematics 1 course. |
| Learning outcome of *Object-oriented programming* | After the course student is able to:<br>- Design an efficient software solution for a given algorithmic problem<br>- Create models of the software solution using standard UML diagrams<br>- Organize date in an efficient way for a given algorithmic problem.<br>- Create models of data structures using standard UML diagrams<br>- Develop a software solution for a given algorithmic problem using object-oriented programming language. |
| **Prior skills** | No prior skills are defined in course curriculum. However, one can conclude that the course requires basic skills in writing and understanding procedural code and the use of integrated development environments as these topics are not tought but are the base for the other topics included. |
| **Areas of prior requirements** | Algorithmization<br>Structural programming<br>Data structures<br>Mathematics UNI<br>Code comprehension<br>Use of IDE |

### 1.2.3. OOP using courses

Following table summarizes OOP using courses as defined int the OOP load analysis.

*Table 14 - OOP using courses considered in the prior requirements analysis*

| University | Subject name | Type of subject | Year |
|---|---|---|---|
| FOI | Programming in Python | Optional | 3 |

On FOI, there is one optional OOP using course in the third year of study analyzed.

*Table 15 - Prior requirements analysis of course Programming in Python on FOI*

| **Subject name** | **Programming in Python** |
|---|---|
| **Type of subject** | Optional |
| **Year** | 3 |
| **Prior knowledge** | The prerequisites include the successful completion of Object-oriented programming course. |
| Learning outcome of *Object-oriented programming* | After the course student is able to:<br>- Design an efficient software solution for a given algorithmic problem<br>- Create models of the software solution using standard UML diagrams<br>- Organize date in an efficient way for a given algorithmic problem.<br>- Create models of data structures using standard UML diagrams<br>- Develop a software solution for a given algorithmic problem using object-oriented programming language. |
| **Prior skills** | The course requires basic skills in writing and understanding procedural and object-oriented code and the use of integrated development environments. |
| **Areas of prior requirements** | Algorithmization<br>Object programming<br>UML<br>Code comprehension<br>Use of IDE |

## 1.3. Analysis of approaches for teaching OOP

### 1.3.1. Remarks on gathered data

In order to investigate practices in universities with regard to teaching object-oriented programming (OOP), we gathered data from relevant courses taught at project partner institutions. These high education institutions come from 5 countries which differ in terms of high education strategy and tradition. Also, some partner institutions are from technical fields, while others have strong social and business component. Finally, the courses themselves differ with regard to academic year they are taught, the content, and the teachers involved. This allowed us to gather sufficiently diverse data to identify wide range of practices, tools and methods used for teaching OOP.

### 1.3.2. Identified themes

#### *1.3.2.1. Forms of instruction / forms of knowledge transfer*

*Lectures*

With regard to forms of instruction all of the 21 analyzed courses favor lectures as a primary means to transfer theoretical knowledge to students. In order to do that, teachers stick with tradition and start by explaining underlying theoretical concepts (in 100% of analyzed courses). However, in 57% of courses teachers provide code examples and use scenarios to motivate students and put course content into real-life context. In majority of courses there is an emphasis on a two-way communication between lecturer and students, and between students themselves. Students are regularly encouraged to ask questions and provide feedback (81% of courses), and in some way to even get involved into discussions (14% of courses). In one of the courses interactivity and using practical examples is particularly emphasized by teacher writing programming code and students discuss it in real-time.

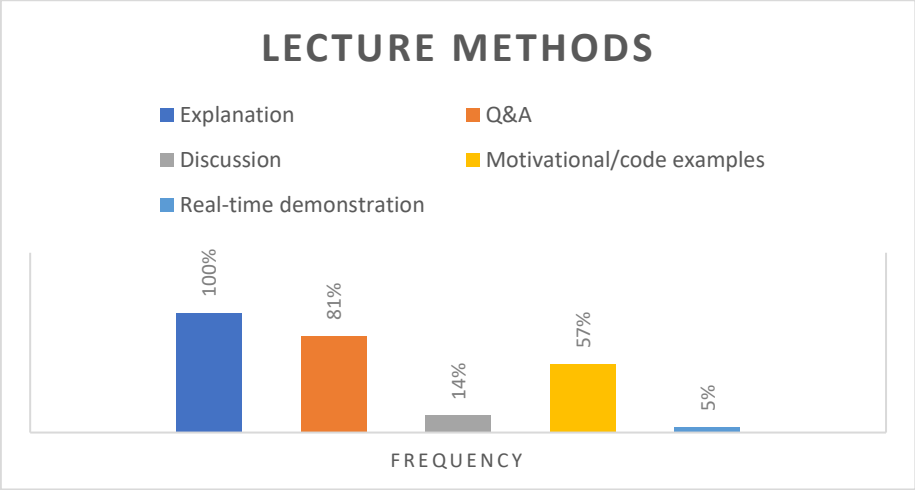


*Chart 6 - Forms of knowledge transfer used in lectures*

*Seminars/Laboratory exercises*

Transferring practical knowledge is seen as an essential activity in all the analyzed courses. Laboratory exercises involving students working on a computer were recognized as the most suitable form of instruction for doing that. Although learning-by-doing is the core approach in laboratory exercises of all courses, this was conducted in two distinct ways: (1) "Teacher-first" - teacher goes through an illustrative example together with students, and then students work on their own with occasional help and guide from the teacher (67% of courses), (2) "Student-first" students immediately start working on their own, but teacher provides intensive assistance (33% of courses). Students' activities and efforts in laboratory exercises are in some courses a prerequisite for taking exam, while in others are graded and are constituent part of course evaluation.

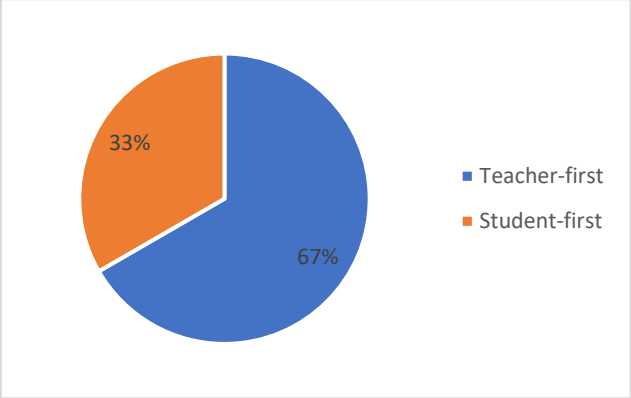


*Chart 7 - Learning-by-doing approach in laboratory exercises*

### *1.3.2.2. Individual work*

*Practical assignments*

In 33% of analyzed courses no home practical assignments were given to students during the semester. In such cases, courses rely solely on laboratory exercises to transfer practical knowledge, and traditional exams to evaluate it. However, most courses (67% of courses) require students to do some practical work at home. Such practical assignments expect students to gain and demonstrate general skills such as problem analysis and problem solving, as well as specific skills related to application of OOP principles and particular technology. The courses, however, differ in terms of when these assignments are done, as well as the size and the number of assignments. In 2 out of 14 courses with home assignments student start working on assignments during laboratory exercises (with teachers providing continuous feedback) and finish them at home. In the rest of courses (12 out of 14) practical assignments are entirely done at home, and only submitted for evaluation at specific points during the semester or at the end of the semester.

The size and the number of practical assignments in analyzed courses are correlated and depend on the overall course load. In some courses students are required to submit one larger-scale practical assignment (e.g. semestral project) which encompasses all relevant topics taught at the course. Other courses prescribe several smaller-scale practical assignments, with each assignment targeting particular topic (e.g. weekly topic), or a phase in software development process (e.g. problem analysis phase, design, implementation, documentation…). In some cases, these smaller-scale assignments are

linked to each other. For example, assignment covering design phase acts as an input model for an assignment covering implementation phase.

Most courses (11 out of 14) see practical assignments as an individual, one-student activity, while only 3 courses either allow or even mandate working in teams in a traditional or agile manner. In addition to teacher's feedback and evaluation, students are often required to present their solutions in front of classmates and also receive their feedback as well.

#### *1.3.2.3. Assessment*

Most analyzed courses (15 out of 21) assess theoretical knowledge of students either by only written exam (4 courses), only oral exam (3 courses) or both written and oral exam (8 courses). Other courses rely solely on practical assignments to demonstrate that students not only acquired theoretical knowledge but were also able to apply it to practical problems.

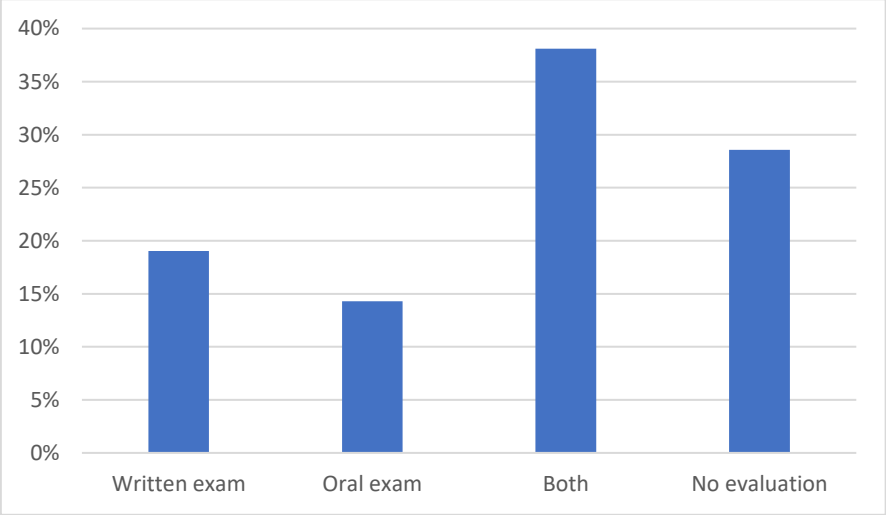


*Chart 8 - Frequency of assessment methods for theoretical knowledge*

While grasping theoretical concepts is always important, acquiring practical skills in analyzed courses was an imperative as 19 out of 21 courses had a formal assessment of practical skills. These assessments included assessing student's efforts on laboratory exercises during semester, evaluating practical assignments (tasks, projects) done at home during the semester, and performing practical parts of exam.

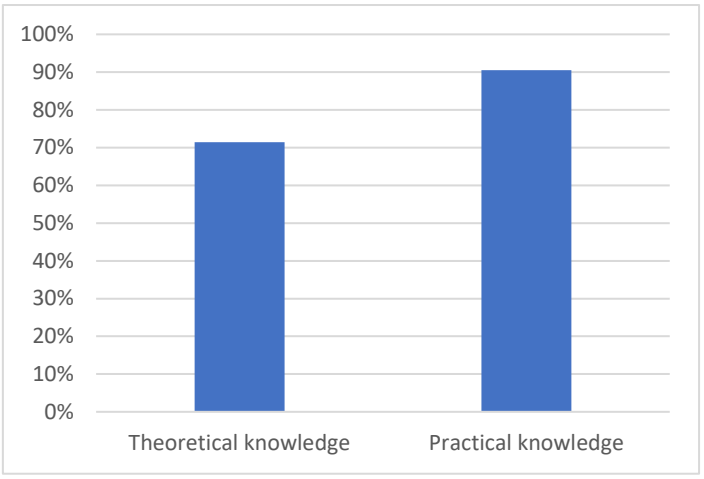


*Chart 9 - Type of knowledge assessed in analyzed courses*

### *1.3.2.4. Tools*

While used technologies vary across analyzed courses, it is not surprising that mainstream object-oriented languages dominate. For example, Java is used in 9 courses, C++ is used in 6 courses, and C# is used in 3 courses. Other mainstream languages include Python, Kotlin and Swift, each of them being used in only 1 course. An interesting technology (albeit used in only 1 introductory course) is RAPTOR - a flowchart-based programming environment which allows visual programming. Such technology can be used to demonstrate object-oriented concepts and mechanisms in a visual and less abstract way.

While most courses were mandating one “official” programming language to be used throughout the course, three courses were more liberal, and allowed students to choose their own preferred programming language and environment.

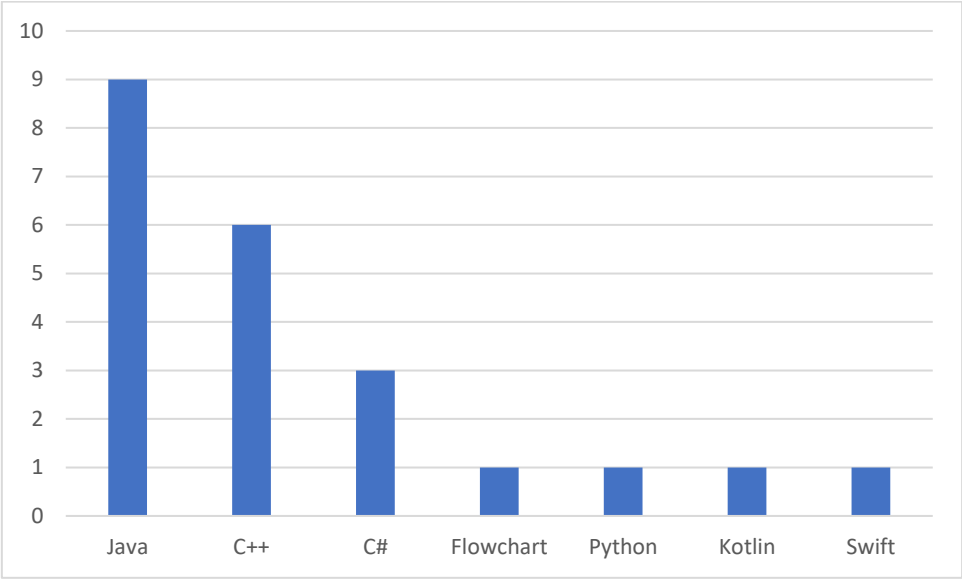


*Chart 10 - Programming language occurence in analyzed courses*

Popularity of integrated development environments (IDE) heavily depend on what programming language is used. Courses favoring Java programming language have most diverse IDE offering, and suggest using NetBeans (4 courses), IntelliJ IDEA (3 courses), Eclipse (2 courses) and BlueJ (2 courses). As we can see, most courses went with mainstream IDEs that are used for real-life Java development. However, from our perspective, a notable mention is also BlueJ IDE due to its support for teaching and learning OOP. Microsoft Visual Studio was a first choice in courses using C++ (4 courses) and C# programming languages (3 courses). This made Visual Studio the most represented IDE in total, and also the only IDE used for more than one language. In addition to Visual Studio, C++ development is also done in Dev-C++ (generally recommended for beginner programmers) and Verifikator (proprietary IDE developed by teachers to enforce good coding practices). Other popular IDEs such as PyCharm (Python), Android Studio (Kotlin) and Xcode (Swift) appeared each only in 1 course. Finally, the 3 courses which allowed students to choose programming language on their own, allowed students to also choose their preferred IDE.

*Table 16 - Suggested or mandated IDEs per programming language*

| Language | IDE 1 | IDE 2 | IDE 3 | IDE 4 |
|---|---|---|---|---|
| Java | Netbeans (4) | IntelliJ IDEA (3) | Eclipse (2) | BlueJ (2) |
| C++ | Visual Studio (4) | Dev-C++ (1) | Verifikator (1) | - |
| C# | Visual Studio (3) | - | - | - |
| Python | PyCharm (1) | - | - | - |
| Kotlin | Android Studio (1) | - | - | - |
| Swift | Xcode (1) | - | - | - |
| Flowchart | RAPTOR (1) | - | - | - |

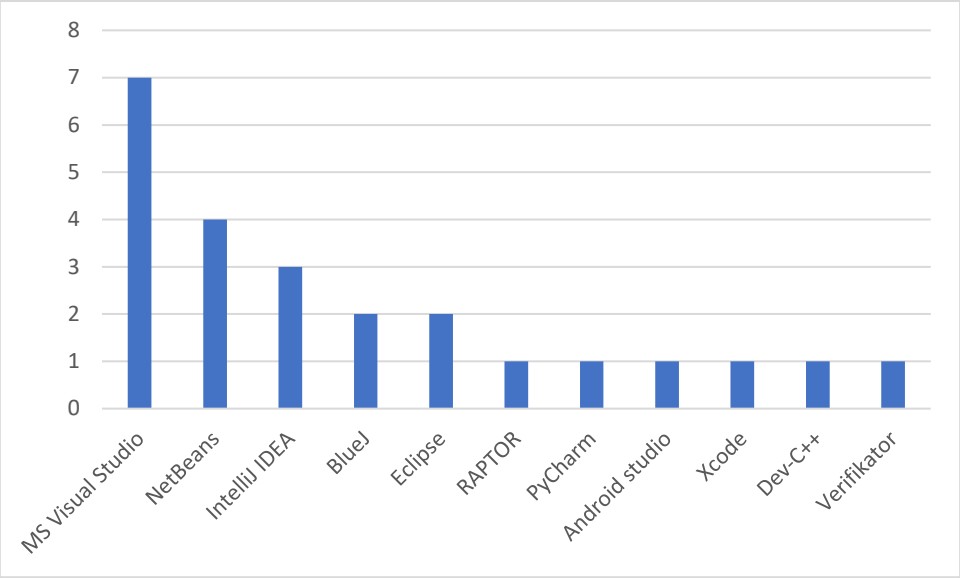


*Chart 11 - Most frequently used IDEs in analyzed courses*

### 1.3.3. Conclusion

While approaches for teaching OOP varied in analyzed courses, there are still some general trends that can be noticed across most of the courses. In terms of transferring theoretical knowledge, analyzed courses favor traditional lectures coupled with practical examples and two-way teacher-student interaction. Transferring practical knowledge is considered essential in all courses and is carried out as a combination of laboratory exercises and individual programming projects students do at home. Most courses have formal evaluation of theoretical knowledge either through written exam, oral exam, or both (most frequent case). Evaluation of practical knowledge is considered even more important in analyzed courses. Students demonstrate practical knowledge by working continuously on laboratory exercises, by submitting tasks and projects during the semester, or by taking practical tests at the end of semester.

Finally, in analyzed courses we can identify a number of different technologies, programming languages and environments. Most courses use mainstream OOP languages (Java, C#, C++) and their respective IDEs (Netbeans, IntelliJ IDEA, Eclipse, Visual Studio), which ensures students are acquainted with tools they are likely to use in real-life software development. Some courses however take a more lightweight approach and favor tools that are more suitable for teaching and learning OOP (RAPTOR, BlueJ, Verifikator).

# 2. Horizontal analysis of high schools' data

In order to make successful and usable gap identification in teaching of object oriented programming between high schools and universities, the next step was to make a horizontal analysis of high schools' data. It was done in a similar way as horizontal analysis of universities' data. The methodology used to collect and analyze data was the same, with some minor changes in scope of analyzed data. It was consisting of these steps:

1. All partners (from high schools) were analyzing their curricula of the subjects that are related to teaching object oriented programming and for every subject these data were collected:
    a. Subject name
    b. Type of subject (compulsory or optional)
    c. Grade/class (in which subject is taught)
    d. Hours of teaching OOP
    e. Learning outcomes
    f. Topics
    g. Description of teaching methods
    h. Type of activities (investigation, discussion, practical work, production, data aquisition, …)
    i. Assessment
    j. Teamwork experience
    k. Literature
    l. Suggestions on what (and how) should be improved in curriculum and/or in teaching OOP in schools
    m. Additional comments
    n. Additional subjects related to programming in general
2. All partners were making a review of data. Each high school partner performed a review of data from all other high school partners and each university partner made a review of a data from a high school in the same country. In this way, the consistency of the data was ensured, as well as the equalization of the way in which the data were collected. After the review phase was over, all the partners analyzed the comments and made necessary changes in data collections.
3. The analysis of data is performed. It was divided into five areas:
    a. Analysis of OOP load. It was focused on type of schools, type of subjects, subject names, grades/classes and hours of teaching OOP.
    b. Analysis of learning outcomes and topics
    c. Analysis of teaching methods, types of activities, assessments and team work experiences
    d. Analysis of literature and teaching materials
    e. Analysis of suggestions on how to improve OOP teaching in schools, additional comments from partners together with review of additional subjects related to programming in general (not object oriented programming)

## 2.1. Analysis of OOP load

There were three types of schools involved in this analysis (schools that are partners in this project):

1. schools that provide general education (gymnasiums),
2. schools that provide vocational education (different types of vocational programmes),
3. schools with both general and vocational education (gymnasium and vocational programmes).

Each partner in the project was obligated to review their own curricula and identify all the subjects in which OOP is present. That means that all the subjects where OOP is taught even on marginal level were taken into consideration. After the partners finished the data gathering process, a total of eight subjects were identified as subjects with OOP content. Number of relevant subjects per school can be observed in Chart 12.

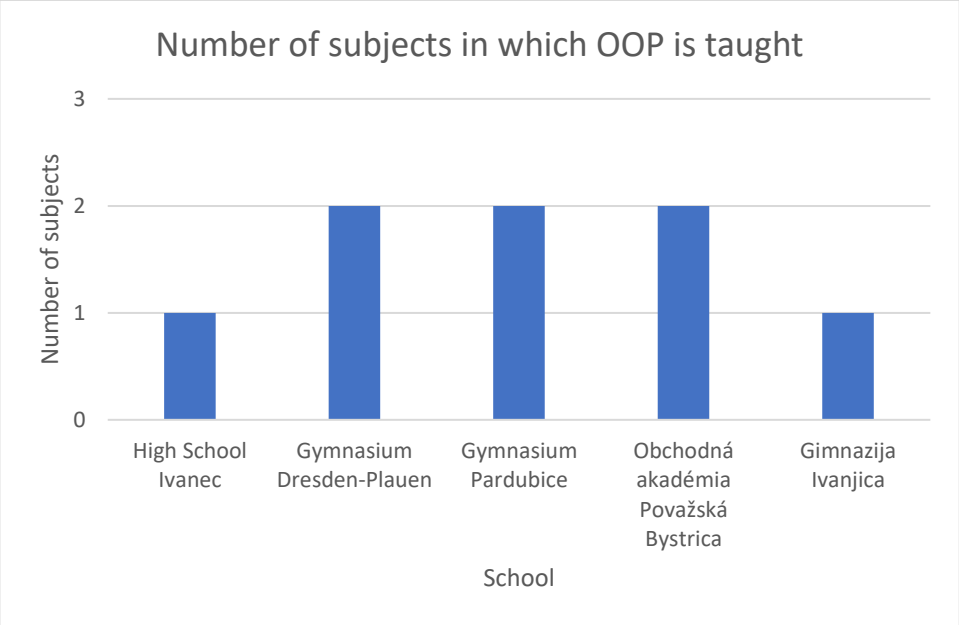


*Chart 12 - High schools and numbers of OOP related subjects*

As shown in the chart, all the schools have only one or two OOP related subjects. During analysis, no big difference was observed in the number of subjects related to OOP between general and vocational schools. However, there is a very big difference in number of subjects in which programming content in general (without OOP) is taught but that will be mentioned in more details later.

Regarding grades (or classes) in which OOP is taught, it wasn't possible to make accurate analysis and correlation because of differences in educational systems in different countries (differences in number of years and grades which are considered as primary and secondary education). However, it is obvious that none of the school is performing OOP teaching in lower grades. In 100% of cases, it is performed in higher grades which means 3rd or 4th year of secondary education (which can be compared to 11th or 12th grade for schools, for example, in Germany). This can be interpreted in a way that students still need to gain some knowledge in other programming areas (basics concepts of programming,

algorithms, data types, problem solving etc.) before they can successfully adopt the concepts of OOP. Data regarding institutions, subject names and types, number of hours and grades can be observed in Table 17.

*Table 17 - Basic data of OOP related subjects*

| High school | Subject name | Grade | Type of subject | Number of hours (teaching OOP) |
|---|---|---|---|---|
| High School Ivanec | Mobile application development | 4 | Optional | 15 |
| Gymnasium Dresden-Plauen | Practical computer science - Advanced programming | 11 or 12 | Compulsory and optional | 14 |
| | Data structure and modularization | 11 or 12 | Compulsory | 10 |
| Gymnasium Pardubice | Seminare of programming 1 | 3 | Optional | 60 |
| | Seminare of programming 2 | 4 | Optional | 45 |
| Obchodná akadémia Považská Bystrica | Applied Informatics - Seminar | 3 | Compulsory | 10 |
| | Applied Informatics - Seminar | 4 | Compulsory | 46 |
| Gimnazija Ivanjica | Object oriented programming | 3 | Compulsory | 148 |

As we can see, object oriented programming is taught in high schools in both compulsory and optional subjects. Although there aren't any big differences noted in learning outcomes or teaching methods between those two types of subjects, there is a difference for students which are later enrolled in OOP courses in universities. Compulsory subject means that it is obligatory to all students, which leads to conclusion that all high school students gain same knowledge and skills. On the other hand, optional subjects are chosen only by students who really want to attend those classes, by their own choice. After they finish their high school education (for example, in one vocational program), not all the students have the same OOP knowledge if the OOP subject was implemented in school as optional subject. The number of subjects divided into compulsory and optional categories is shown in Chart 13.

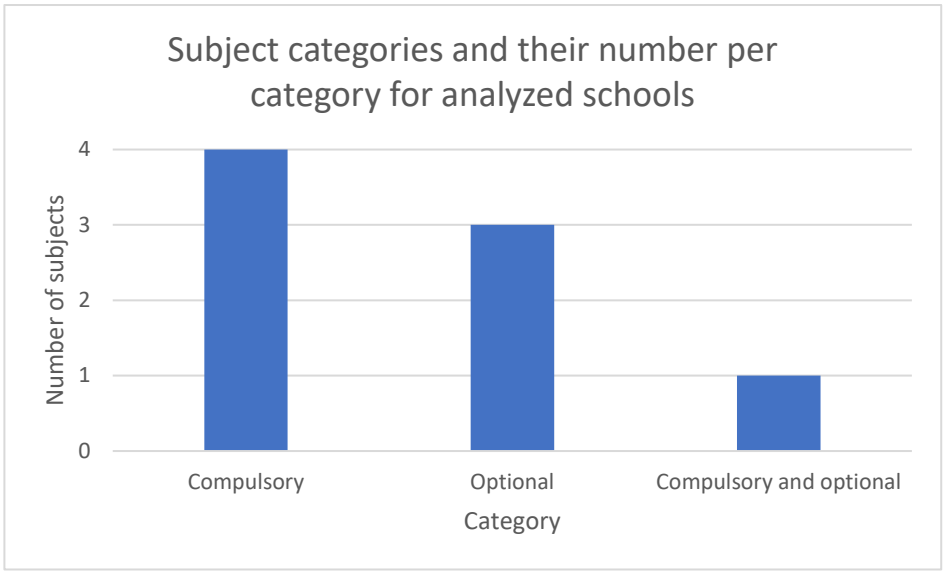


*Chart 13 - Categorization of subjects and their number per category*

As mentioned earlier, a total of eight subjects were taken into consideration for this analysis and their distribution by category looks like this: there are four compulsory subjects, three optional subjects and there is one subject which can be categorized as both compulsory and optional because it is one of four optional topics in the curriculum. Teachers can decide not to implement OOP in their lessons.

When it comes to the number of hours in which OOP is taught, there are big differences between schools. In some schools it is represented with just over 10 hours per subject while in other schools there are subjects fully dedicated to object oriented programming with large number of hours. Those differences can be observed in Chart 14.

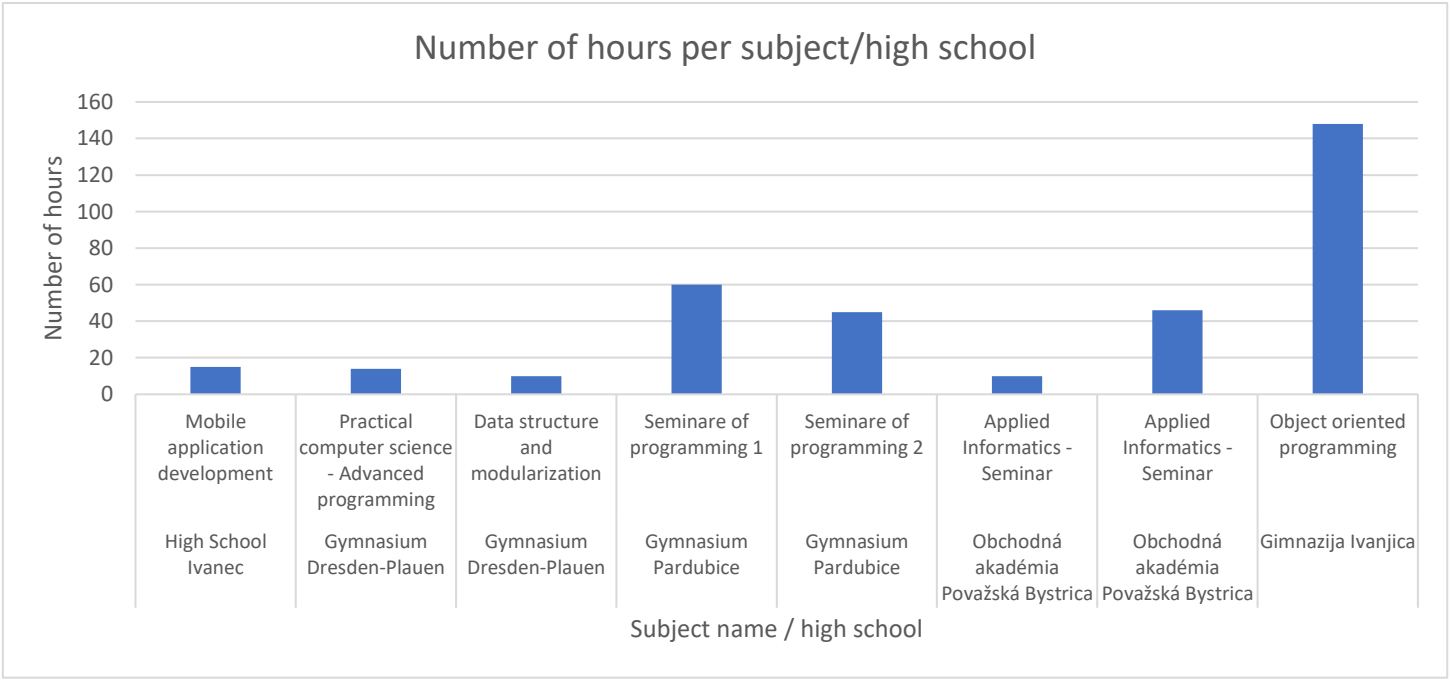


*Chart 14 - Distribution of hours per subject dedicated to OOP contents*

In some schools, object oriented programming is taught as a whole subject with full number of hours dedicated to OOP content, while in most schools OOP is taught just as partial topic in one or two subjects. For example, in Gimnazija Ivanjica (Serbia), OOP is taught in total of 148 hours in one subject fully dedicated to OOP. At the same time, in High school Ivanec (Croatia), OOP is taught only 15 hours, in one subject named Mobile application development, simply because OOP topics are needed for other subject contents. In other countries, subjects that contain OOP topics are Seminare of programming 1 and 2 (Gymnasium Pardubice), Applied Informatics – Seminars (Obchodná akadémia Považská Bystrica) and in Gymnasium Dresden-Plauen OOP is taught through subjects named Practical computer science - Advanced programming and Data structure and modularization.

With such big differences in curricula among different schools, it is quite clear that students cannot get same level of knowledge and unified skills in different countries in area of object oriented programming. This also results in different prior knowledge of the students which are needed to continue their education at the universities.

## 2.2. Analysis of teaching methods, types of activities, assessments and team work experience

### 2.2.1. Teaching methods

The next area that was analyzed for participating schools is teaching methods involved in educational process. Data which were gathered regarding teaching methods are types of teaching methods and explanations of how those methods were implemented with some simple examples.

Analysis shows that couple of teaching methods are used in most (if not all) of OOP subjects in high schools. Those methods are:

- explanation,
- programming/practical work,
- problem solving and
- questions and answers.

It is evident that in all subjects the common way of teaching is that teacher first explains new content and then students are performing practical work under teacher's supervision or on their own. Teacher explains basic concepts of given topics using presentations and demonstrations (for theoretical background of given topic) and then he uses different examples related to the topics for better explanation. For example, teacher shows how to write a code in given programming language or explains basic terms related to OOP (class, subclass, superclass, object) using visual elements and simple examples. That way students can notice differences visually for better understanding.

Programming (including practical work) is another common method used in teaching process. This method implies that contents and topics, that are explained by teacher, should be given a practical component. Concepts (that are explained theoretically) are implemented in programming language. Teacher shows how to write a code in programming language and students are following the instructions. After that, students are working on codes on their own in similar way, under supervision from a teacher (for example, teacher explains how to define a class, how to define attributes and

access modifiers for some attributes and students are doing the same for more attributes, they check how changing of access modifiers affects program execution etc.). Students are working on similar problems that were explained by the teacher, but analysis also shows that in some subjects, students are working on more complex problems, such as developing a working software product, for example, simple information system for business trip management or creating a visual (graphical) application. These kind of programming is mostly present in subjects with significant number of hours related to OOP.

Besides explanation and programming, problem solving is another method greatly used in OOP subjects in high schools. It means that students are given a specific problem which they have to solve by applying learned content and using gained practical programming skills. The difficulty of the problem (assignment) also depends on the number of practical hours that the students have achieved during classes, which is again directly related to the total number of hours of OOP in a particular subject. That means that students enrolled in OOP subject with larger number of hours are better prepared to solve more complex problems while the students enrolled in subjects where OOP is less represented have a lower ability to solve complex problems.

Another method greatly presented in teaching is a method of questions and answers. Analysis shows that students are free to ask questions if some concepts are not fully clear and then entire group starts a discussion about it, until concepts are explained. The questions are related to the given example and topic, for example, what is inheritance, what types of polymorphism are known in OOP, how can objects communicate with each other, etc.). That way a group discussion is forming which produces a peer learning, one of the most popular approaches in educational practice. Student that asked a question benefit from this approach, but also other students who are explaining and giving answers, because students think about the problem and give their own thoughts and suggestions.

All the mentioned methods and how often are they used in the subjects can be seen in Chart 15.

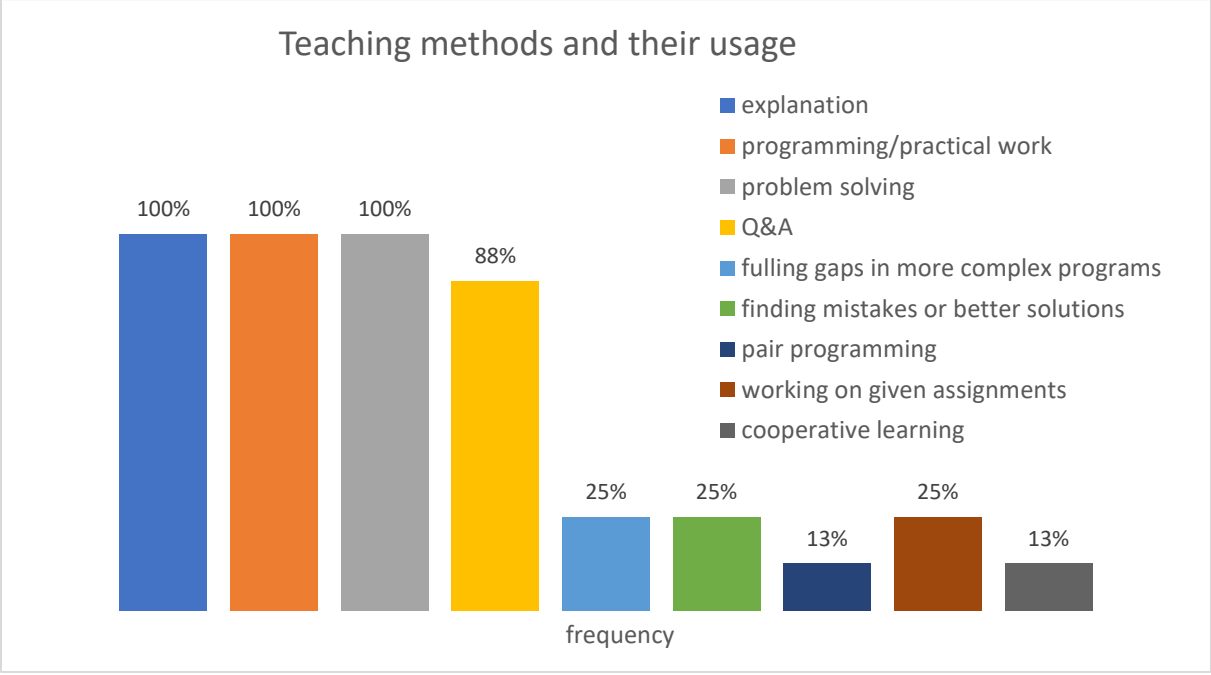


*Chart 15 - Representation of teaching methods used in high school subjects*

As shown in chart and mentioned in text earlier, explanation, programming/practical work, problem solving and questions and answers are the most common methods used in high school subjects. But,

besides these methods, it is evident that some other methods are also used, although in less amount of subjects.

For example, fulling gaps in more complex programs and finding mistakes or finding better solutions are used in 25% of subjects. That consists of giving students more complex structures and they have to add missing parts as well as optimization of program solutions and debugging (finding and correcting mistakes).

It was mentioned earlier that programming is one of the most common methods used for teaching OOP. In addition, in one of the analyzed subjects, students are doing pair programming, which means that two students (developers) work together on one station to design, code and test solutions. We can also see in Chart 4. that working on given assignments is represented in 25% of subjects. It implies that students work either on a small task for a short time, such as programming a calculator that can work with fractions, or on a more complex task over a longer period such as programming an information system for business trip management.

One final teaching method that is mentioned in this analysis is cooperative learning where students work together in small groups on a structured activity which, among other benefits, increases individual responsibility in each team member. This method is present only in one of analyzed subjects.

It is obvious that fair amount of teaching methods is used in teaching OOP in high school subjects. Some of these methods are very common in almost all schools, such as explanation, programming, problem solving and questions and answers. It is also evident that some schools and subjects use methods that are not common for majority of OOP subjects, but that depends on the fact whether OOP is the main content in a subject or it is taught as one of the topics in the subject just because it is needed for successful understanding other topics in that subject. This is also strictly related to the number of hours which are dedicated to OOP contents and that also directly dictates the depth to which one can go in terms of teaching new contents.

### 2.2.2. Types of activities

Similar to the teaching methods, activities for students and their variety in high school subjects were also analyzed. Students' activities are closely related to teaching methods and intertwine with each other. In high school subjects that were analyzed, there are two activities which dominate in educational process:

- discussion and
- practical work

In most of the analyzed subjects, discussion is stated as one of the most used activities for students. Students and teacher are constantly engaged in conversation to make sure that students fully understand the concepts. Students are also making discussion between themselves when they have to complete some tasks (for example, what attributes to include in some class, how to define methods etc.). Discussion can be done between students who are divided in groups or between students and teacher.

In one of the analyzed subjects, it is stated that during the lesson of theory, the teacher presents the topic through a presentation and shows and explains an example on the computer. Mostly these are

real-life examples, for easier understanding. During this time, students try that example on their computers.

While having practical classes, students first work with the teacher on computer examples, and then in pairs or small groups, they solve certain tasks assigned to them by the teacher. Students discuss, exchange ideas with each other and with the teacher, looking for the simplest possible solution. After that, they independently solve similar examples based on what they have learned so far, and they are also given tasks for more complex problems and their task is to find solutions on the Internet and understand those solutions.

This is one example of how to put a student in a center of educational process, but this kind of activity is not present in majority of the OOP related subjects. Again, same as with teaching methods, it is related to number of hours and depth to which certain contents are taught. It is obvious that in subjects where OOP is taught for fewer hours, there is not enough time for this type of activity.

Practical work, as the activity for students, is involved in all subjects. It consists of programming and problem solving when students have to make their own program solutions. Students, independently or in groups, repeat programming steps to solve problems and even to create more complex programs. Depending on complexity of given task, students are working either in pairs or individually. It is also noted that students start with more simple problems and create their own solutions step by step, which results in solving more complex problems. One of the example mentioned in analysis is the following one: The students start with projects composed of one class (for example, Date, Person, Animal) and try to implement them from basics. This part focuses on basic and medium algorithmization and on basic concepts of OOP. Later, they work with projects containing at most three or four classes and try to create simple programs, such as calculator, information system or text game. This part focuses on explanation of advanced concepts of OOP.

Very similar to the teaching methods, in classes, a lot of attention is paid to the practical work with the students. In this way, students acquire the necessary knowledge and skills to solve simple problems from everyday life using concepts of OOP. The range and complexity of the problems are related to the amount of OOP hours and ranges from solving the simplest problems in schools with a smaller number of OOP hours to more complex problems in schools where entire subjects are dedicated to OOP.

### 2.2.3. Assessments

The assessment of adoption of educational outcomes and acquired skills is also one of the very important activities in the educational process. It relies on the defined outcomes and contents that were taught in class, but also, the type of tasks that are evaluated must be in accordance with the tasks and practical problems that the students encountered during the classes.

The data collected by the analysis is quite superficial for some subjects, but it is obvious that the practical type of task appears in the assessments of all subjects. In about 40% of subjects, assessment consists of both theoretical and practical exam. Theoretical part is related to theoretical concepts where students have to prove they have learned the concepts of OOP.

There are differences in subjects of how practical skills are assessed. In 50% of subjects, students have to create independent software product. It is stated that students get marks for their work and it

consists of following criteria: exactness, completion and complexity of the student's solution. In some subjects, students also have to present their work.

In two of the analyzed subjects, as part of practical assignment, students must add several simple functionalities into existing project, but also demonstrate the skills of errors understanding and their correction. Students also work on projects. Within the course, it is necessary for students to work on the assigned (or selected) software project and then to defend the resulting software project in a suitable way.

In one subject that is fully dedicated to OOP (from Gimnazija Ivanjica, Serbia, with a total of 148 hours), the evaluation of the achievement of educational goals is done through monitoring students' activities in class and their progress during the school year. It consists of initial tests, assessing practical work on computers, dialogues with students, discussion in class, oral exams, students' participation during lessons, homeworks, presentations, development of projects tasks etc. The areas that are covered by assessment are: Historical development of object oriented paradigm, basic concepts of OOP, relationships between classes and polymorphism, creating a project task. For each area, there are three levels of achievement defined: basic, intermediate and advanced level and for each level there are outcomes and skills defined.

Considering the fact that this is the subject with the largest number of hours compared to the other analyzed subjects and because that is strictly OOP subjects (all the topics and contents in this subject are primary related to OOP), this cannot be taken as representative sample for all the subjects. All the outcomes that students achieve as well as skills they gain in these particular subjects will be analyzed in more details in vertical analysis between high schools and universities in the same countries. Of course, this type of analysis. Of course, such an analysis will also be conducted for all other schools and universities in the partner countries of the project.

### 2.2.4. Teamwork experience

From the aspect of education related to OOP, students' teamwork is a weak point in high schools, which can be observed in Chart 16.

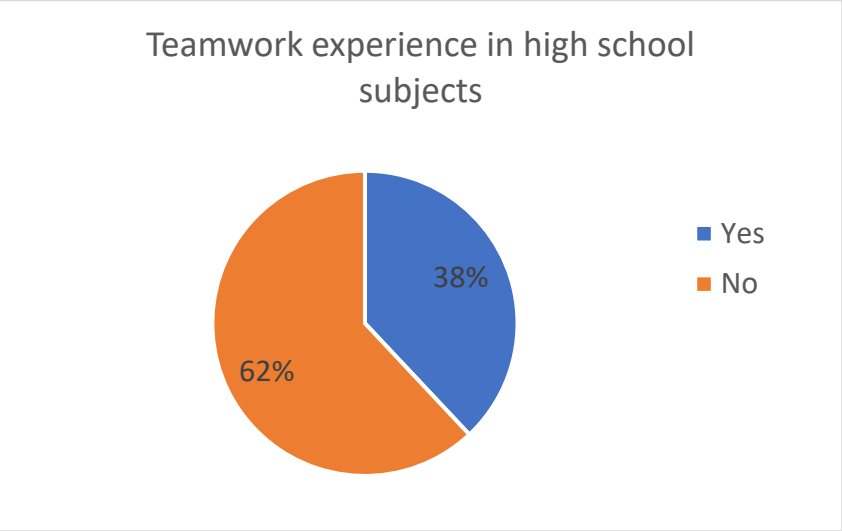


*Chart 16 - The presence of students' teamwork in high school OOP subjects*

In only 38% of subjects where OOP is taught, teamwork is present and students gain teamwork experience during lessons. In majority of subjects, students don't work in teams, they only sometimes collaborate when they work in pairs and have to solve a certain problem, but it is without any assigned roles.

For the subjects where teamwork is present, students work on different project tasks which are included in curriculum of computer science and partner work is common for programming tasks. In another subject, students are not separated into groups but everyone is involved in group. There are maximum of 10 students in the exercises and they work as a team. Students are more active and involved, express their opinions, cooperate with each other, solve set tasks together etc.

## 2.3. Analysis of literature and teaching materials

In terms of materials and literature that are used for teaching, teachers use materials intended for teaching object-oriented programming but also their own materials which they create specifically for their own classes. Different types of handbooks, textbooks, digital materials are used, but there also big differences in amount of literature that is used for different subjects. In Croatia, situation is as follows:

*Table 18 - Literature and other materials used in OOP subjects*

| Subject name | School, country | Used materials and literature | Additional comment |
|---|---|---|---|
| Mobile application development | High school Ivanec, Croatia | 1. STAPIĆ, Z., ŠVOGOR, I., FODREK, D.: Mobile application development, handbook for the 4th grade of high school, Varaždin, 2016, ISBN: 978-953-6071-54-8<br>2. VOLARIĆ, T., TOIĆ DLAČIĆ, K., IVOŠEVIĆ, I., DRAGANJAC, M.: Think IT, computer science textbook for the 4th grade of high school, Alfa d.d., Zagreb, 2021, ID: HR-ALFA-INF4-3489 | Teachers can choose one of the textbook from catalogue but also create their own materials. Not obligated to use textbooks. |

## 2.4. Analysis of suggestions on how to improve OOP teaching in schools

When analyzing their curricula of the subjects related to OOP, all the partners put their own suggestions on what can be improved in OOP teaching and how to do it. There are many suggestions that could be considered and it's very clear that they depend on differences in educational systems among countries. There is also a fact that this analysis covers different curricula (different subjects) in which OOP is taught, which means that maybe a suggestion for improvement by one partner is already

adopted in curricula and classes for another partner. A list of problems and possible suggestions and solutions in Croatia are shown in Table 19.

*Table 19 - Problems that teachers are facing with and suggestions for improvement the quality of classes*

| Country | Problems and suggestions for improvement |
|---|---|
| Croatia | • More practical tasks which will include students cooperating and working together (teamwork)<br>• More materials with practical tasks and exercises to support teacher's lessons<br>• More hours dedicated to OOP in compulsory subjects in high schools<br>• In primary schools, informatics and computer science in general should be obligatory for all pupils, so they all enroll high schools with same skills and knowledge |

It is evident that teachers are facing with different kind of problems and obstacles in their classes, but in general, all the problems are generated around two important areas: lack of literature and reduced amounts of teaching hours related to OOP. Regarding literature, teachers try to overcome this obstacle in different ways, from creating their own materials to searching different sources (textbooks, magazines, digital platforms) for more examples which can be used with students. Regarding the problem of small number of hours in which OOP is covered, it is mainly prescribed by the curricula of the teaching subjects, and teachers themselves can partially influence it.

## 2.5. Additional comments

Due to pedagogical standards in Croatia, which prescribe minimum number of students that are necessary to be enrolled in subject (minimum of 10 students) and number of students which is continuously decreasing at national level, it is very hard to gather a big enough group so the school is allowed to conduct compulsory subject.

## 2.6. Review of additional subjects related to programming in general

The subjects that were analyzed earlier are very much related to OOP, so they were the main focus for analysis. Besides those subjects, there are several more subjects from IT field in every school that are not related to OOP, but some of the topics that are taught in those subjects are prerequisite for successful adoption of OOP contents. The brief description of contents of those subjects are shown in Table 20.

*Table 20 - Other IT subjects taught in partner institutions*

| School | Subject name | Grade | Topics |
|---|---|---|---|
| High school Ivanec | Informatics 1 - obligatory subject | 1st grade | programming languages, algorithm, pseudocode, variables, data types, input/output operations, relation, arithmetic and logic expressions, basic algorithmic structures (sequence, selection, iteration), |

| | | | |
|---|---|---|---|
| | | | analysis of the algorithm, correctness of the algorithm, error correction, simple problem solving (mathematical problems), implement solutions in Python |
| | Informatics 2 - optional subject | 2nd grade | one-dimensional data structures (string, array), nested loops, data indexing, more complex problem solving, implement solutions in Python |
| | Informatics 3 - optional subject | 3rd grade | using concepts from Informatics 1 and Informatics 2 to solve more complex problems, sorting algorithms, search algorithms, recursion, user defined functions, work with text files, using graphical modules to visualize simple problems, implement solutions in Python |

All the subjects mentioned in the Table 20 are related to IT and they are covering big variety of topics. Some of the listed contents are important for starting to learn programming and are more or less related to the contents of OOP.