



CURRENT STATE OF PROGRAMMING IN
PROJECT PARTNERS' HIGH SCHOOLS AND
UNIVERSITIES - HORIZONTAL ANALYSIS



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

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	2022

Disclaimer:

Funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or Slovak Academic Association for International Cooperation. Neither the European Union nor the granting authority can be held responsible for them.

Table of contents

1.	Horizontal analysis of universities' data.....	8
1.1.	Analysis of OOP load	8
1.1.1.	FOI	12
1.1.2.	UNIZA.....	16
1.1.3.	UPCE	19
1.1.4.	FON	22
1.1.5.	HTW	24
1.1.6.	Summary.....	27
1.2.	Analysis of prior requirements of universities' OOP related courses	30
1.2.1.	OOP teaching courses.....	31
1.2.1.1.	FOI.....	31
1.2.1.2.	HTW	32
1.2.1.3.	FON	33
1.2.1.4.	UNIZA.....	33
1.2.1.5.	UPCE	35
1.2.1.6.	Prior requirements matrix.....	39
1.2.2.	OOP practicing courses.....	41
1.2.2.1.	FOI.....	42
1.2.2.2.	HTW	44
1.2.2.3.	FON	45
1.2.2.4.	UNIZA.....	45
1.2.2.5.	UPCE	48
1.2.2.6.	Prior requirements matrix.....	48
1.2.3.	OOP using courses.....	50
1.2.3.1.	FOI.....	51
1.2.3.2.	HTW	52
1.2.3.3.	FON	52
1.2.3.4.	UNIZA.....	53
1.2.3.5.	UPCE	53
1.2.3.6.	Prior requirements matrix.....	53
1.2.4.	Conclusion	55
1.3.	Analysis of approaches for teaching OOP	59
1.3.1.	Remarks on gathered data	59

1.3.2.	Identified themes	59
1.3.2.1.	Forms of instruction / forms of knowledge transfer	59
1.3.2.2.	Individual work	60
1.3.2.3.	Assessment	61
1.3.2.4.	Tools	62
1.3.3.	Conclusion	64
2.	Horizontal analysis of high schools' data	65
2.1.	Analysis of OOP load	66
2.2.	Analysis of learning outcomes and topics and their comparison for universities and high schools.....	69
2.2.1.	Analysis and comparison	69
2.2.2.	Conclusion	78
2.2.3.	'Light OOP' topics	79
2.3.	Analysis of teaching methods, types of activities, assessments and team work experience	79
2.3.1.	Teaching methods	79
2.3.2.	Types of activities	81
2.3.3.	Assessments	82
2.3.4.	Teamwork experience	83
2.4.	Analysis of literature and teaching materials.....	84
2.5.	Analysis of suggestions on how to improve OOP teaching in schools	86
2.6.	Additional comments	87
2.7.	Review of additional subjects related to programming in general.....	88
	Bibliography.....	91

List of tables

Table 1 - University courses categorization	10
Table 2 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on FOI	12
Table 3 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 1st year of study on FOI	12
Table 4 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on FOI	13
Table 5 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on FOI	13
Table 6 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 3rd year of study on FOI	14
Table 7 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 3rd year of study on FOI.....	14
Table 8 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on UNIZA	16
Table 9 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 1st year of study on UNIZA	16
Table 10 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on UNIZA	17
Table 11 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on UNIZA	17
Table 12 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on UPCE	19
Table 13 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 1st year of study on UPCE	19
Table 14 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on UPCE	20
Table 15 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on UPCE.....	20
Table 16 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on FON.....	22
Table 17 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on FON	22
Table 18 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on HTW.....	24
Table 19 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 1st year of study on HTW.....	24
Table 20 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on HTW	25
Table 21 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on HTW	25
Table 22 - OOP teaching courses considered in the prior requirements analysis	31
Table 23 - Prior requirements analysis of course Object-oriented programming on FOI.....	32
Table 24 - Prior requirements analysis of course Programming II on HTW	32

Table 25 - Prior requirements analysis of course Programming 2 on FON	33
Table 26 - Prior requirements analysis of course Informatics 1 on UNIZA	33
Table 27 - Prior requirements analysis of course Practice of programming 1 on UNIZA.....	34
Table 28 - Prior requirements analysis of course Informatics 2 on UNIZA	34
Table 29 - Prior requirements analysis of course Practice of programming 2 on UNIZA.....	34
Table 30 - Prior requirements analysis of course Basics of Algorithmization on UPCE	35
Table 31 - Prior requirements analysis of course Algorithmization and programming practicum on UPCE	35
Table 32 - Prior requirements analysis of course Basics of Programming Using Java Programming Language on UPCE.....	36
Table 33 - Prior requirements analysis of course Object Oriented Programming on UPCE	37
Table 34 - Prior requirements analysis of course Language C++ I on UPCE.....	38
Table 35 - Prior requirements analysis of course The C# Programming Language on UPCE	39
Table 36 - Prior requirements matrix of OOP teaching courses	40
Table 37 - Distribution of requirements of OOP teaching courses	41
Table 38 - OOP practicing courses considered in the prior requirements analysis	41
Table 39 - Prior requirements analysis of course Windows Applications Development on FOI.....	42
Table 40 - Prior requirements analysis of course Programming 2on FOI	43
Table 41 - Prior requirements analysis of course Mobile applications and games development on FOI	44
Table 42 - Prior requirements analysis of course Software Engineering 2 on HTW	45
Table 43 - Prior requirements analysis of course Programming of component architectures on HTW.....	45
Table 44 - Prior requirements analysis of course Informatics 3 on UNIZA	46
Table 45 - Prior requirements analysis of course Algorithms and Data structures 1 on UNIZA	47
Table 46 - Prior requirements analysis of course Data Structures on UPCE	48
Table 47 - Prior requirements matrix of OOP practicing courses	49
Table 48 - Distribution of requirements of OOP practicing courses	50
Table 49 - OOP using courses considered in the prior requirements analysis.....	51
Table 50 - Prior requirements analysis of course Programming in Python on FOI	51
Table 51 - Prior requirements analysis of course Programming I on HTW	52
Table 52 - Prior requirements analysis of course Programming distributed systems on HTW	52
Table 53 - Prior requirements analysis of course Data structures and algorithms on FON.....	53
Table 54 - Prior requirements matrix of OOP using courses.....	54
Table 55 - Distribution of requirements of OOP using courses.....	55
Table 56 - Distribution of requirements of all OOP courses	56
Table 57 - Suggested or mandated IDEs per programming language.....	63
Table 58 - Basic data of OOP related subjects.....	67
Table 59 - Literature and other materials used in OOP subjects	85
Table 60 - Problems that teachers are facing with and suggestions for improvement the quality of classes.....	86
Table 61 - Other IT subjects taught in partner institutions.....	88

List of charts

Chart 1 - Hierarchy of OOP course categories.....	9
Chart 2 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on FOI.....	12
Chart 3 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on FOI	13
Chart 4 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 3rd year of study on FOI	14
Chart 5 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on FOI	15
Chart 6 - Total OOP teaching/practicing/using hours distribution between 3 years of study on FOI of courses teaching OOP	15
Chart 7 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on UNIZA.....	16
Chart 8 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on UNIZA.....	17
Chart 9 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on UNIZA	18
Chart 10 - Total OOP teaching/practicing/using hours distribution between 3 years of study on UNIZA of courses teaching OOP	18
Chart 11 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on UPCE.....	19
Chart 12 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on UPCE	20
Chart 13 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on UPCE	21
Chart 14 - Total OOP teaching/practicing/using hours distribution between 3 years of study on UPCE of courses teaching OOP	21
Chart 15 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on FON.....	22
Chart 16 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on FON	23
Chart 17 - Total OOP teaching/practicing/using hours distribution between 3 years of study on FON of courses teaching OOP	23
Chart 18 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on HTW	24
Chart 19 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on HTW	25
Chart 20 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on HTW	26
Chart 21 - Total OOP teaching/practicing/using hours distribution between 3 years of study on HTW of courses teaching OOP	26
Chart 22 - Comparison of total hours of teaching OOP of mandatory courses between all universities in every year	27

Chart 23 - Comparison of total hours related to OOP of mandatory courses between all universities in every year	27
Chart 24 - Comparison of total hours of teaching OOP of optional courses between all universities in every year	28
Chart 25 - Comparison of total hours related to OOP of optional courses between all universities in every year	28
Chart 26 - Comparison of total hours of teaching OOP of OOP teaching courses between all universities in every year	29
Chart 27 - Comparison of total hours of teaching OOP of OOP practicing courses between all universities in every year.....	29
Chart 28 - Forms of knowledge transfer used in lectures	59
Chart 29 - Learning-by-doing approach in laboratory exercises	60
Chart 30 - Frequency of assessment methods for theoretical knowledge	61
Chart 31 - Type of knowledge assessed in analyzed courses	62
Chart 32 - Programming language occurrence in analyzed courses.....	62
Chart 33 - Most frequently used IDEs in analyzed courses	64
Chart 34 - High schools and numbers of OOP related subjects	66
Chart 35 - Categorisation of subjects and their number per category	68
Chart 36 - Distribution of hours per subject dedicated to OOP contents	68
Chart 37 – Number of universities teaching each topic.....	70
Chart 38 – Number of high schools teaching each topic.....	71
Chart 39 – Topics analysis in Czech Republic	72
Chart 40 – Topics analysis in Slovakia	73
Chart 41 – Topics analysis in Germany.....	74
Chart 42 – Topics analysis in Croatia	76
Chart 43 – Topics analysis in Serbia	77
Chart 44 - Representation of teaching methods used in high school subjects.....	80
Chart 45 - The presence of students' teamwork in high school OOP subjects	84

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.

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): $T_\sigma \in \{M, O\}$
- b. Year of study: $y_\sigma \in \{1, 2, 3\}$
- c. Total hours: TH_σ
- 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 hours 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.

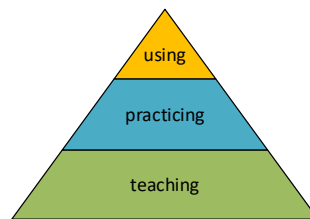


Chart 1 - Hierarchy of OOP course categories

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
HTW	Programming I	Mandatory	1	75	0	0%	6	8%	OOP Using
HTW	Programming II	Mandatory	1	60	50	83%	10	17%	OOP Teaching
HTW	Software Engineering 2	Mandatory	2	60	30	50%	30	50%	OOP Practising
HTW	Programming distributed systems	Optional	2	60	0	0%	10	17%	OOP Using
HTW	Programming of component architectures	Optional	2	60	10	17%	20	33%	OOP Practising
FON	Programming 2	Mandatory	2	52	48	92%	4	8%	OOP Teaching
FON	Data structures and algorithms	Mandatory	2	52	8	15%	6	12%	OOP Using
UNIZA	Informatics 1	Mandatory	1	65	60	92%	0	0%	OOP Teaching
UNIZA	Informatics 2	Mandatory	1	65	45	69%	0	0%	OOP Teaching
UNIZA	Practice of programming 1	Optional	1	26	20	77%	0	0%	OOP Teaching
UNIZA	Practice of programming 2	Optional	1	26	20	77%	0	0%	OOP Teaching
UNIZA	Informatics 3	Mandatory	2	65	15	23%	40	62%	OOP Practising
UNIZA	Algorithms and Data structures 1	Mandatory	2	52	13	25%	13	25%	OOP Practising
UPCE	Basics of Algorithmization	Mandatory	1	26	22	85%	4	15%	OOP Teaching
UPCE	Algorithmization and programming practicum	Mandatory	1	26	24	92%	2	8%	OOP Teaching
UPCE	Basics of Programming Using Java Programming Language	Mandatory	1	52	52	100%	0	0%	OOP Teaching
UPCE	Object Oriented Programming	Mandatory	2	65	65	100%	0	0%	OOP Teaching
UPCE	Data Structures	Mandatory	2	52	26	50%	26	50%	OOP Practising
UPCE	Language C++ I	Mandatory	2	52	52	100%	0	0%	OOP Teaching
UPCE	The C# Programming Language	Mandatory	2	52	52	100%	0	0%	OOP Teaching

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.

1.1.1. FOI

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 2, Chart 3 and Chart 4. Overall analysis of relevant FOI courses is presented in charts Chart 5 and Chart 6.

1st year of study

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

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

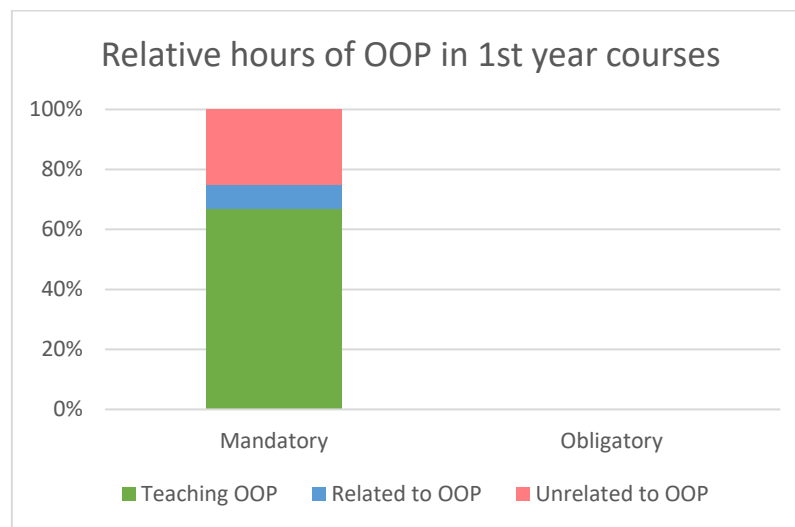


Chart 2 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on FOI

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		Related to OOP		Unrelated to OOP	
			Σ	%	Σ	%	Σ	%
OOP Teaching	OOP Teaching	60	60	100	0	0	0	0
	OOP Practising	60	20	33,33333	10	16,66667	30	50
	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	

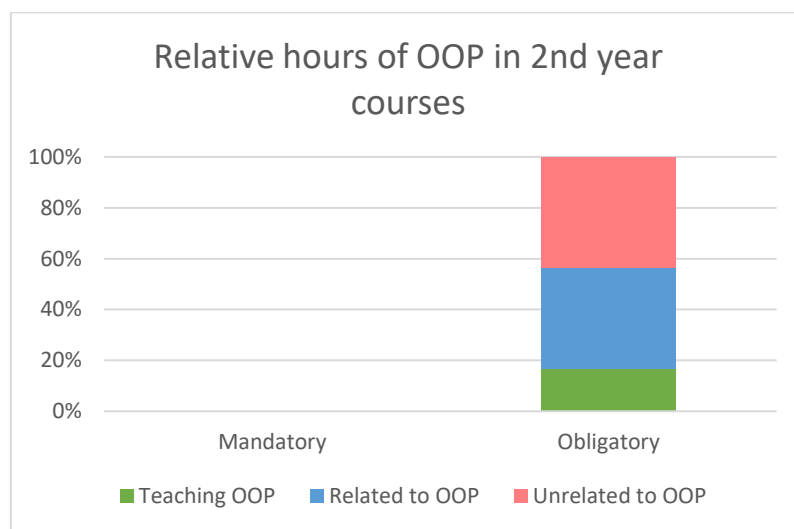


Chart 3 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on FOI

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	

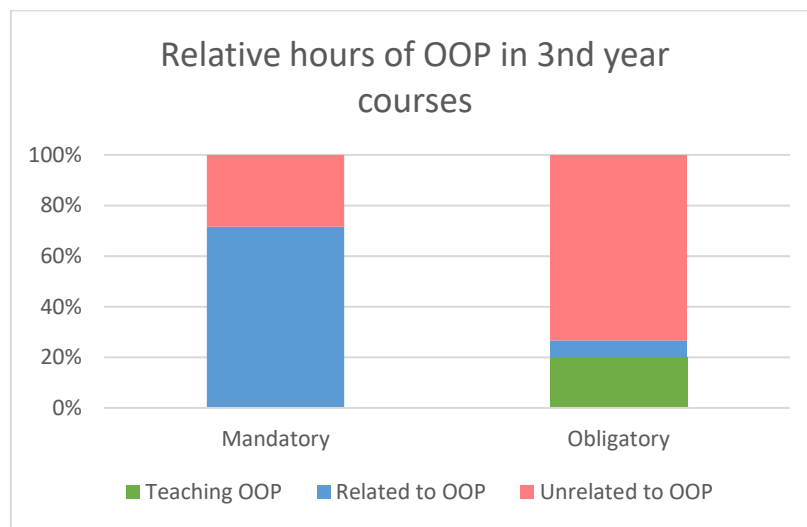


Chart 4 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 3rd year of study on FOI

Table 7 - Total OOP teaching/related/unrelated hours per OOP teaching/practising 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

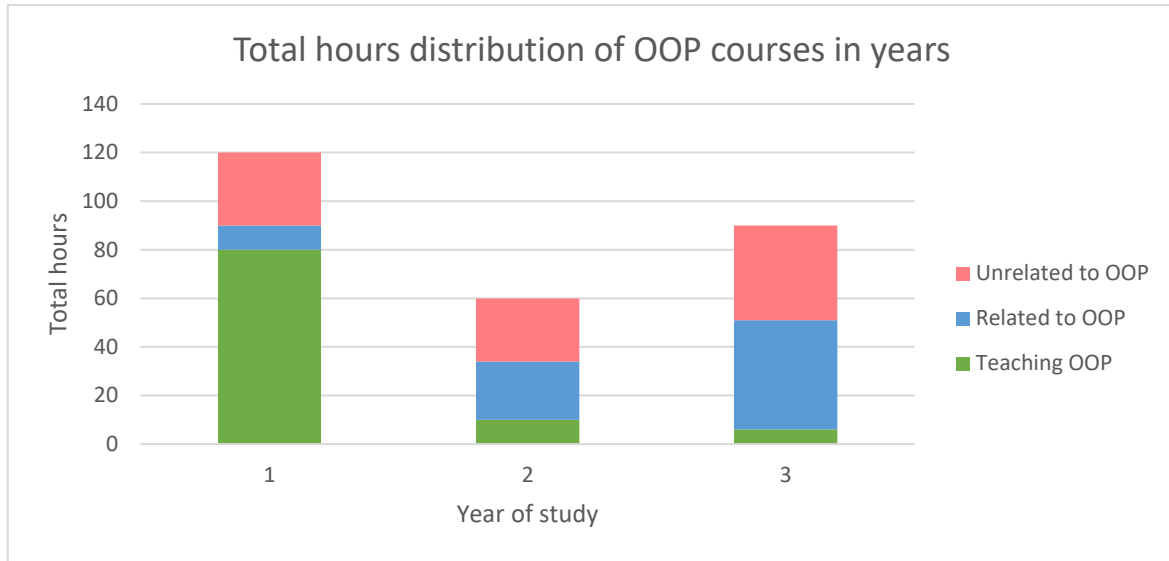


Chart 5 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on FOI

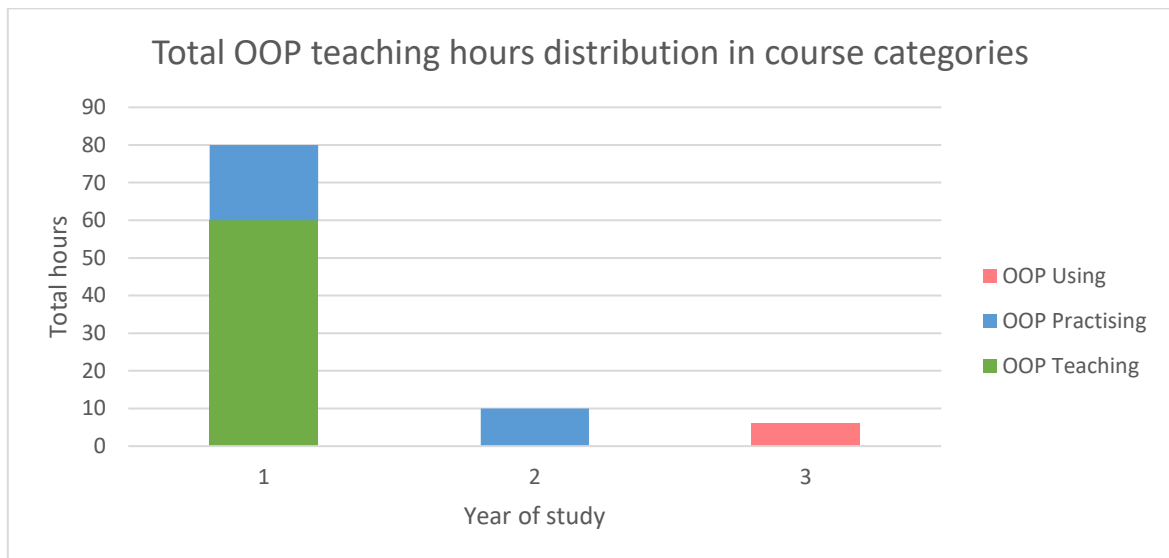


Chart 6 - Total OOP teaching/practising/using hours distribution between 3 years of study on FOI of courses teaching OOP

1.1.2. UNIZA

The informatics related study program at UNIZA places courses that are relevant to OOP in first two years of study. Tables Table 8 and Table 9 summarize 1st year of study, Table 10 and Table 11 Table 4 and

Table 5 summarize 2nd year of study. Visualization of relative distributions of hours in respective years of study are presented in Chart 7 and Chart 8. Overall analysis of relevant UNIZA courses is presented in charts Chart 9 and Chart 10.

1st year of study

Table 8 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on UNIZA

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	130	105	80,76923	0	0	25	19,23077
Optional	52	40	76,92308	0	0	12	23,07692
Σ	182	145		0		37	

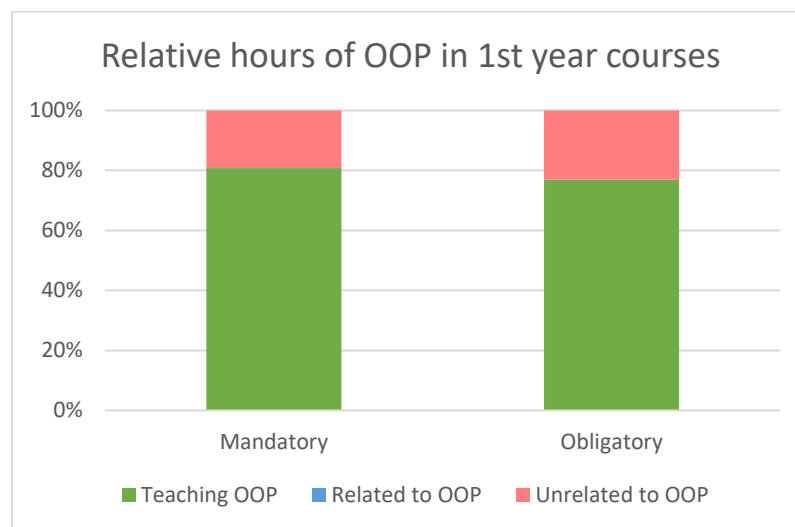


Chart 7 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on UNIZA

Table 9 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 1st year of study on UNIZA

Course category	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP					
		Σ	%	Σ	%	Σ	%				
OOP Teaching	182	145	79,67033	0	0	37	20,32967				
OOP Practising								0	0	0	0
OOP Using								0	0	0	0
Σ	182	145		0		37					

2nd year of study

Table 10 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on UNIZA

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	117	28	23,93162	53	45,29915	36	30,76923
Optional	0	0	0	0	0	0	0,00000
Σ	117	28		53		36	

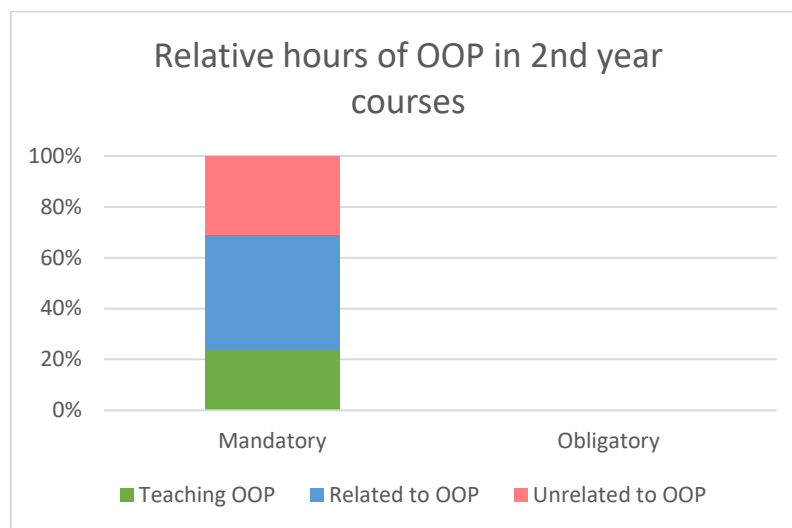


Chart 8 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on UNIZA

Table 11 - Total OOP teaching/related/unrelated hours per OOP teaching/practising and using subjects of 2nd year of study on UNIZA

Course category	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
OOP Teaching OOP	0	0	0	0	0	0	
Practising OOP	117	28	23,93162	53	45,29915	36	30,76923
Using OOP	0	0	0	0	0	0	
Σ	117	28		53		36	

Overall

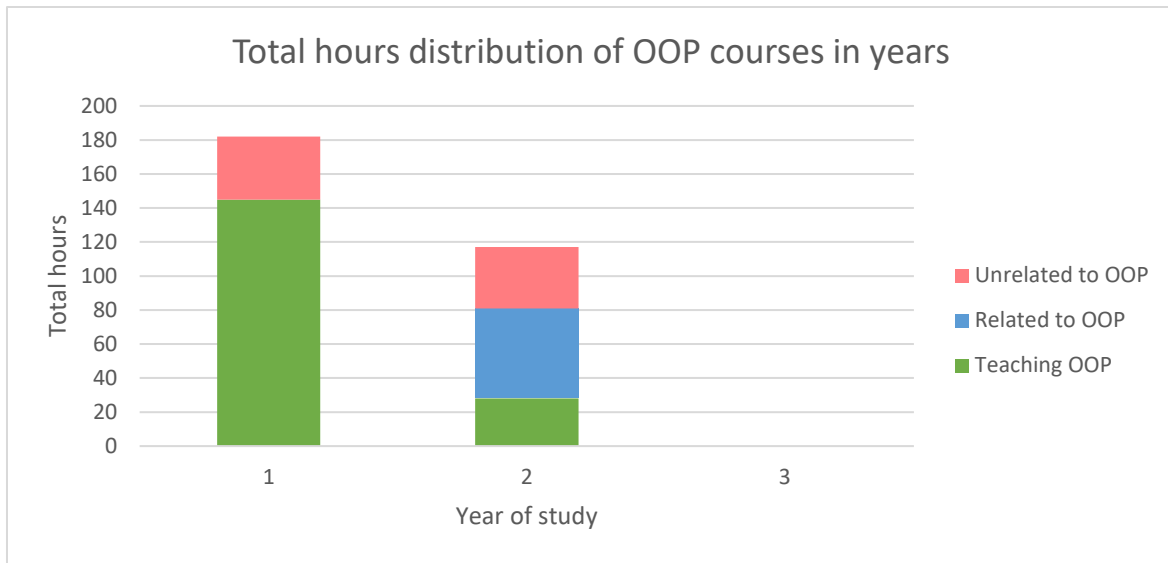


Chart 9 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on UNIZA

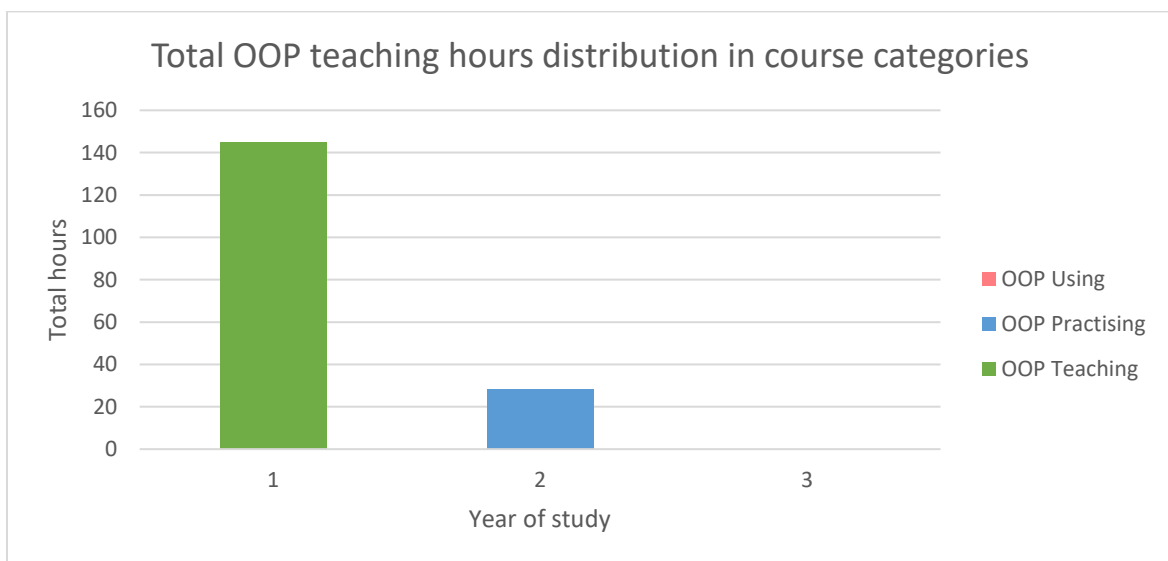


Chart 10 - Total OOP teaching/practising/using hours distribution between 3 years of study on UNIZA of courses teaching OOP

1.1.3. UPCE

The informatics related study program at UPCE places courses that are relevant to OOP in first two years of study. Tables Table 12 and Table 13 summarize 1st year of study, Table 14 and Table 15 Table 4 summarize 2nd year of study. Visualization of relative distributions of hours in respective years of study are presented in Chart 11 and Chart 12. Overall analysis of relevant UPCE courses is presented in charts Chart 13 and Chart 14.

1st year of study

Table 12 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on UPCE

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	104	98	94,23077	6	5,769231	0	0
Optional	0	0	0	0	0	0	0
Σ	104	98		6		0	0

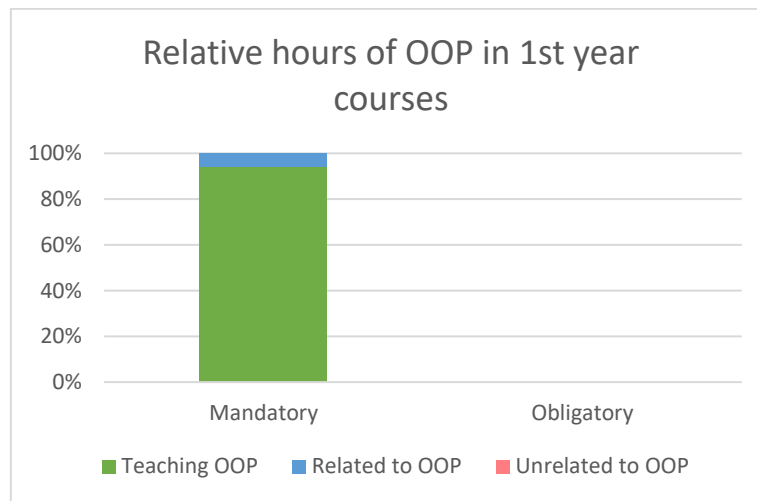


Chart 11 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on UPCE

Table 13 - Total OOP teaching/related/unrelated hours per OOP teaching/practising and using subjects of 1st year of study on UPCE

Course category	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
OOP Teaching	104	98	94,23077	6	5,769231	0	0
OOP Practising	0	0	0	0	0	0	0
OOP Using	0	0	0	0	0	0	0
Σ	104	98		6		0	0

2nd year of study

Table 14 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on UPCE

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	221	195	88,23529	26	11,76471	0	0
Optional	0	0	0	0	0	0	0,00000
Σ	221	195		26		0	

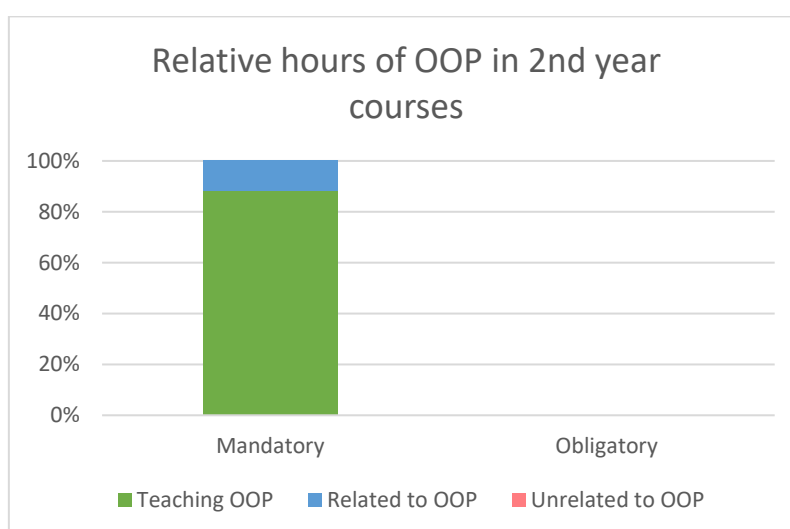


Chart 12 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on UPCE

Table 15 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on UPCE

Course category	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
OOP Teaching	169	169	100	0	0	0	0
OOP Practising	52	26	50	26	50	0	0
OOP Using	0	0	0	0	0	0	0
Σ	221	195		26		0	

Overall

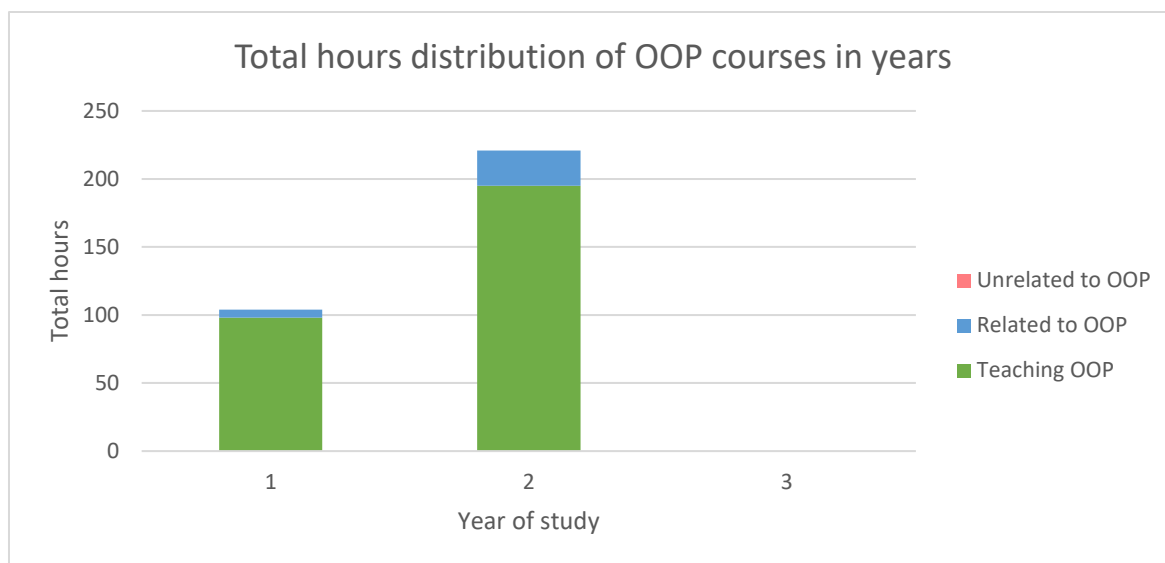


Chart 13 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on UPCE

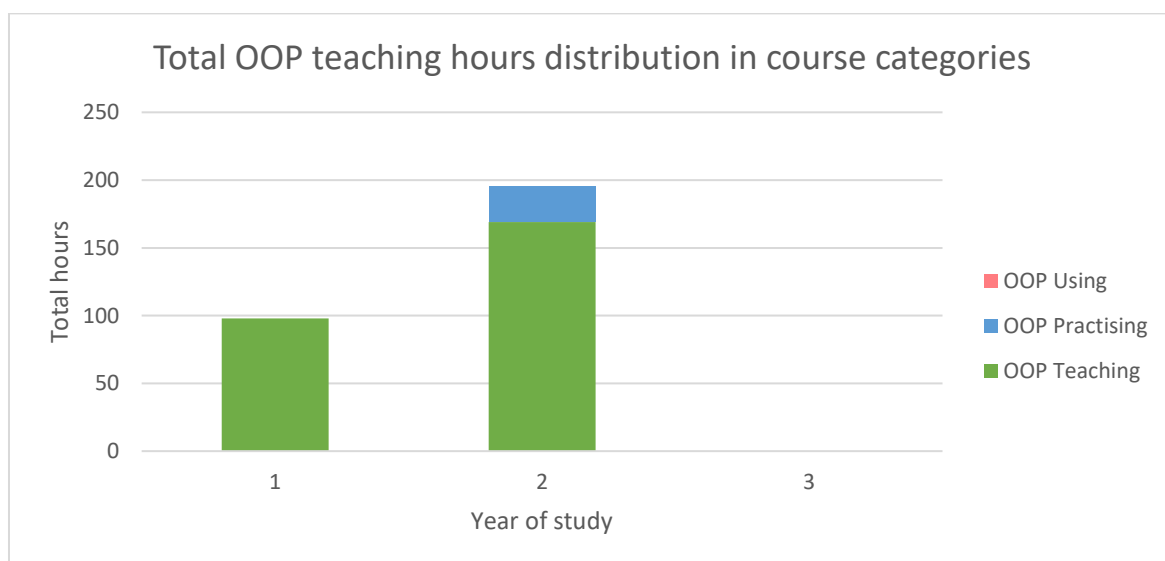


Chart 14 - Total OOP teaching/practising/using hours distribution between 3 years of study on UPCE of courses teaching OOP

1.1.4. FON

The informatics related study program at FON places courses that are relevant to OOP in 2nd year of study. Table 16 and Table 17 summarize data. Visualization of relative distributions of hours in 2nd year of study is presented in Chart 15. Overall analysis of relevant FON courses is presented in charts Chart 16 and Chart 17.

2nd year of study

Table 16 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on FON

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	104	56	53,84615	10	9,615385	38	36,53846
Optional	0	0	0	0	0	0	0,00000
Σ	104	56		10		38	

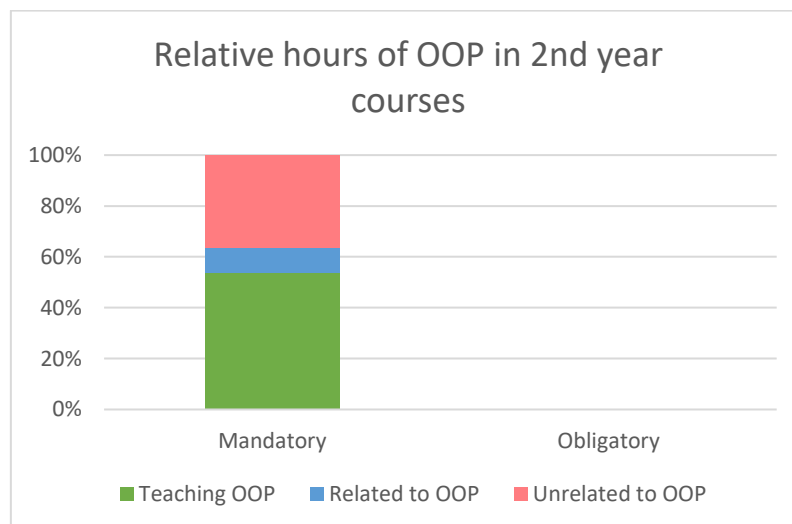


Chart 15 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on FON

Table 17 - Total OOP teaching/related/unrelated hours per OOP teaching/practicing and using subjects of 2nd year of study on FON

Course category	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
OOP Teaching OOP	52	48	92,30769	4	7,692308	0	0
Practising OOP	0	0	0	0	0	0	0
Using OOP	52	8	15,38462	6	11,53846	38	73,07692
Σ	104	56		10		38	

Overall

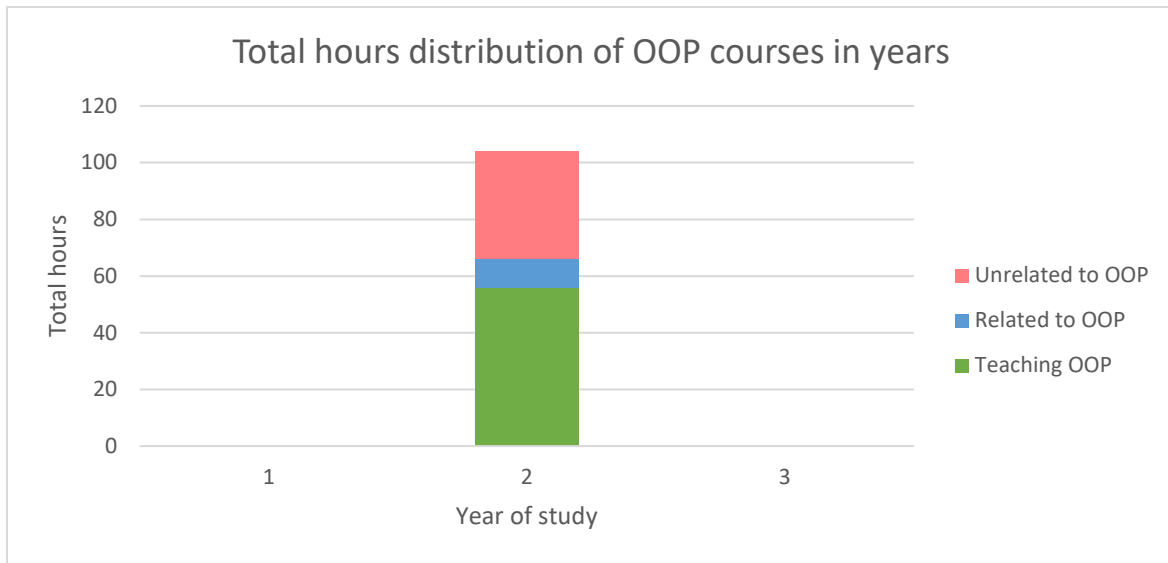


Chart 16 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on FON

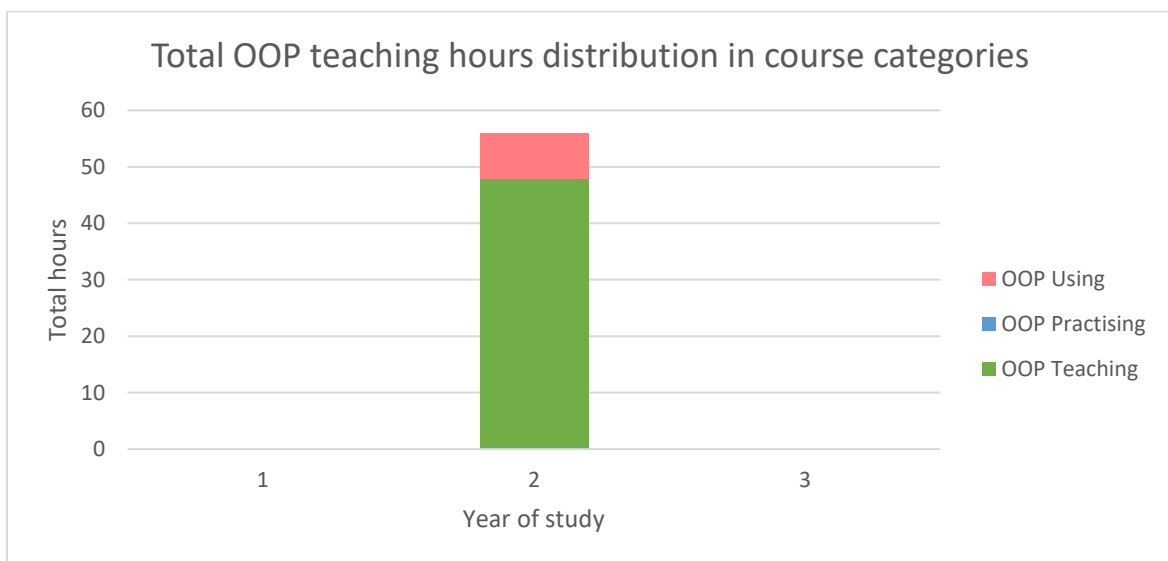


Chart 17 - Total OOP teaching/practising/using hours distribution between 3 years of study on FON of courses teaching OOP

1.1.5. HTW

The informatics related study program at HTW places courses that are relevant to OOP in first two years of study. Tables Table 18 and Table 19 summarize 1st year of study, Table 20 and Table 21 summarize 2nd year of study. Visualization of relative distributions of hours in respective years of study are presented in Chart 18 and Chart 19. Overall analysis of relevant UPCE courses is presented in charts Chart 20 and Chart 21.

1st year of study

Table 18 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 1st year of study on HTW

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	135	50	37,03704	16	11,85185	69	51,11111
Optional	0	0	0	0	0	0	0
Σ	135	50		16		69	

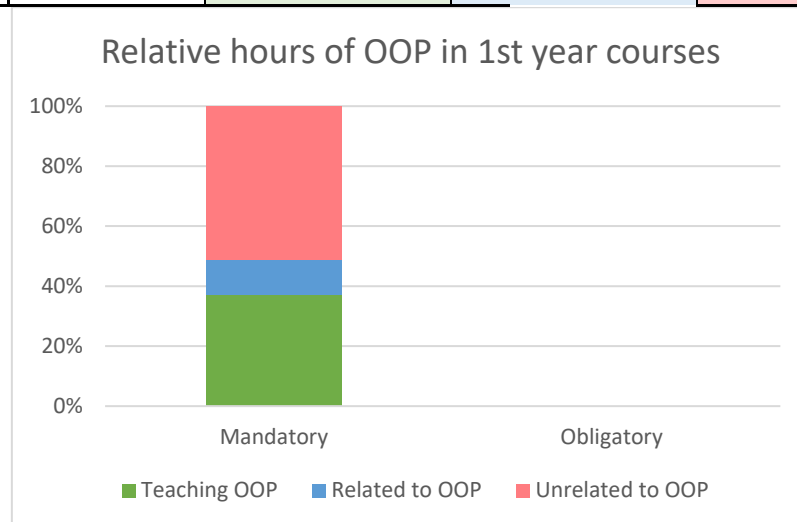


Chart 18 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 1st year of study on HTW

Table 19 - Total OOP teaching/related/unrelated hours per OOP teaching/practising and using subjects of 1st year of study on HTW

Course category	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
OOP Teaching	60	50	83,33333	10	16,66667	0	0
OOP Practising	0	0	0	0	0	0	0
OOP Using	75	0	0	6	8	69	92
Σ	135	50		16		69	

2nd year of study

Table 20 - Total OOP teaching/related/unrelated hours per mandatory and optional subjects of 2nd year of study on HTW

Course type	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
Mandatory	60	30	50	30	50	0	0
Optional	120	10	8,333333	30	25	80	66,66667
Σ	180	40		60		80	

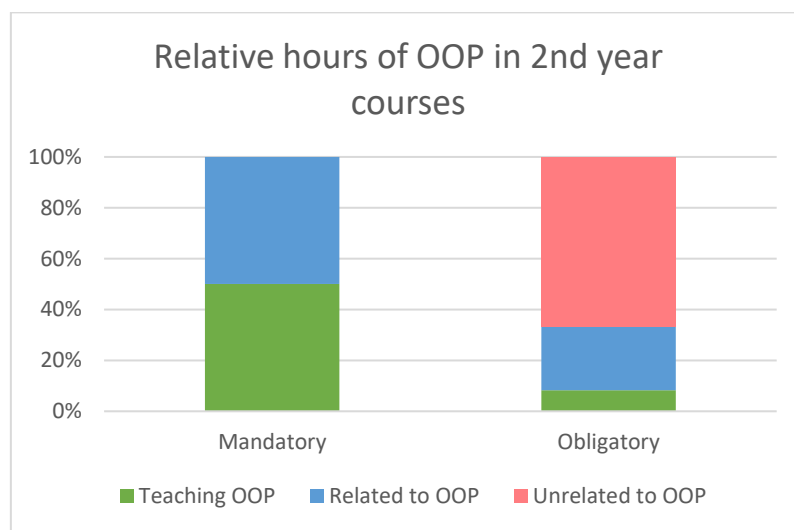


Chart 19 - Relative distribution of OOP teaching/related/unrelated hours between mandatory and optional subjects of 2nd year of study on HTW

Table 21 - Total OOP teaching/related/unrelated hours per OOP teaching/practising and using subjects of 2nd year of study on HTW

Course category	Hours per year						
	Σ Course total	Teaching OOP		Related to OOP		Unrelated to OOP	
		Σ	%	Σ	%	Σ	%
OOP Teaching OOP	0	0	0	0	0	0	
Practising OOP Using	120	40	33,33333	50	41,66667	30	25
Σ	60	0	0	10	16,66667	50	83,33333
Σ	180	40		60		80	

Overall

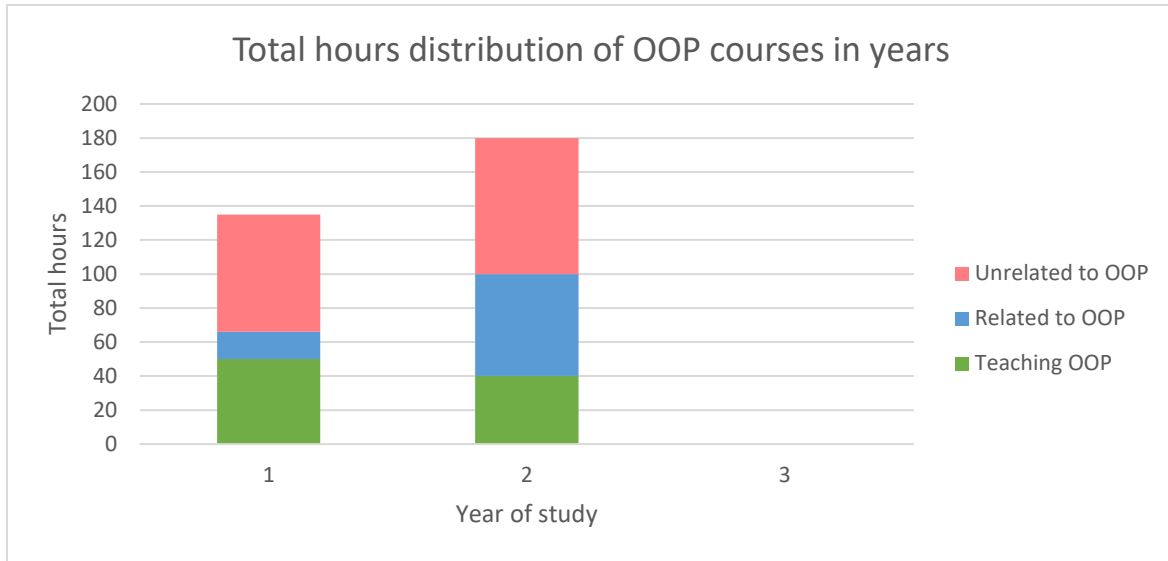


Chart 20 - Total OOP teaching/related/unrelated hours distribution between 3 years of study on HTW

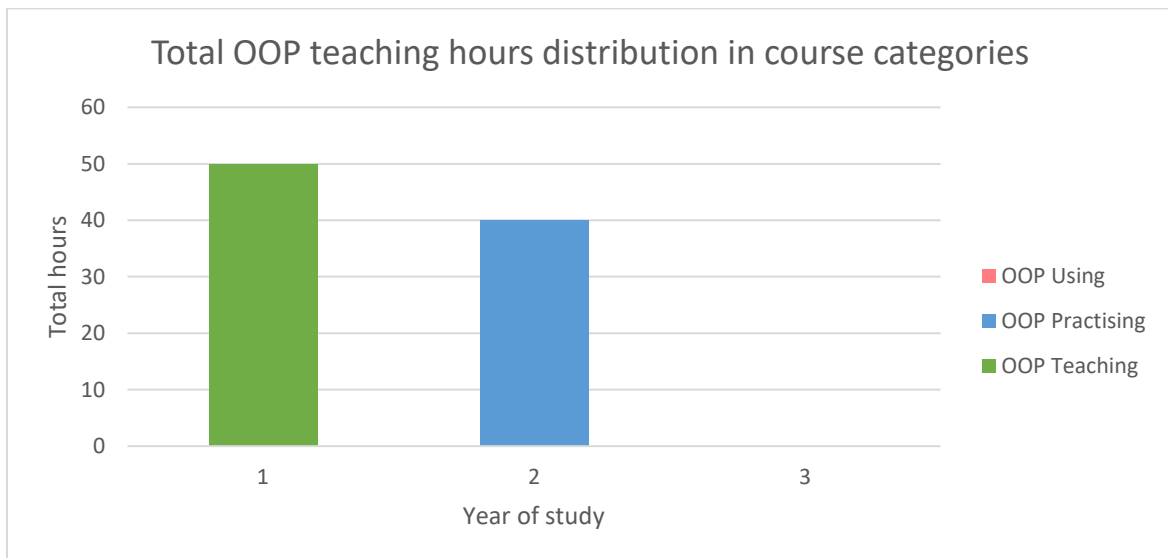


Chart 21 - Total OOP teaching/practicing/using hours distribution between 3 years of study on HTW of courses teaching OOP

1.1.6. Summary

In order to summarize all data we performed last comparative analysis. First we compared sum of values $TH_{\sigma}^{teaching}$ and $TH_{\sigma}^{related}$ of both mandatory and optional courses. These charts are depicted below:

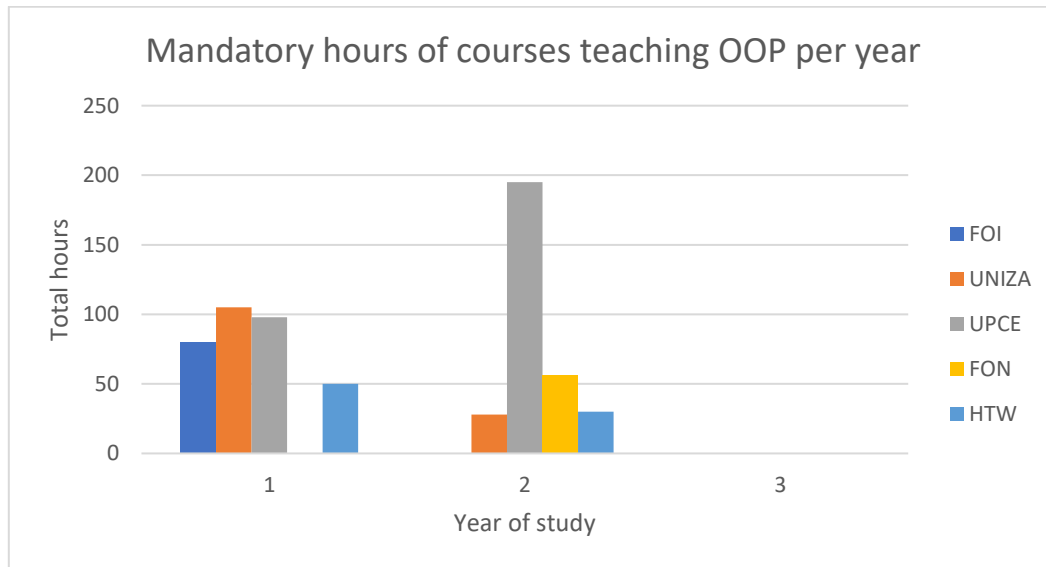


Chart 22 - Comparison of total hours of teaching OOP of mandatory courses between all universities in every year

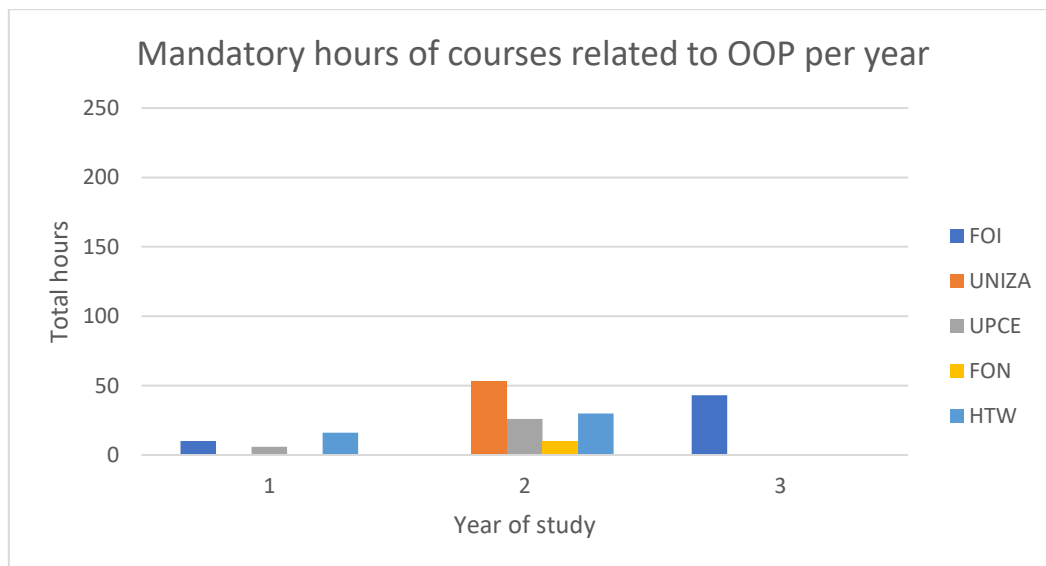


Chart 23 - Comparison of total hours related to OOP of mandatory courses between all universities in every year

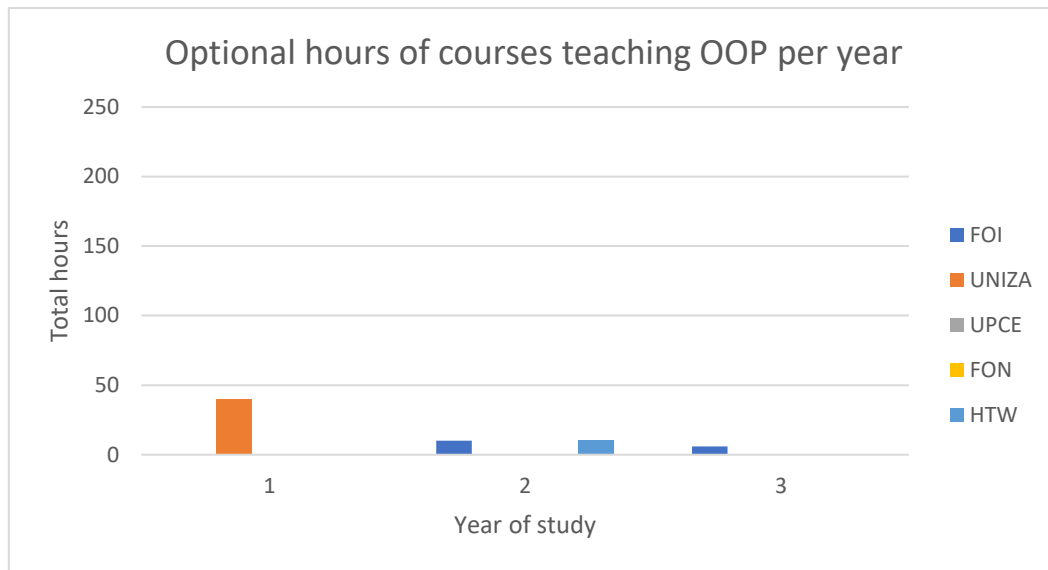


Chart 24 - Comparison of total hours of teaching OOP of optional courses between all universities in every year

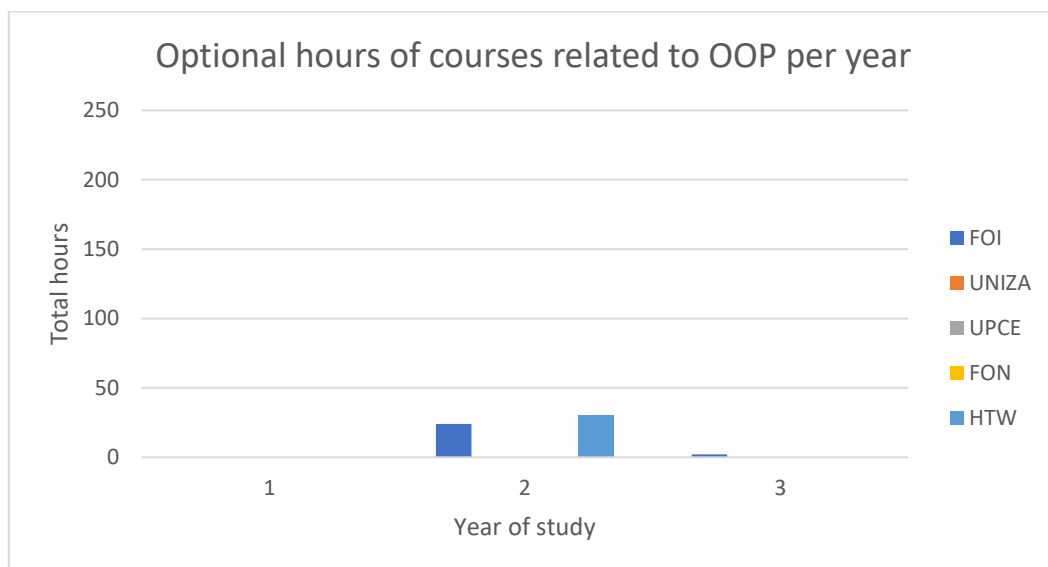


Chart 25 - Comparison of total hours related to OOP of optional courses between all universities in every year

We can conclude that FOI, UNIZA, UPCE and HTW invest in the first year of study significant number of hours in mandatory subject to teach OOP. In the second year, the teaching of OOP is supported with the subjects related to OOP. Different approach can be observed in FON, that starts teaching of OOP in the second year of study. Regarding to UPCE, the OOP topics have a significant number of hours also in second year of study. According to data, we can conclude that every university starts teaching OOP in the beginning of bachelor's study. If suitable OOP background would be covered in high schools:

- students will not have to change the way of thinking between procedural and object oriented programming.
- universities may focus on more advanced concepts of OOP, what may lead to better understanding of such concepts from students.

As a second comparative analysis we compared sum of values $TH_{\sigma}^{teaching}$ of OOP teaching and OOP practicing categories. For this analysis only the hours of teaching OOP were considered since these are where the basics of OOP are covered. Respective charts are depicted below:

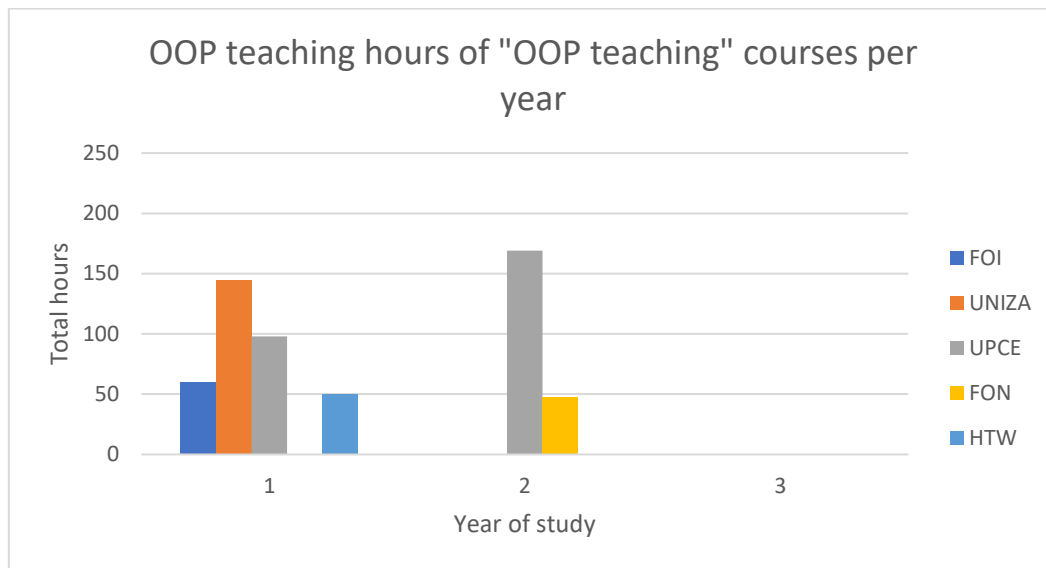


Chart 26 - Comparison of total hours of teaching OOP of OOP teaching courses between all universities in every year

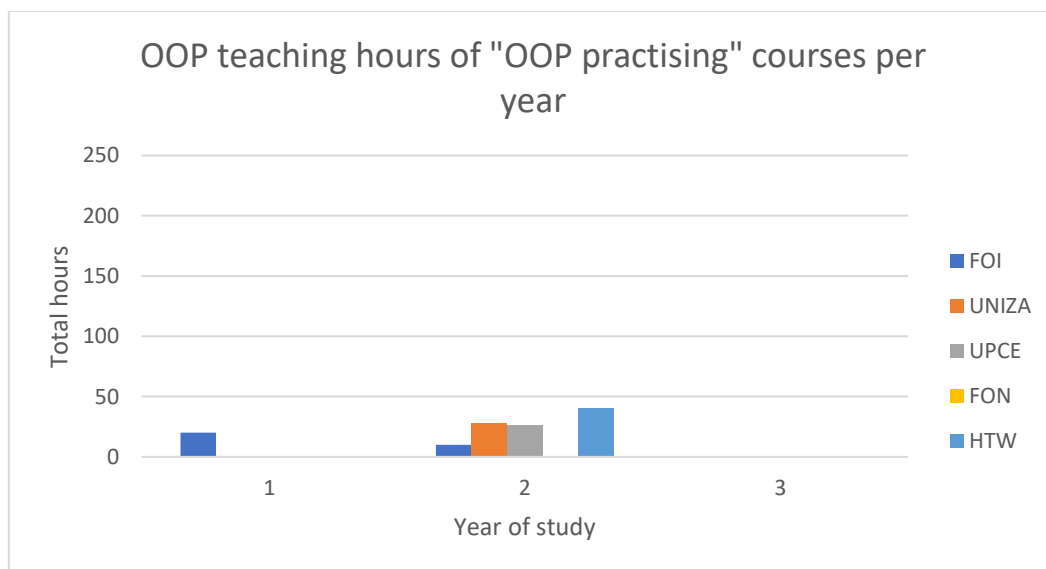


Chart 27 - Comparison of total hours of teaching OOP of OOP practicing courses between all universities in every year

From this analysis we may conclude that following proposed categorization the first years of study are dedicated to the fundamentals of OOP, while later years of study are more practical oriented. This supports the conclusion from first comparative analysis, that universities invest significant number of hours in the first OOP oriented courses.

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.

Table 22 - OOP teaching courses considered in the prior requirements analysis

University	Subject name	Type of subject	Year
FOI	Object-oriented programming	Mandatory	1
HTW	Programming II	Mandatory	1
FON	Programming 2	Mandatory	2
UNIZA	Informatics 1	Mandatory	1
UNIZA	Informatics 2	Mandatory	1
UNIZA	Practice of programming 1	Optional	1
UNIZA	Practice of programming 2	Optional	1
UPCE	Basics of Algorithmization	Mandatory	1
UPCE	Algorithmization and programming practicum	Mandatory	1
UPCE	Basics of Programming Using Java Programming Language	Mandatory	1
UPCE	Object Oriented Programming	Mandatory	2
UPCE	Language C++ I	Mandatory	2
UPCE	The C# Programming Language	Mandatory	2

1.2.1.1. FOI

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

Table 23 - 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 Structural programming Data structures Code comprehension Use of IDE

1.2.1.2. HTW

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

 Table 24 - Prior requirements analysis of course *Programming II on HTW*

Subject name	Programming II
Type of subject	Mandatory
Year	1
Prior knowledge	Knowledge of Programming in C (builds on <i>Programming I</i>)
Learning outcome of <i>Programming I</i>	Ability to implement algorithms in an imperative programming language (C), use of program development tools, testing of programs
Prior skills	Working with development tools, editor, compiler, debugger
Areas of prior requirements	Algorithmization Programming language C Use of IDE

1.2.1.3. FON

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

Table 25 - Prior requirements analysis of course Programming 2 on FON

Subject name	Programming 2
Type of subject	Mandatory
Year	2
Prior knowledge	Competence of students to develop software using the method of functional decomposition and structured programming in an imperative language.
Prior skills	Active usage of IDE components (editor, debugger,...)
Areas of prior requirements	Algorithmization Structural programming Use of IDE

1.2.1.4. UNIZA

On UNIZA, there are two mandatory and two optional OOP teaching courses in the first year of study analyzed.

Table 26 - Prior requirements analysis of course Informatics 1 on UNIZA

Subject name	Informatics 1
Type of subject	Mandatory
Year	1
Prior knowledge	No prior prerequisites are specified. The course is in the first semester of the first year of study. Strategy of faculty is to provide the education in informatics for any students including students with no prior knowledge of programming. An additional course for students without programming knowledge is a Practice of programming 1.
Prior skills	Basic usage of PC (turn on/off, install IDE, pack/unpack files, browse web, work with word processor).
Areas of prior requirements	Use of PC

Table 27 - Prior requirements analysis of course Practice of programming 1 on UNIZA

Subject name	Practice of programming 1
Type of subject	Optional
Year	1
Prior knowledge	No prior prerequisites are specified. The course is in the first semester of the first year of study. Strategy of faculty is to provide the education in informatics for any students including students with no prior knowledge of programming. This is reflected in the course Informatics 1. Practice of programming is supplementary course, so no special requirements beside the very basic of computer usage are required.
Prior skills	Basic usage of PC (turn on/off, install IDE, pack/unpack files, browse web, work with word processor), no programming skills are required.
Areas of prior requirements	Use of PC

Table 28 - Prior requirements analysis of course Informatics 2 on UNIZA

Subject name	Informatics 2
Type of subject	Mandatory
Year	1
Prior knowledge	Knowledge of Informatics 1 are required. It is not required to absolve Practice of programming 1.
Learning outcome of Informatics 1	After the course student: <ul style="list-style-type: none"> - is able to construct algorithms using all basic construction elements, - correctly applies the principles of object-oriented programming, - creates complex object-oriented programs, - integrates the proposed algorithms into the designed object structure
Prior skills	Basic usage of PC (turn on/off, install IDE, pack/unpack files, browse web, work with word processor).
Areas of prior requirements	Algorithmization Object programming Use of PC

Table 29 - Prior requirements analysis of course Practice of programming 2 on UNIZA

Subject name	Practice of programming 2
Type of subject	Optional
Year	1
Prior knowledge	Knowledge of Informatics 1 are required. It is not required to absolve Practice of programming 1.
Learning outcome of Informatics 1	After the course student: - is able to construct algorithms using all basic construction elements, - correctly applies the principles of object-oriented programming, - creates complex object-oriented programs, - integrates the proposed algorithms into the designed object structure
Prior skills	Basic usage of PC (turn on/off, install IDE, pack/unpack files, browse web, work with word processor).
Areas of prior requirements	Algorithmization Object programming Use of PC

1.2.1.5. UPCE

On UPCE, there are six mandatory OOP teaching courses analyzed. Three of them take part in first year of study, three of them in the second year of study.

Table 30 - Prior requirements analysis of course Basics of Algorithmization on UPCE

Subject name	Basics of Algorithmization
Type of subject	Mandatory
Year	1
Prior knowledge	Only basic knowledge of mathematics on the level of a secondary school is necessary.
Prior skills	Only basic usage of PC (turn on/off, install IDE, pack/unpack files, browse web).
Areas of prior requirements	Mathematics HS. Use of PC

Table 31 - Prior requirements analysis of course Algorithmization and programming practicum on UPCE

Subject name	Algorithmization and programming practicum
Type of subject	Mandatory
Year	1
Prior knowledge	Subject matter of the subject Introduction to Algorithms.
Prior skills	Only basic usage of PC (turn on/off, install IDE, pack/unpack files, browse web).
Areas of prior requirements	Use of PC

Table 32 - Prior requirements analysis of course *Basics of Programming Using Java Programming Language* on UPCE

Subject name	Basics of Programming Using Java Programming Language
Type of subject	Mandatory
Year	1
Prior knowledge	A prerequisite for mastering this course is successful completion of the course "Basics of Algorithmization"
Learning outcome of <i>Basics of Algorithmization</i>	The aim of this course is to make students familiar with the basics of algorithmization, algorithmic way of thinking and preparing students for learning basics of programming. Improving of analytical and logical thinking . Ability to create and write algorithms
Prior skills	Ability to create and write algorithms.
Areas of prior requirements	Algorithmization Code comprehension Use of PC

Table 33 - Prior requirements analysis of course Object Oriented Programming on UPCE

Subject name	Object Oriented Programming
Type of subject	Mandatory
Year	2
Prior knowledge	A prerequisite for mastering this course is successful completion of the course "Basics of Programming Using Java Programming Language"
Learning outcome of <i>Basics of Programming Using Java Programming Language</i>	<p>Students learn to analyze basic tasks in the area of programming and then to implement them using Java programming language.</p> <p>Student will be able to implement a simple algorithms using the Java programming language.</p>
Prior skills	Ability to create and write algorithms in Java language, basic knowledge of Object, Classes.
Areas of prior requirements	<p>Algorithmization</p> <p>Code comprehension</p> <p>Object programming</p> <p>Programming language Java</p>

Table 34 - Prior requirements analysis of course Language C++ I on UPCE

Subject name	Language C++ I
Type of subject	Mandatory
Year	2
Prior knowledge	Successful completion of the course Introduction to C language. A prerequisite is basic knowledge from the field of algorithmization and object oriented programming.
Learning outcome of <i>Language C</i>	To learn basic programming techniques of structured programming in C language. After taking the course, the student has basic skills in C language programming.
Prior skills	Ability to create and write algorithms in C language, knowledge of Object, Classes
Areas of prior requirements	Algorithmization Code comprehension Structural programming Object programming Programming language C

Table 35 - Prior requirements analysis of course The C# Programming Language on UPCE

Subject name	The C# Programming Language
Type of subject	Mandatory
Year	2
Prior knowledge	Successful completion of the course Introduction to C language. A prerequisite is basic knowledge from the field of algorithmization and object oriented programming, data structures and C++ programming
Learning outcome of Language C	To learn basic programming techniques of structured programming in C language. After taking the course, the student has basic skills in C language programming.
Prior skills	Ability to create and write algorithms in C language, knowledge of Object, Classes
Areas of prior requirements	Algorithmization Code comprehension Structural programming Object programming Data structures Programming language C

1.2.1.6. Prior requirements matrix

To focus only on relevant data, we present following matrix of prior requirements for OOP teaching courses.

Table 36 - Prior requirements matrix of OOP teaching courses

	Object-oriented programming	Programming II	Programming 2	Informatics 1	Informatics 2	Practice of programming 1	Practice of programming2	Basics of Algorithmization	Algorithmization and programming practicum	Basics of Programming Using Java Programming Language	Object Oriented Programming	Language C++ I	The C# Programming Language
Algorithmization	x	x	x		x		x			x	x	x	x
Structural programming	x		x									x	x
Object programming					x		x				x	x	x
Data structures	x												x
Programming language Java											x		
Programming language C		x										x	x
Mathematics HS								x					
Code comprehension	x									x	x	x	x
Use of IDE	x	x	x										
Use of PC				x	x	x	x	x	x	x			

Requirements distribution of OOP teaching courses is summarized in following table. We highlighted top third of most important requirements:

Table 37 - Distribution of requirements of OOP teaching courses

Requirement	Required by	%
Algorithmization	9	69%
Use of PC	7	54%
Object programming	5	38%
Code comprehension	5	38%
Structural programming	4	31%
Programming language C	3	23%
Use of IDE	3	23%
Data structures	2	15%
Programming language Java	1	8%
Mathematics HS	1	8%

OOP teaching courses take place in first years of study. One has to realize that previous step of education is high school. Moreover, we deem listed requirements to be the most important, because they are required by OOP teaching courses. Recall that these courses are primarily focused on teaching new concepts. Focus on these competencies on high school is therefore very important for successful university study in informatics related study programs.

1.2.2. OOP practicing courses

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

Table 38 - 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
HTW	Software Engineering 2	Mandatory	2
HTW	Programming of component architectures	Optional	2
UNIZA	Informatics 3	Mandatory	2
UNIZA	Algorithms and Data structures 1	Mandatory	2
UPCE	Data Structures	Mandatory	2

1.2.2.1. FOI

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 39 - 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 <i>Object-oriented programming</i>	After the course student is able to: <ul style="list-style-type: none"> - Design an efficient software solution for a given algorithmic problem - Create models of the software solution using standard UML diagrams - Organize data in an efficient way for a given algorithmic problem. - Create models of data structures using standard UML diagrams - 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 Object programming UML Code comprehension Use of IDE

Table 40 - 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	<ul style="list-style-type: none"> Algorithmization Structural programming Data structures Use of IDE

Table 41 - 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 <i>Object-oriented programming</i>	After the course student is able to: <ul style="list-style-type: none"> - Design an efficient software solution for a given algorithmic problem - Create models of the software solution using standard UML diagrams - Organize data in an efficient way for a given algorithmic problem. - Create models of data structures using standard UML diagrams - 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 taught but are the base for the other topics included.
Areas of prior requirements	<ul style="list-style-type: none"> Algorithmization Structural programming Data structures Mathematics UNI Code comprehension Use of IDE

1.2.2.2. HTW

On HTW, there is one mandatory and one optional OOP practicing course in the second year of study analyzed.

Table 42 - Prior requirements analysis of course Software Engineering 2 on HTW

Subject name	Software Engineering 2
Type of subject	Mandatory
Year	2
Prior knowledge	Object-oriented programming languages (C++ or JAVA)
Prior skills	Fundamental programming skills in procedural and object-oriented styles
Areas of prior requirements	Structural programming Object programming Programming language Java Programming language C++

Table 43 - Prior requirements analysis of course Programming of component architectures on HTW

Subject name	Programming of component architectures
Type of subject	Optional
Year	2
Prior knowledge	Object-oriented programming languages (C++ or JAVA)
Prior skills	Sophisticated programming skills, fundamentals of software architecture and computer networks
Areas of prior requirements	Programming language Java Programming language C++ Sophisticated programming Software architecture Computer networks

1.2.2.3. FON

There are no OOP practicing courses to be analyzed on FON.

1.2.2.4. UNIZA

On UNIZA, there are two mandatory OOP practicing courses in the second year of study analyzed.

Table 44 - Prior requirements analysis of course Informatics 3 on UNIZA

Subject name	Informatics 3
Type of subject	Mandatory
Year	2
Prior knowledge	Students should have knowledge of Informatics 1 and 2
Learning outcome of Informatics 1	After the course student is able to: <ul style="list-style-type: none"> - Design an efficient software solution for a given algorithmic problem - Create models of the software solution using standard UML diagrams - Organize data in an efficient way for a given algorithmic problem. - Create models of data structures using standard UML diagrams - Develop a software solution for a given algorithmic problem using object-oriented programming language.
Learning outcome of Informatics 2	After the course student: <ul style="list-style-type: none"> - uses the principle of polymorphism in the design of object structure and algorithms, - clearly organizes the structure of projects using packages, - resolves program error states, - uses advanced principles in design of the object structure, such as inheritance and generics, - creates a simple user interface
Prior skills	Basic knowledge of an IDE software for source code editing.
Areas of prior requirements	Algorithmization Object programming Use of IDE

Table 45 - Prior requirements analysis of course Algorithms and Data structures 1 on UNIZA

Subject name	Algorithms and Data structures 1
Type of subject	Mandatory
Year	2
Prior knowledge	Informatics 2 Informatics 3
Learning outcome of <i>Informatics 2</i>	After the course student: <ul style="list-style-type: none"> - uses the principle of polymorphism in the design of object structure and algorithms, - clearly organizes the structure of projects using packages, - resolves program error states, - uses advanced principles in design of the object structure, such as inheritance and generics, - creates a simple user interface
Learning outcome of <i>Informatics 3</i>	After completing the course, student will be able to: <ul style="list-style-type: none"> - analyze and create solutions using procedural and object-oriented approach, - create basic applications using C and C++ languages, - create and debug applications using MS Visual Studio IDE.
Prior skills	<ul style="list-style-type: none"> - Analysis of given problem, focus on what is important. - Skill with some UML modelling tool with focus on class diagrams. - To create semestral project skills in object oriented language with low memory management (C++). - To create documentation: skill with any word processor (MS Word, OpenOffice Writer, LaTeX..). - Skills in any table processor in order to analyze input data and plot functions based on them. - Ability to formulate and present conclusions according to data analysis.
Areas of prior requirements	Sophisticated programming Programming language C++ UML Use of IDE Use of PC

1.2.2.5. UPCE

On UPCE, there is one mandatory OOP practicing course in the second year of study analyzed.

Table 46 - Prior requirements analysis of course Data Structures on UPCE

Subject name	Data Structures
Type of subject	Mandatory
Year	2
Prior knowledge	A prerequisite is basic knowledge from the field of algorithmization and object oriented programming.
Prior skills	There is expected elementary knowledge from the field of algorithmic techniques and object-oriented programming.
Areas of prior requirements	Algorithmization Object programming

1.2.2.6. Prior requirements matrix

To focus only on relevant data, we present following matrix of prior requirements for OOP practicing courses.

Table 47 - Prior requirements matrix of OOP practicing courses

	Windows Applications Development	Programming 2	Mobile applications and games development	Software Engineering 2	Programming of component architectures	Informatics 3	Algorithms and Data structures 1	Data Structures
Algorithmization	x	x	x			x		x
Structural programming		x	x	x				
Object programming	x			x		x		x
Sophisticated programming					x		x	
UML	x						x	
Code comprehension	x							
Use of IDE	x	x	x			x	x	
Use of PC							x	
Data structures			x					
Mathematics UNI			x					
Code comprehension			x					
Programming language Java				x	x			
Programming language C++				x	x		x	
Software architecture					x			
Computer networks					x			

Requirements distribution of OOP practicing courses is summarized in following table. We highlighted top third of most important requirements:

Table 48 - Distribution of requirements of OOP practicing courses

Requirement	Required by	%
Algorithmization	5	63%
Use of IDE	5	63%
Object programming	4	50%
Structural programming	3	38%
Programming language C++	3	38%
Sophisticated programming	2	25%
UML	2	25%
Programming language Java	2	25%
Code comprehension	1	13%
Use of PC	1	13%
Data structures	1	13%
Mathematics UNI	1	13%
Code comprehension	1	13%
Software architecture	1	13%
Computer networks	1	13%

OOP practicing courses mostly take place in second years of study. They build upon the OOP teaching courses. One can see bigger variety in prior requirements (there can be clearly observed movement towards particular languages/technologies and sophisticated programming), however the algorithmization is still key competence of students. Note, that three out of four requirements are shared with OOP teaching courses, what justifies the importance of proper outcomes that should be provided by high schools.

1.2.3. OOP using courses

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

Table 49 - OOP using courses considered in the prior requirements analysis

University	Subject name	Type of subject	Year
FOI	Programming in Python	Optional	3
HTW	Programming I	Mandatory	1
HTW	Programming distributed systems	Optional	2
FON	Data structures and algorithms	Mandatory	2

1.2.3.1. FOI

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

Table 50 - 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 <i>Object-oriented programming</i>	After the course student is able to: <ul style="list-style-type: none"> - Design an efficient software solution for a given algorithmic problem - Create models of the software solution using standard UML diagrams - Organize data in an efficient way for a given algorithmic problem. - Create models of data structures using standard UML diagrams - 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 Object programming UML Code comprehension Use of IDE

1.2.3.2. HTW

On HTW, there is one mandatory and one optional OOP practicing course in the first and second year of study analyzed.

Table 51 - Prior requirements analysis of course Programming I on HTW

Subject name	Programming I
Type of subject	Mandatory
Year	1
Prior knowledge	None
Prior skills	Basic handling of computers, file editing
Areas of prior requirements	Use of PC

Table 52 - Prior requirements analysis of course Programming distributed systems on HTW

Subject name	Programming distributed systems
Type of subject	Optional
Year	2
Prior knowledge	Object-oriented programming languages (C++ or JAVA)
Prior skills	Sophisticated programming skills, fundamentals of software architecture and computer networks
Areas of prior requirements	Programming language Java Programming language C++ Sophisticated programming Software architecture Computer networks

1.2.3.3. FON

On FON, there is one mandatory OOP using course in the second year of study analyzed.

Table 53 - Prior requirements analysis of course Data structures and algorithms on FON

Subject name	Data structures and algorithms
Type of subject	Mandatory
Year	2
Prior knowledge	Basic knowledge of Java
Prior skills	Active usage of IDE components (editor, debugger,...)
Areas of prior requirements	Programming language Java Use of IDE

1.2.3.4. UNIZA

There are no OOP practicing courses to be analyzed on UNIZA.

1.2.3.5. UPCE

There are no OOP practicing courses to be analyzed on UPCE.

1.2.3.6. Prior requirements matrix

To focus only on relevant data, we present following matrix of prior requirements for OOP using courses.

Table 54 - Prior requirements matrix of OOP using courses

	Programming in Python	Programming 1	Programming distributed systems	Data structures and algorithms
Algorithmization	x			
Object programming	x			
Sophisticated programming			x	
UML	x			
Code comprehension	x			
Use of IDE	x			x
Use of PC		x		
Programming language Java			x	x
Programming language C++			x	
Software architecture			x	
Computer networks			x	

Requirements distribution of OOP using courses is summarized in following table. We highlighted the most different requirements.

Table 55 - Distribution of requirements of OOP using courses

Requirement	Required by	%
Use of IDE	2	50%
Programming language Java	2	50%
Algorithmization	1	25%
Object programming	1	25%
Sophisticated programming	1	25%
UML	1	25%
Code comprehension	1	25%
Use of PC	1	25%
Programming language C++	1	25%
Software architecture	1	25%
Computer networks	1	25%

There can not be observed any pattern of occurrence in OOP using courses. These courses build upon the OOP teaching and practicing courses. Even more variety in prior requirements than in OOP practicing courses can be seen. Based on very small difference between requirements and because of the small number of courses in this category, we should not make any conclusion regarding the requirements. However one can observe that the requirement of algorithmization is present also in this group of subjects.

1.2.4. Conclusion

To identify the most relevant requirements we took most important requirements of different type of courses (see tables

Table 37, Table 48 and Table 55). Then we computed the distribution of every requirement among all courses and sorted them. The processed data are summarized in following table. Again, we highlighted the most different requirements.

Table 56 - Distribution of requirements of all OOP courses

Requirement	Teaching		Practicing		Using		Total	
	out of 13	%	out of 8	%	out of 4	%	out of 25	%
Algorithmization	9	69	5	63	1	25	15	60
Use of IDE	3	23	5	63	2	50	10	40
Use of PC	7	54	1	13	1	25	9	36
Object programming	5	38	4	50	1	25	9	36
Structural programming	4	31	3	38	-	-	7	28
Code comprehension	5	38	1	13	1	25	6	24
Programming language Java	1	8	2	25	2	50	5	20
Programming language C++	-	-	3	38	1	25	4	16

Regarding performed analysis the most required prior skills, which must be properly integrated into the new concept of teaching, are:

- *Algorithmization.*
- *Use of IDE.*
- *Use of PC.*

We can compare prior requirements to learning outcomes from lower levels of education. In Slovakia, programming is taught in primary and secondary schools. It is also based on the innovative State Educational Program, where various programming topics are incorporated into the Computer science subject. Regarding algorithmization, these are, for example [1]:

- identification of relationships between information (input - output),
- writing the algorithm and executing the program,
- statements, parameters and sequencing,
- variables and mathematical operations with numbers (addition, subtraction, multiplication, division),
- loops,
- conditional statements,
- debugging.

In February 2022, a survey was conducted in Slovakia focused on "the professional competences and attitudes of computer science teachers in the field of programming about teaching programming at elementary and secondary schools". With a closer focus on secondary schools, we found the following [2]:

- 45.1% of secondary school teachers have less than 5 years of experience teaching programming,
- 19% of secondary school teachers are not qualified to teach informatics (including programming),

- teachers teach the most common programming languages Python, Imagine Logo, Scratch, C/C++/ C#.
- the Java programming language is taught in 5.2% of secondary schools,
- 9.1% of teachers do not know any programming language,
- most programming languages are taught by university-qualified teachers,
- OOP is taught by 27.5% of teachers,
- OOP is not mastered by 47.5% of secondary school teachers.

Results of this survey imply disproportion with higher education prior requirements. However, teachers have positive attitudes towards programming, and compared to surveys from 2019 and 2022, the number of qualified computer science teachers has increased, which can have a positive impact on teaching programming.

Another survey concerned university students concludes [3]:

- 52% of students are satisfied with the computer science they learned in high school,
- only 46% of students said that computer science prepared them for university studies,
- 64.5% of students said that they lacked programming skills in secondary school computer science with regard to the needs of university studies.

Literature review performed by Qian and Lehman [4] summarizes difficulties, related factors and potential strategies to address them. Authors declare that learners' prior knowledge plays an important role in forming misconceptions. They imply that according to conceptual change theories, learners' prior knowledge plays an important role in forming misconceptions. They divide knowledge into three areas – syntactic, conceptual and strategic. Among other strategies and tools to overcome difficulties, authors present following strategies addressing them:

- Advanced editors or graphical programming environments can highlight or prevent syntax errors, reduce cognitive load, and help students with syntactical difficulties.
- Well-chosen program examples can help students build accurate understanding of programming and improve knowledge transfer.
- Visualization tools such as Python. Tutor can help students to visualize code execution step by step and build correct mental models.
- Explicitly teaching debugging strategies and using enhanced debugging tools (e.g., providing detailed error messages) may improve students' debugging skills.

Choosing proper IDE and programming language, both meeting forementioned issues and prior requirements, is important part of this project. Batur did a literature review [5], where she focused on OOP. She found that OOP courses are usually taught with educational integrated development environment (eIDE) like Greenfoot or BlueJ. Novice programmers have to learn the concept of OOP, the syntax of Java and the usage of an eIDE. Currently a frame-based approach using Stride language is available in these eIDEs [6]. Benefits of using BlueJ summarizes Hubwieser [7]. From the paper we point out that it allows the students to work interactively with classes and objects before writing their first program, e.g. inspect the attribute values or invoke methods.

When teaching programming, it is necessary to focus on a programming language that has the so-called a low threshold (it's easy to start with) and a high ceiling (even as time passes, it still provides opportunities to create more and more complicated tasks) [8]. In connection with this, there is the concept of mediated transfer, where two methods are used:

- *hugging* - creating such connections between contexts when the teacher introduces a new educational situation that is similar to a previous one,
- *bridging* - the teacher points out the parallels between content elements and helps the process of abstraction, the student must consciously use abstraction and look for connections between two contexts.

Vygotsky's theory is based on *the zone of proximal development*. It means that the new knowledge should be closely related to the knowledge that the student has already learned. To support the acquisition of knowledge, the concept of *scaffolding* can be used, which tries to facilitate the understanding of jointly performed activities and enable them to acquire skills for solving a problem, performing a task, or achieving a goal, despite the fact that they may not have sufficient experience to handle the activities independently. The most common educational approach in the field of programming is constructivism, and supporting approaches in teaching OOP are [8]:

- creating games,
- team or pair work,
- problem-based teaching focusing on a realistic problem,
- project teaching,
- Inquiry-based education.

One of the specific methodologies that are used for OOP [9]:

1. discuss fundamental principles of object-orientation with respect for conventional thinking,
2. introduce an object concept by observing the real world,
3. acquire the class concept by abstraction of many common objects,
4. introduce instantiation after the class concept is learned,
5. illustrate subclasses by adding more details to an existing class and superclasses by finding common things among several classes,
6. (optional) discuss metaclasses to master the class completely and object concepts.

At secondary schools, conditions are created for teaching programming from the point of view of the innovative State Educational Program. Based on the survey, however, it appears that there are secondary school teachers who do not have programming knowledge and we assume that they do not even teach programming. On the other hand, many teachers have positive attitudes towards programming. Based on a survey of university students, it appears that they want to learn programming and it is necessary for their university studies. There are several programming approaches that can be used in OOP. With respect to prior requirements of universities' OOP courses, we need to focus on algorithmization and proper IDE using low threshold language. A good choice will be educational integrated development environment (eIDE) like Greenfoot or BlueJ that are most often used and offer the use of Java programming language as well as frame-based approach using Stride.

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.

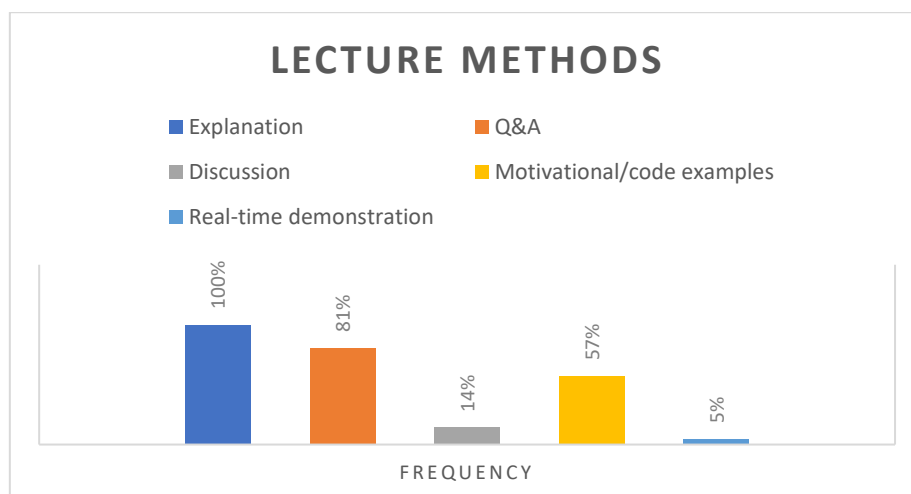


Chart 28 - Forms of knowledge transfer used in lectures

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.

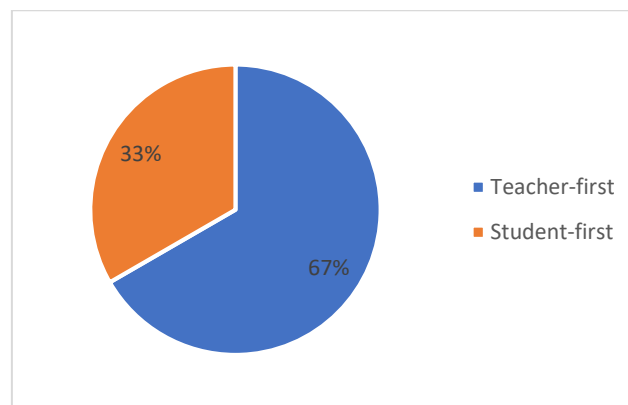


Chart 29 - Learning-by-doing approach in laboratory exercises

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.

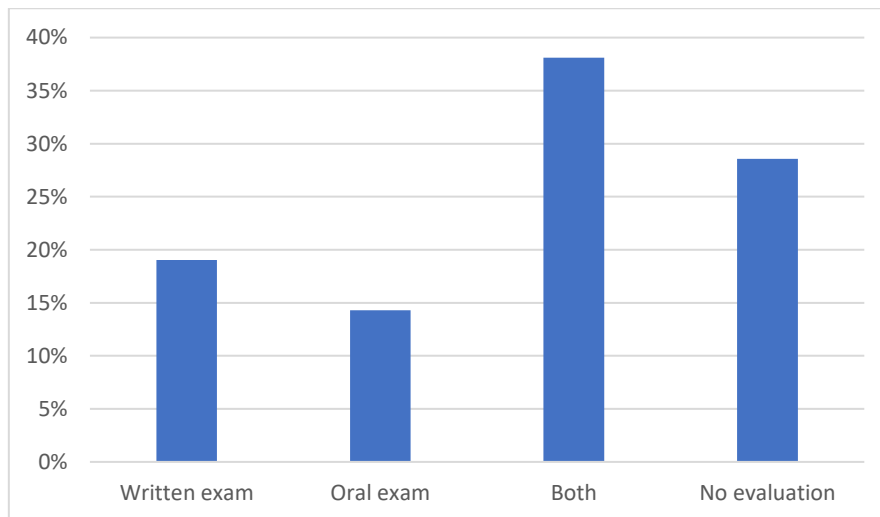


Chart 30 - Frequency of assessment methods for theoretical knowledge

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.

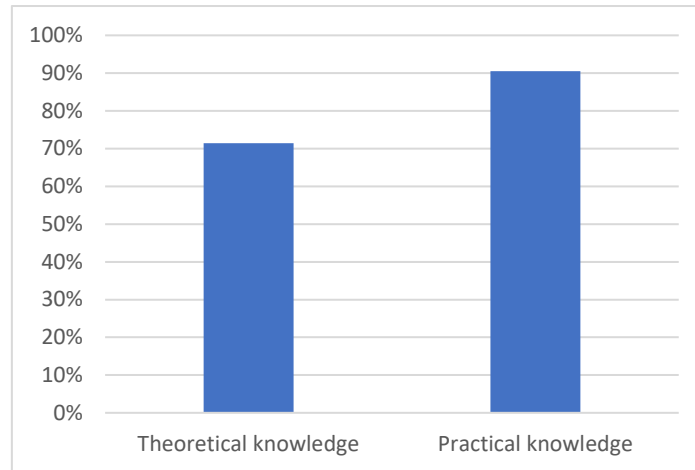


Chart 31 - Type of knowledge assessed in analyzed courses

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.

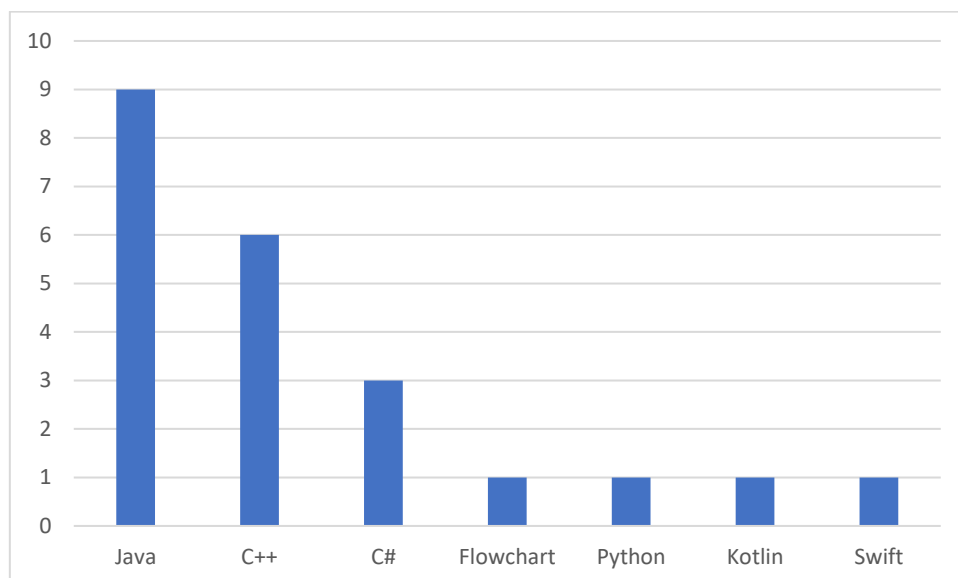


Chart 32 - Programming language occurrence in analyzed courses

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 57 - 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)	-	-	-

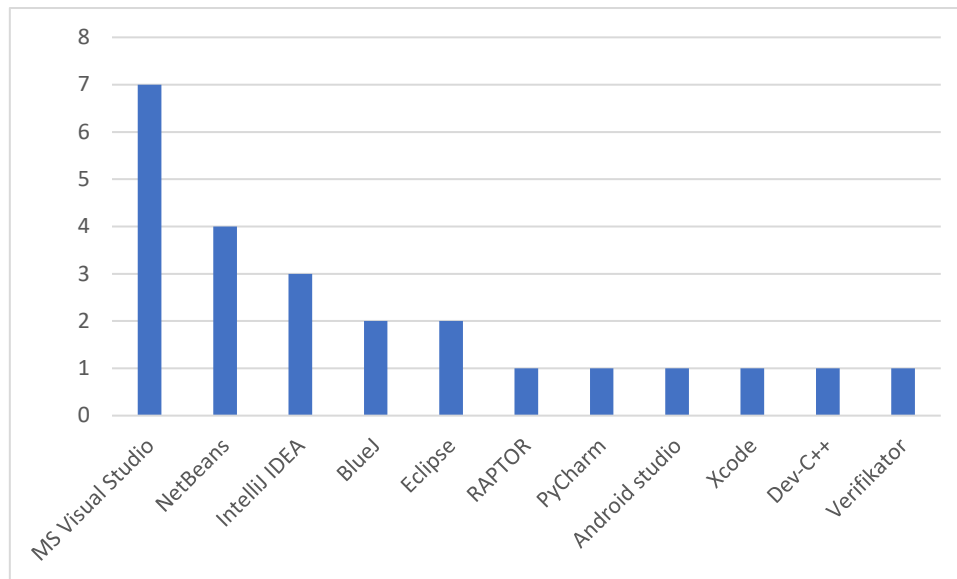


Chart 33 - Most frequently used IDEs in analyzed courses

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 acquisition, ...)
 - 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 34.

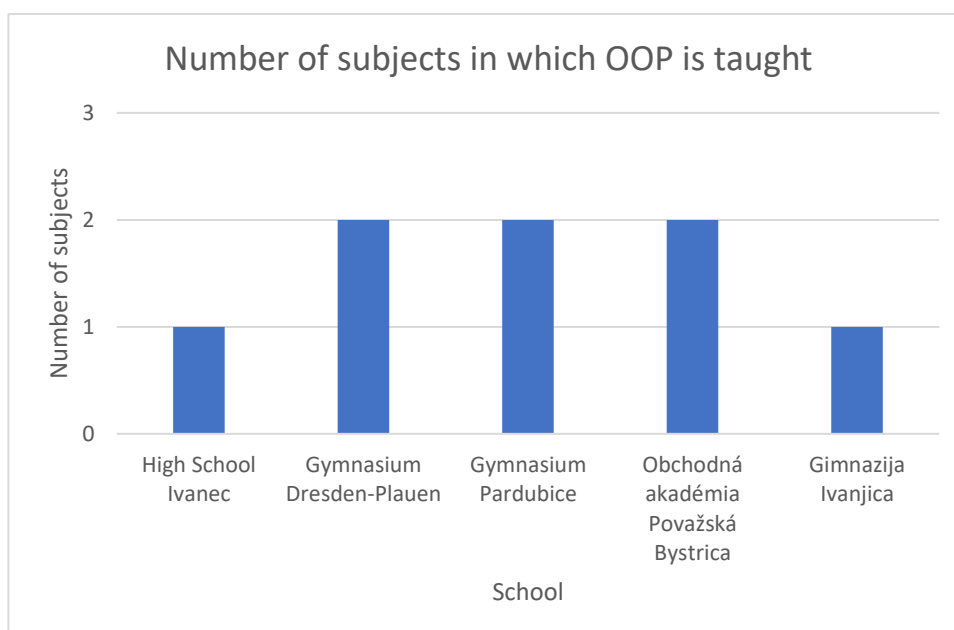


Chart 34 - High schools and numbers of OOP related subjects

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 58.

Table 58 - 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 35.

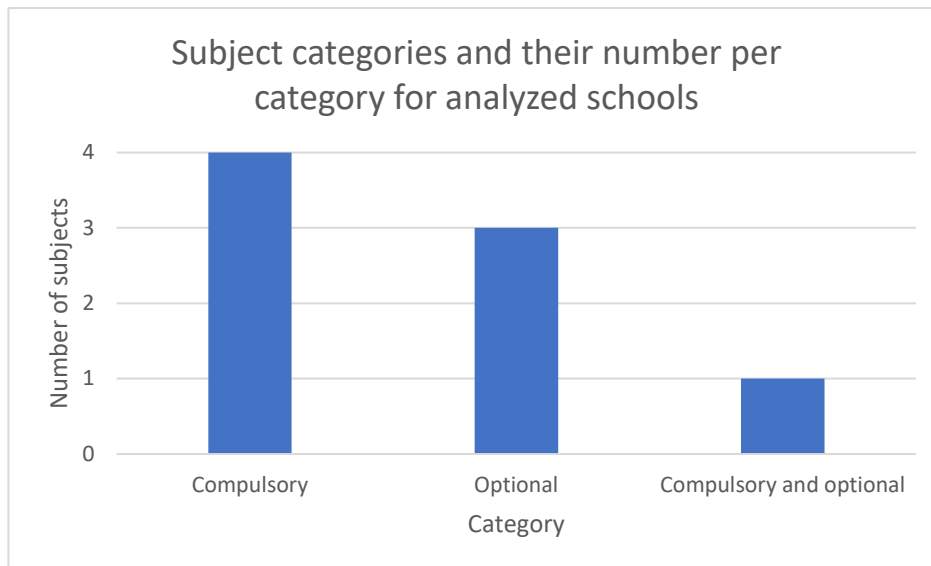


Chart 35 - Categorization of subjects and their number per category

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 36.

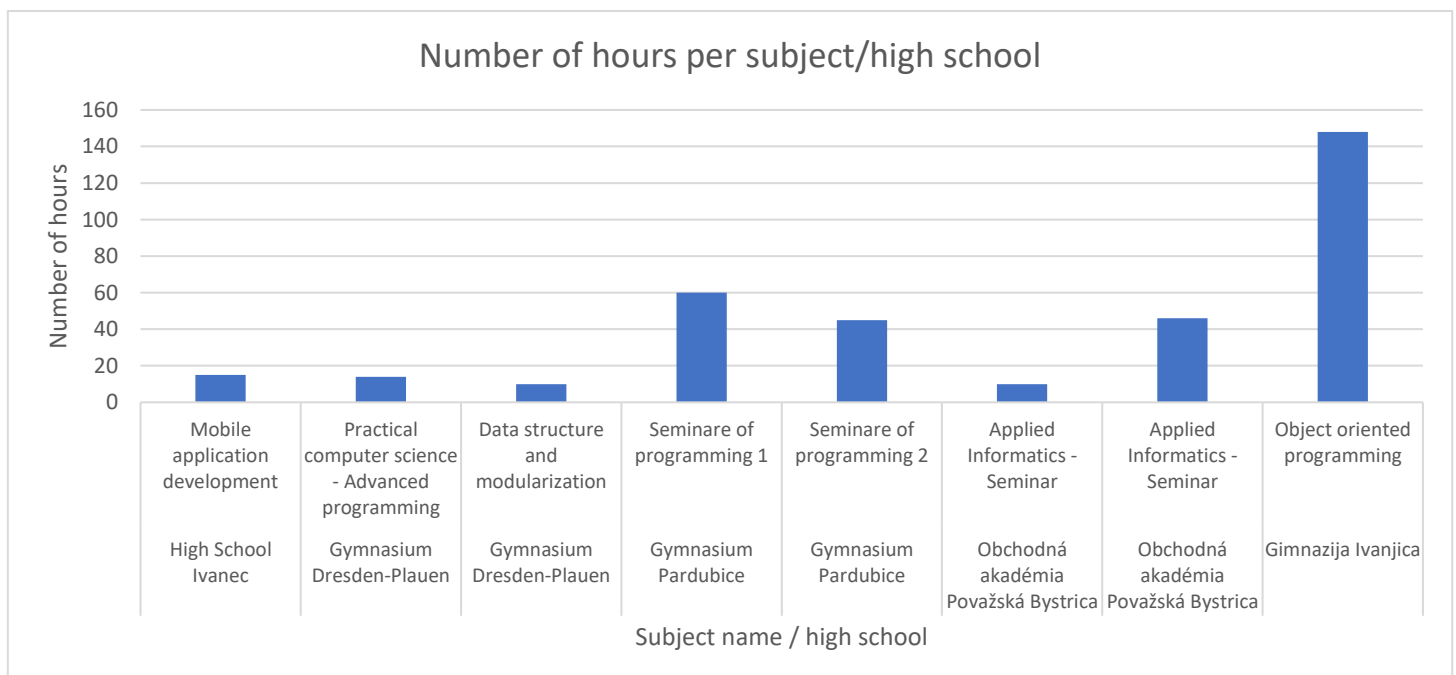


Chart 36 - Distribution of hours per subject dedicated to OOP contents

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 learning outcomes and topics and their comparison for universities and high schools

2.2.1. Analysis and comparison

In the next list, there are 24 main topics which are taught at high schools and universities:

1. classes, objects, instance
2. methods, passing methods arguments
3. constructors
4. attributes
5. method and constructor overloading
6. static variables and methods
7. packages
8. encapsulation
9. class diagram
10. association
11. inheritance
12. composition
13. send object message
14. immutable object
15. aggregation
16. abstract classes
17. polymorphism
18. interface
19. exception
20. object live cycle
21. virtual methods
22. UML
23. Generic classes
24. Nested classes

The following charts show at how many universities are each topic taught:

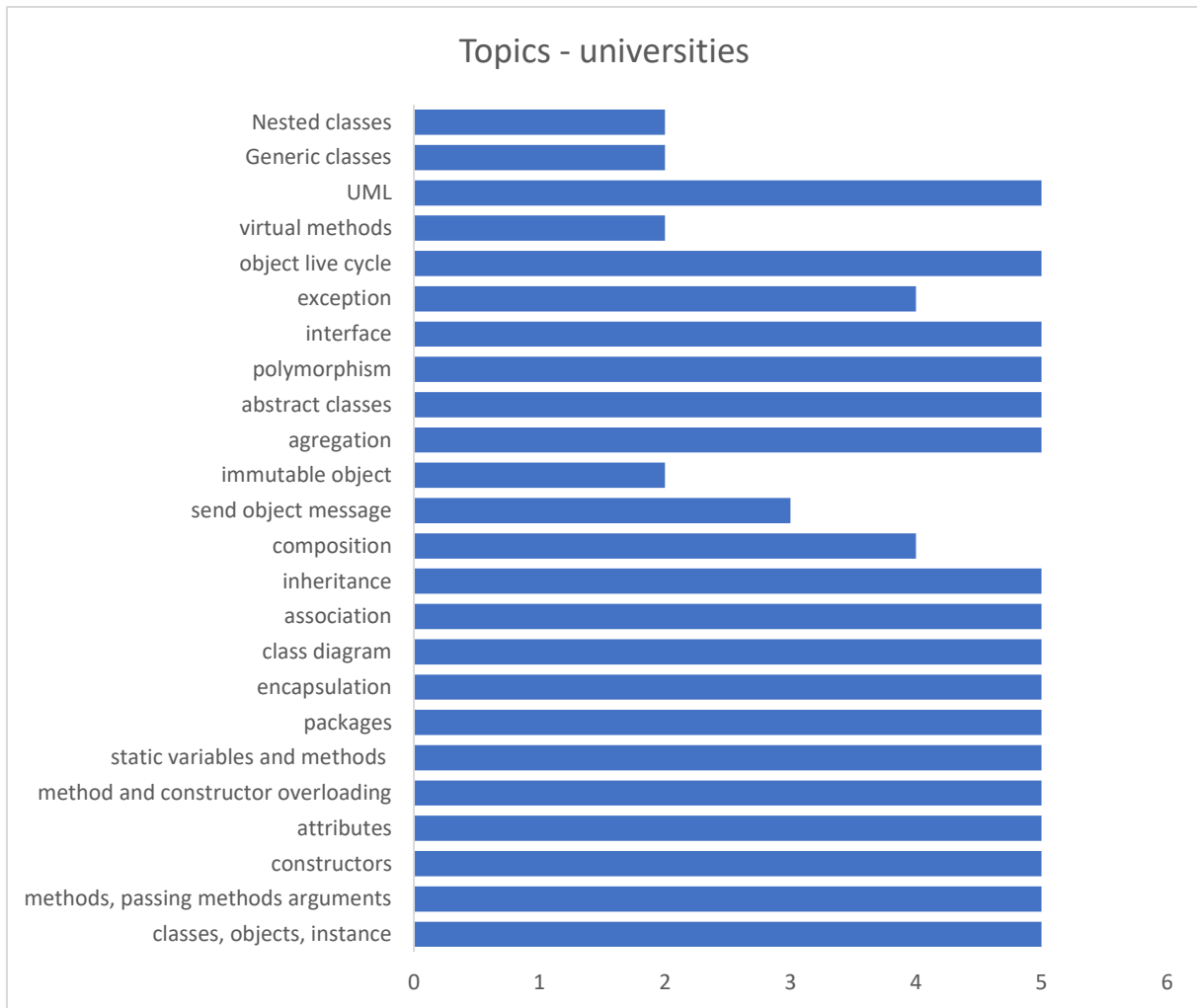


Chart 37 – Number of universities teaching each topic

Here is the list of topics that are taught at each university:

- classes, objects, instance (1)
- methods, passing methods arguments (2)
- constructors (3)
- attributes (4)
- method and constructor overloading (5)
- static variables and methods (6)
- packages (7)
- encapsulation (8)
- class diagram (9)
- association (10)
- inheritance (11)
- abstract classes (16)
- polymorphism (17)

- interface (18)
- object live cycle (20)
- UML (22)

The following charts show at how many high schools are each topic taught:

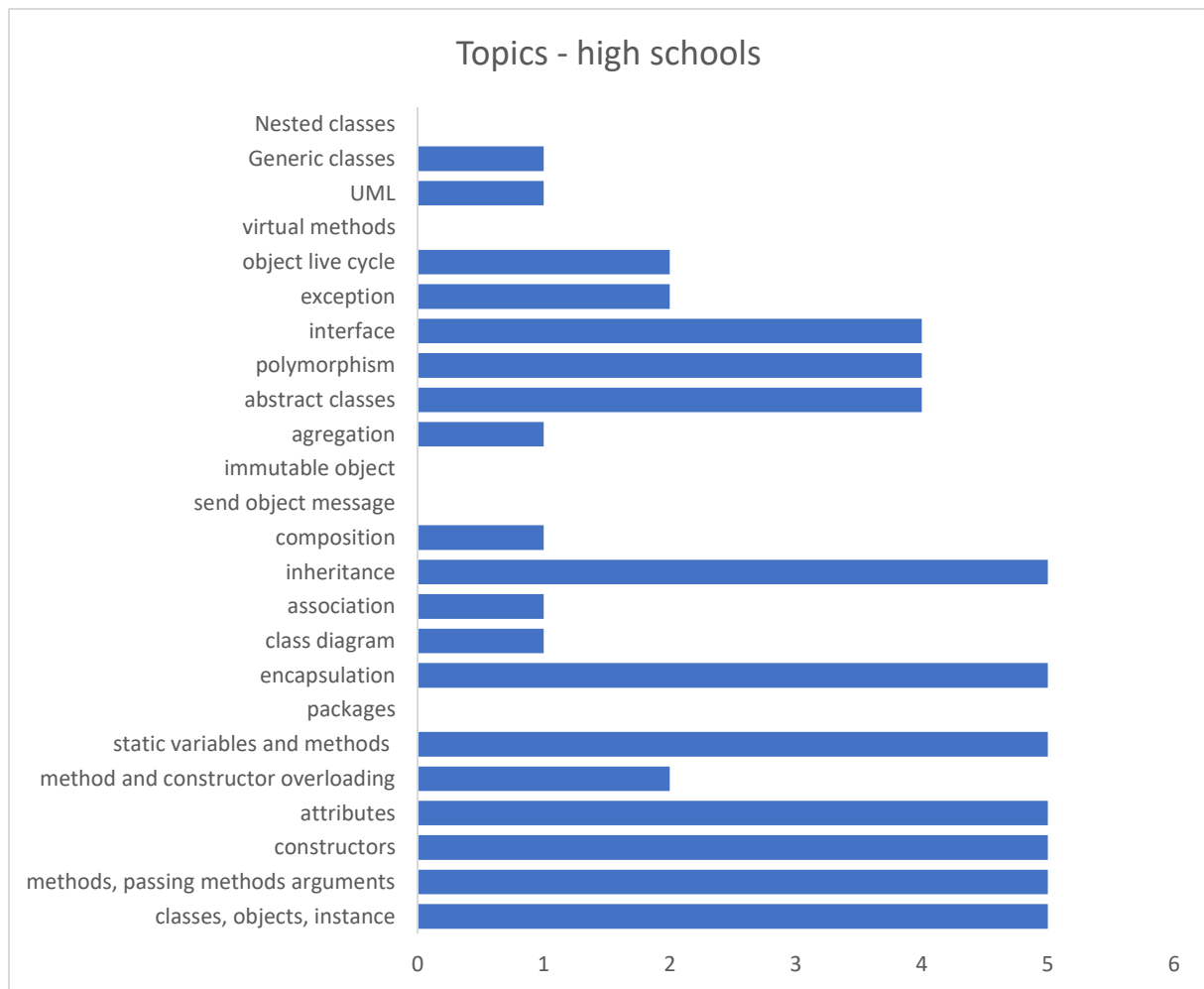


Chart 38 – Number of high schools teaching each topic

Here is a list of topics that are taught at each high school:

- classes, objects, instance (1)
- methods, passing methods arguments (2)
- constructors (3)
- attributes (4)
- static variables and methods (6)
- encapsulation (8)
- inheritance (11)

All five high schools teach OOC topics. Let us make topic analysis at each country. We will start in Czech Republic. Following chart compares topics:

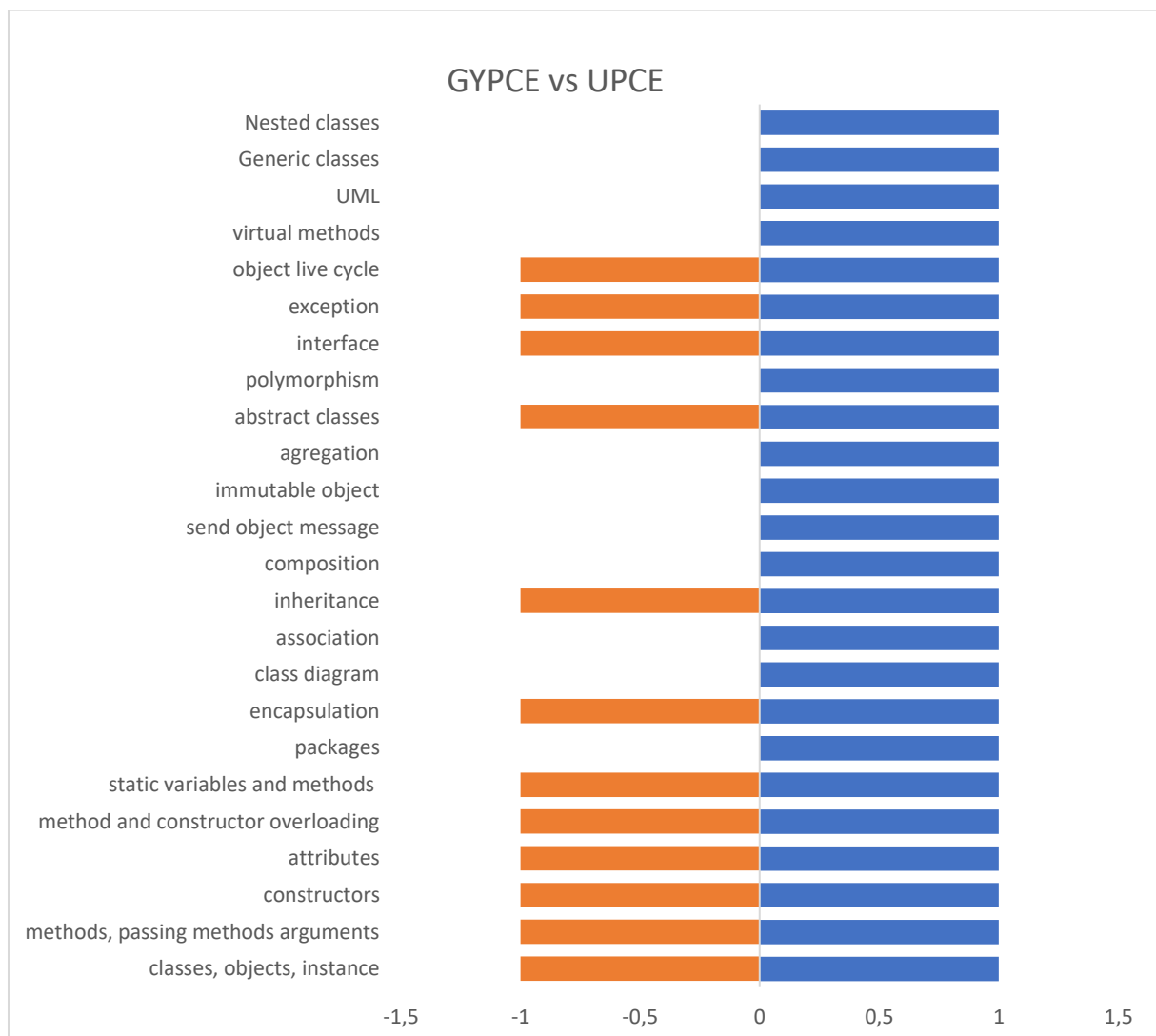


Chart 39 – Topics analysis in Czech Republic

Following topics are not in the list of all high schools' topics:

- packages (7)
- class diagram (9)
- association (10)
- composition (12)
- send object message (13)
- immutable object (14)
- aggregation (15)
- polymorphism (17)
- virtual method (21)
- UML (22)
- Generic Classes (23)
- Nested classes (24)

The next chart is displaying intersection of topics at University of Zilina and Obchodna academia Povazska Bystrica:

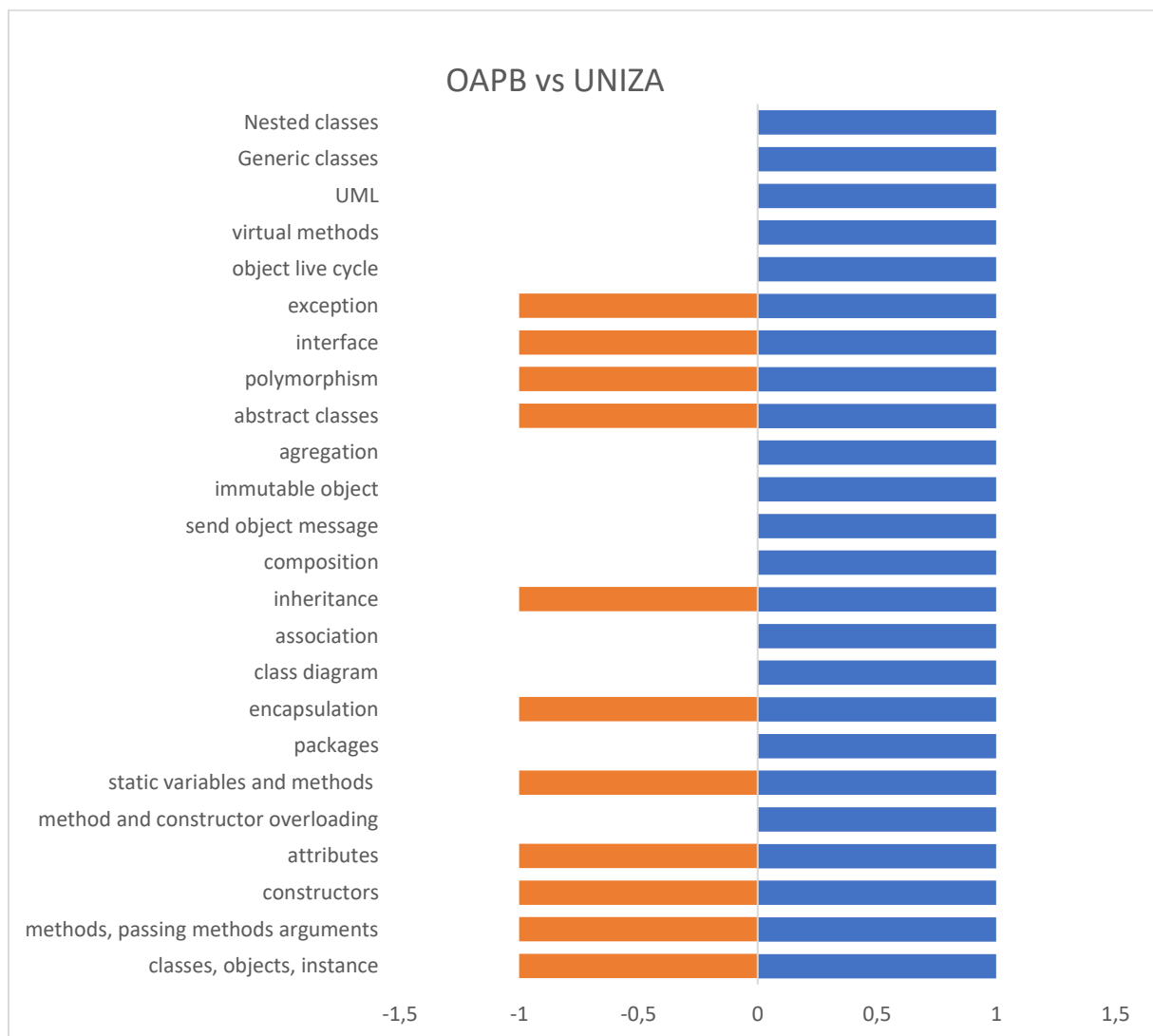


Chart 40 – Topics analysis in Slovakia

Following topics are not in the list of all high schools' topics:

- method and constructor overloading (5)
- packages (7)
- class diagram (9)
- association (10)
- composition (12)
- send object message (13)
- immutable object (14)
- aggregation (15)
- object life cycle (20)
- virtual method (21)
- UML (22)

- Generic Classes (23)
- Nested classes (24)

Next chart will show Germany comparison:

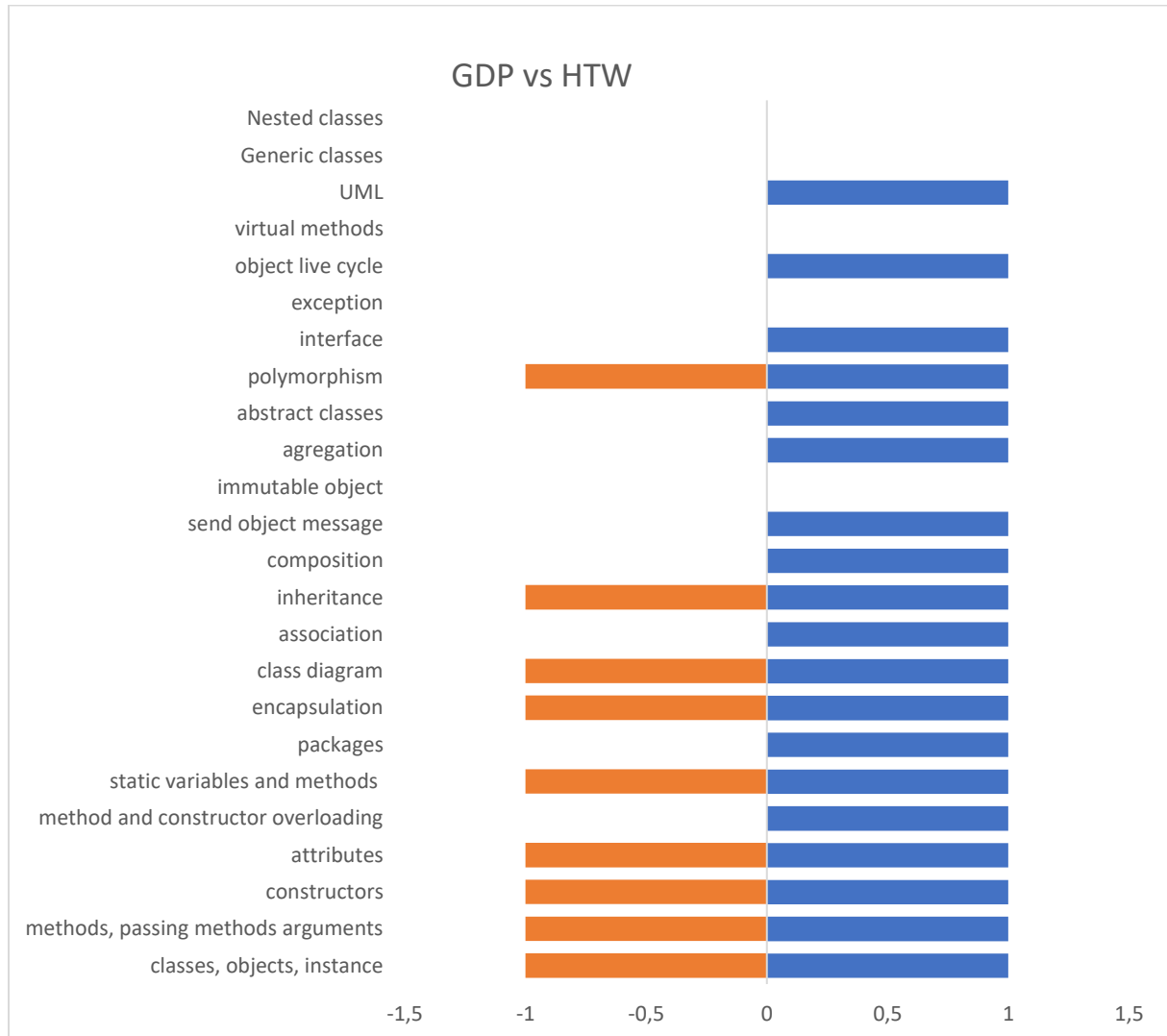


Chart 41 – Topics analysis in Germany

Following topics are not in the list of all high schools' topics:

- method and constructor overloading (5)
- packages (7)
- association (10)
- composition (12)
- send object message (13)
- immutable object (14)
- aggregation (15)
- abstract classes (16)
- interface (18)

- exception (19)
- object life cycle (20)
- virtual method (21)
- UML (22)
- Generic Classes (23)
- Nested classes (24)

In Germany, there are also some topics that are not taught at University. They are displayed in the following list:

- immutable object (14)
- exception (19)
- virtual methods (21)
- generic classes (23)
- nested classes (24)

Next chart will show comparison in Croatia – Faculty of organization and informatics and Srednja škola Ivanec:

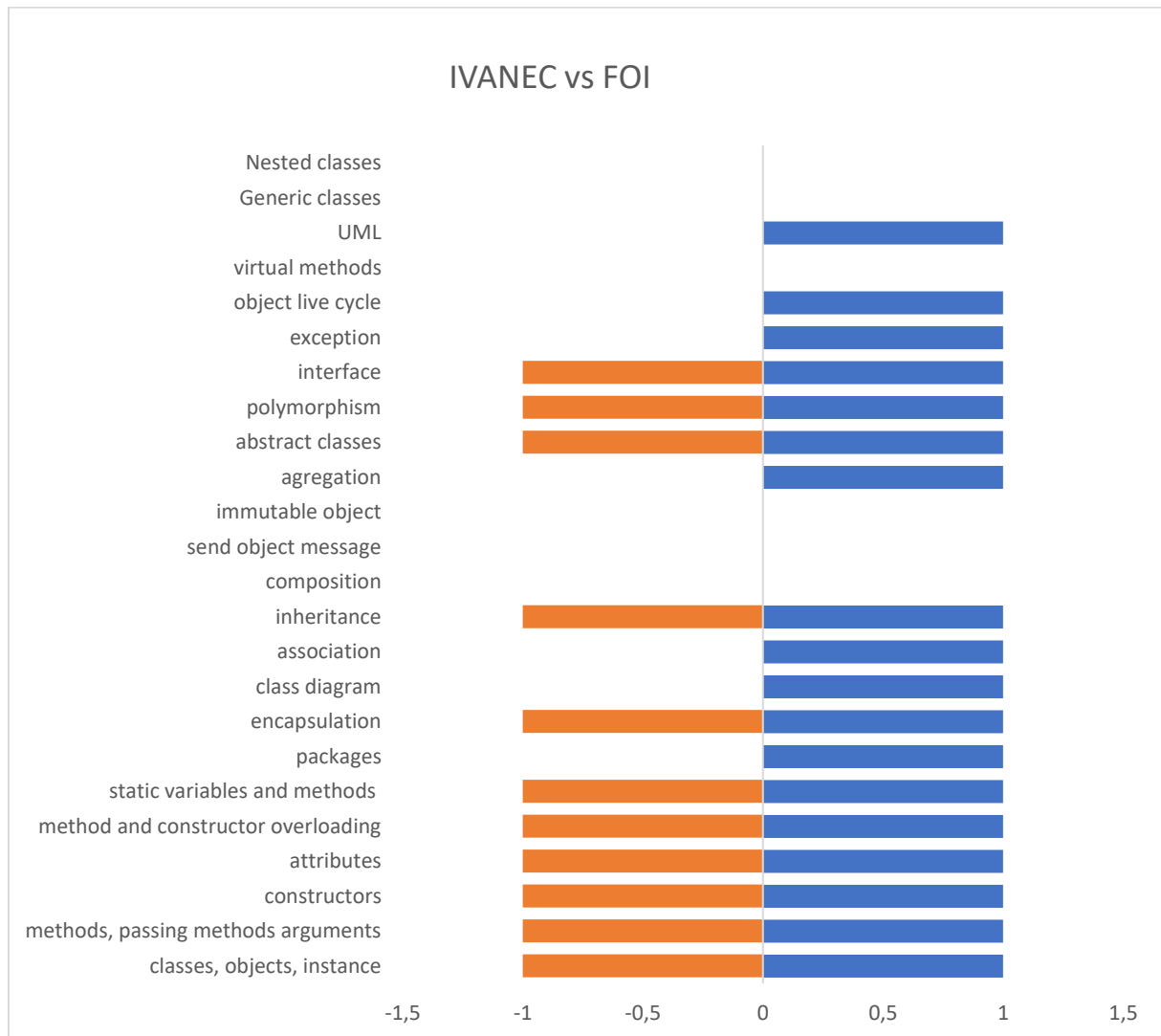


Chart 42 – Topics analysis in Croatia

Following topics are not in the list of all high schools' topics:

- packages (7)
- class diagram (9)
- association (10)
- composition (12)
- send object message (13)
- immutable object (14)
- aggregation (15)
- exception (19)
- object life cycle (20)
- virtual method (21)
- UML (22)
- Generic Classes (23)
- Nested classes (24)

In Croatia there are also some topics that are not taught at University. They are displayed in the following list

- composition (12)
- send object message (13)
- immutable object (14)
- virtual methods (21)
- generic classes (23)
- nested classes (24)

Last country to compare is Serbia – Univerzitet u Beogradu and Gimnazija Ivanjica. Following chart shows topics.

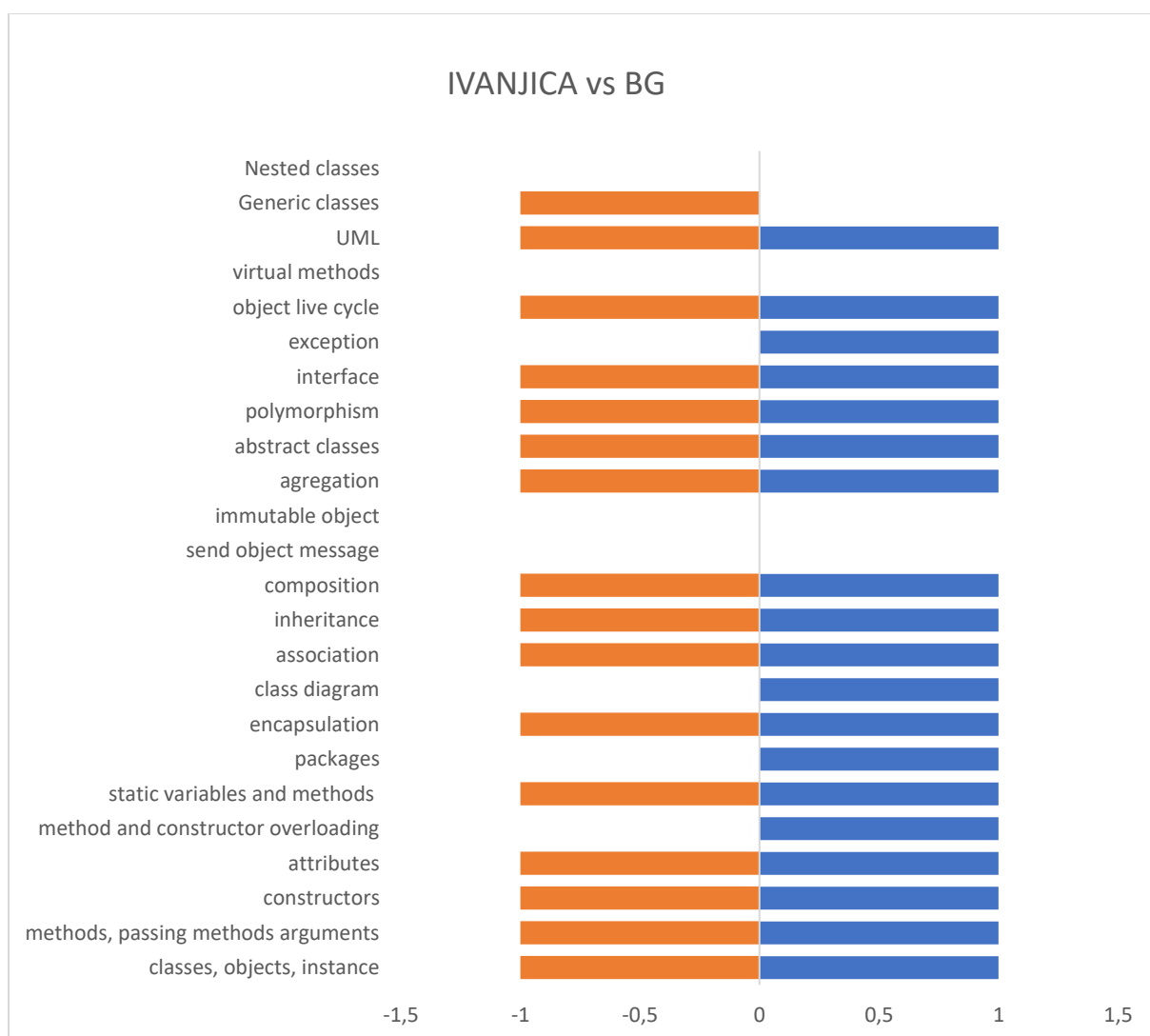


Chart 43 – Topics analysis in Serbia

Following topics are not in the list of all high schools' topics:

- Method and constructor overloading (5)
- packages (7)
- class diagram (9)
- send object message (13)
- immutable object (14)
- exception (19)
- virtual method (21)
- Nested classes (24)

In Serbia there are also some topics that are not taught at university. They are displayed in the following list:

- send object message (13)
- immutable object (14)
- virtual methods (21)
- generic classes (23)
- nested classes (24)

There is also one topic that is taught at high schools but not in university:

- generic classes (23)

2.2.2. Conclusion

Almost all topics are taught at each universities except topics 19 (exception) and 12 (composition) which are taught at 4 institutions, topic 13 (send object message) which is missing at 2 institutions and topics 14 (immutable object), 21 (virtual methods), 23 (generic classes) and 24 (nested classes) which are taught only at 2 institutions. We can conclude that the main idea of OOP teaching is satisfied at all universities. We should recommend exception topic to be added at missing institution. Such topic is needed in case of making robust applications.

If we look to the list of topics that are taught at high schools, following ones are taught at each institution. They are topics 1 (classes, objects, instance), 2 (methods, passing methods arguments), 3 (constructors), 4 (attributes), 6 (static variables and methods), 8 (encapsulation) and 11 (inheritance). We can conclude that all these topics cover basics of OOP programming. At each country, the set of high school topics is a subset of university topics with only one exception – topic number 23 - generic classes is taught Serbia on high school but not in university. At 4 institutions topics number 16-18 (abstract classes, polymorphism and interface) are taught. Abstract classes and interface topics are important for making larger applications and working in team. We should recommend to add this topic to all institutions to support teamwork experience. Also other two topics should be added to all high schools. It is topic (20) object live cycle for better understanding of OOP and topic (19) exception for making large applications.

2.2.3. 'Light OOP' topics

From analysis above it is necessary to make list of topics called 'Light OOP' which should be taught at each high school. Such list has minimal necessary knowledge to understand the principle of OOP, to make large applications and to work in team.

1. classes, objects, instance
2. methods, passing methods arguments
3. constructors
4. attributes
6. static variables and methods
8. encapsulation
11. inheritance
16. abstract classes
18. interface
19. exception
20. object live cycle

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

2.3.1. 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 44.

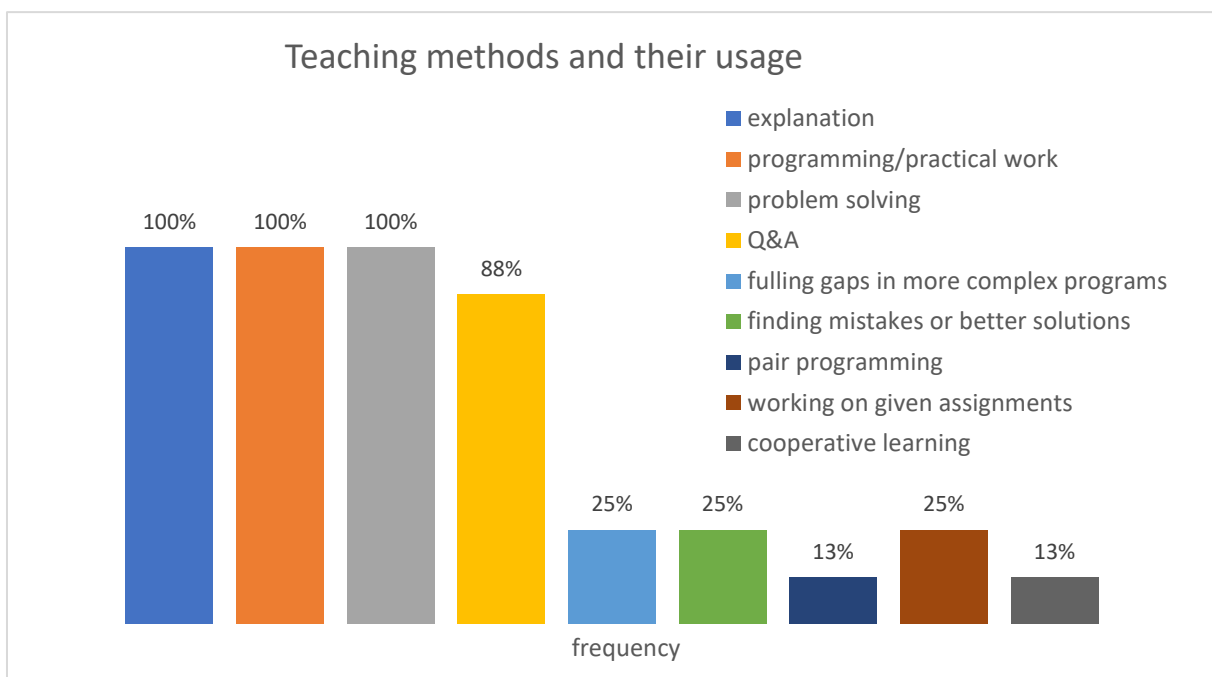


Chart 44 - Representation of teaching methods used in high school subjects

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, filling 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.

2.3.2. 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.

2.3.3. 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.

2.3.4. 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 45.

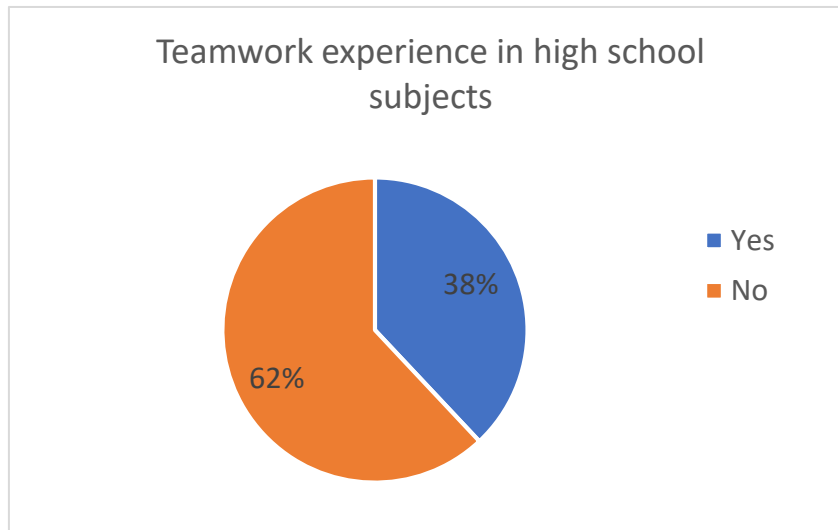


Chart 45 - The presence of students' teamwork in high school OOP subjects

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.

2.4. 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 each country there is specific situation which can be seen in Table 59.

Table 59 - 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	<ol style="list-style-type: none"> 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 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.
Practical computer science - Advanced programming	Gymnasium Dresden-Plauen, Germany	<ol style="list-style-type: none"> 1. Duden computer science high school – revision, ISBN: 978-3-8355-1313-6 2. Digital school book: inf-schule.de 3. Curriculum Computer Science Gymnasium Saxony (2019): www.bildung.sachsen.de/apps/lehrplandb/ 	Teachers use their own materials.
Datastructure and Modularization			
Seminare of programming 1	Gymnasium Pardubice, Czech Republic		Teachers use their own electronic materials.
Seminare of programming 2			
Applied Informatics – Seminar, 3rd grade	Obchodná akadémia Považská Bystrica, Slovakia	<ol style="list-style-type: none"> 1. PECINOVSKÝ, R.: We start programming in Python, Praha, Grada publishing, 2020, ISBN 978-80-271-1237-1 2. MATTHES, E.: Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, No Starch Press, 2019, ISBN 978-15-932-7928-8 	
Applied Informatics – Seminar, 4th grade			
Object Oriented Programming	Gimnazija Ivanjica, Serbia	<ol style="list-style-type: none"> 1. VUKOVIC, D.: Programming-class and objects 2. http://www.ucenjenadaljiniu.edu.rs/course/view.php?id=578 3. MILES, R.: C # basics of programming 4. LIBERTY, J.: Programming in c # 	

		5. GOCIC, M.: Programming language c # -questions, answers and solved tasks 6. MATKOVIC, S., DJURISIS, M., BAJKOVIC-LAZAREVIC, B., MA, ZORANOVIC, D.: Fundamentals of Programming in the Environment of Graphic Operating Systems- programming language c # 7. Material from various faculties, sites, courses,...	
--	--	--	--

2.5. 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 divided by countries are shown in Table 60.

Table 60 - Problems that teachers are facing with and suggestions for improvement the quality of classes

Country	Problems and suggestions for improvement
Croatia	<ul style="list-style-type: none"> • More practical tasks which will include students cooperating and working together (teamwork) • More materials with practical tasks and exercises to support teacher's lessons • More hours dedicated to OOP in compulsory subjects in high schools • 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
Germany	<ul style="list-style-type: none"> • Simple concept of class-object-method-attribute is taught in grade 7-10 in various ways, but there is not enough time for programming and practical applications • Motivating tasks like programming games should be implemented in the curriculum • OOP is only a small segment in secondary school, there should be more time for programming projects • Formative assessments should be used • There is normally a high heterogeneity, more individual learning settings should be offered by teachers
Czech Republic	<ul style="list-style-type: none"> • Learn object principals and programming code in obligatory subject – Informatic • Learn simple programming techniques in obligatory subject – Informatic • Learn programming language in obligatory subject - Informatic

Slovakia	<ul style="list-style-type: none"> • Reduce the amount of curriculum devoted to procedural programming at the expense of OOP • Attract students to programming by some nice projects • Use the material "OBJEKTOVÝ PRÍSTUP K RIEŠENIU PROBLÉMOV" developed at UNIZA, which is an OOP in a reasonable and easy-to-understand form • Add the possibility of elaboration within the group of students • Reduce time needed for repetition of knowledge from 3rd class and to demonstrate usefulness of programming to students
Serbia	<ul style="list-style-type: none"> • Lack of literature • The plan and program are roughly done for all IT courses, but different programming languages are used in schools (some work in C#, others in Java...) and are not harmonized • Teachers are looking for material for their lessons from various sources, magazines, textbooks and everyone does their best to make the lesson as good as possible, in order to help students understand and comprehend

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.

2.6. Additional comments

During the process of gathering data, all the participating partners were also asked to write down all the additional comments which might be relevant for this analysis.

Partners from Serbia stated that in the 1st and 2nd grades of high school education, while teaching the subject Programming in the IT direction, classes and objects are mentioned as a concept of Object Oriented Programming (OOP). Functions (methods) are also taught, but their essence is not included because they are taught in detail from the subject of OOP in the 3rd grade. In all classes of other fields (excluding IT) Python is taught as a programming language, but OOP related topics are not taught. OOP is mentioned as a concept in programming.

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.

In Germany, Practical computer science - Advanced programming is one of four optional topics in the curriculum. Teachers can decide to not implement OOP in their lessons. Partners also stated that basic programming skills should be taught in grade 7-10, but often the teacher has to start at the beginning and repeat all the basics from lower grades.

Regarding the subject called Applied Informatics – seminar in Slovakia, this subject is compulsory for around 10 students. Preparation of students for informatics related universities is only marginal goal of the school because its main focus is on economic related universities. From 10 students, there are usually only 2 who want to study informatics at university. It is really hard to motivate or engage students that do not want to continue in further study of informatics. Also, another problem is that some students are not good in mathematics and solving some simple problems, such as prime checking, is really hard for them. Regarding the subject Applied Informatics in 4th grade, this subject is continuation of the same subject from 3rd grade. Most of the mentioned topics are considered only marginally because students forget many things from the previous class during summer holiday and during the school year, they have various activities related to the last class and school-leaving examination ("maturita"). Actually, in the second half of the school year we primarily deal with students that want to continue further study of informatics at universities.

2.7. 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 61.

Table 61 - 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
Gymnasium Dresden-Plauen	OOP and programming related topics	weekly offers after school	programming for beginners, programming for advanced, 3D Printing, game programming with

		time for interested students	Python, electronics and robotics (Arduino, LEGO Mindstorms, etc.)
	Computer science – compulsory subject	7th - 10th grade	basic knowledge about algorithm and programming (basic algorithm structures), computer network, databases, hardware, multimedia, etc.
Gymnasium Pardubice	Informatics - obligatory subject	2nd grade	algorithm, pseudocode, flow chart, Scratch, variables, data types, input/output operations, relation, arithmetic and logic expressions, basic algorithmic structures (sequence, selection, iteration)
Obchodná akadémia Považská Bystrica	Applied Informatics – Seminar – obligatory subject	1st and 2nd grade	flowcharts, basics of algorithmization explained using real world examples (e.g., cooking recipe), introduction to scripting in Python in IDLE environment
	Web pages development – obligatory subject	2nd grade	create a simple website, but also to prepare images, videos, text parts of the site by appropriate editing
Gimnazija Ivanjica	Computer application 1	1st grade	information and communication technologies in modern society, data organization and adaptation of the working environment, creating and editing digital documents, table calculation programs, application of data processing
	Computer systems	1st grade	introduction to computer systems, digital data record, logical basics of data processing, fundamentals of computer architecture and organization, assembly programming
	Programming 1	1st grade	concept and examples of algorithms, basic concepts of programming, languages and program development environments, basic algorithms of linear and branched structure, basic cyclic structure algorithms, detailed overview of basic data types (variables, constants, operators and expressions), arrangements, low and basic algorithms for working with them, multidimensional arrangement
	Computer application 2	2nd grade	computer graphics, multimedia, presentations on the internet
	Operating systems and	2nd grade	introduction to operating systems, processes, competitiveness and synchronization of processes,

	computer networks		stack, memory management, file system, control of input-output devices, computer networks
	Programming 2	2nd grade	multidimensional arrays, matrices and basic algorithms for working with them, user defined types, program input and output, algorithm correctness analysis, algorithm complexity analysis, elementary techniques of construction of efficient algorithms, use of data structures, basics of recursion, general techniques of algorithm construction, projected task
	Database 1	3rd grade	database design, relative databases, question language SQL
	Programming 3	3rd grade	graphs and algorithms for working with graphs, text algorithms, geometric algorithms, number theory algorithms, bit algorithms, overview of selected data structures and AI
	Computer application 3	3rd grade	application of computers in mathematics, application of computers in various fields, computer graphics
	Web programming	4th grade	computer networks, internet services and protocols, description language html, style sheets - css language, javascript language script for client programming, server programming
	Database 2	4th grade	programming and databases data, other current technologies
	Programming paradigms	4th grade	expressive logic, predicate logic, logical programming, functional programming

All the subjects mentioned in the Table 61 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.

As mentioned at the beginning of this analysis, there is a very big difference in number of subjects among schools in which programming content in general is taught. For example, there are total of twelve subjects in Gimnazija Ivanjica which are related to IT contents, while there is only one subject in Gimnasium Pardubice. It is evident that Gimnazija Ivanjica is far more IT oriented school and their students are no doubt better prepared for any university in which computer technology knowledge is necessary.

Bibliography

1. SPU, „Innovative State Educational Program,“ 2014. [Online]. Available: https://www.statpedu.sk/files/articles/dokumenty/inovovany-statny-vzdelavaci-program/informatika_nsv_2014.pdf. [Cit. 05 12 2022].
2. Ľ. Šnajder a J. Guniš, „Prieskum kompetencií a postojov učiteľov informatiky v oblasti programovania,“ 2022. [Online]. Available: https://di.ics.upjs.sk/publikacie/prieskum_ucitelia_programovanie_2022.pdf. [Cit. 05 12 2022].
3. A. Fogašová, „Survey among university students,“ Informatika 2.0, Bratislava, 2022.
4. Y. Qian, and J. Lehman.: “Students’ Misconceptions and Other Difficulties in Introductory Programming: A Literature Review”. ACM Transactions on Computing Education, Vol. 18, No. 1, Article 1. Publication date: October 2017.
5. F. Batur, „How Does an Educational IDE Influence Students' Conceptions of Object-Oriented Programming? Design of a Ph. D. Research Project to Explore Secondary School Students' Conceptions of OOP,“ *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*, pp. 1-2, 2019.
6. Kölling, M, Brown, NCC & Altdmri, A 2017, 'Frame-Based Editing', *Journal of Visual Languages and Sentient Systems*, vol. 3, pp. 40-67.
<<http://www.ksiresearch.org/vlss/journal/VLSS2017/vlss-2017-kolling-brown-altadmri.pdf>>
7. Peter Hubwieser, „A smooth way towards object oriented programming in secondary schools“, 2007.
8. T. Tóth a G. Lovászová, „Mediation of knowledge transfer in the transition from visual to textual programming,“ *Informatics in Education*, zv. 20, %1. vyd.3, pp. 489-511, 2022.
9. H. Zhu a M. Zhou: „Methodology first and language second: A way to teach object-oriented programming.,“ *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 140 - 147, 2003.