Does Cyclic Learning have Positive Impact on Teaching Object-Oriented Programming?

Virginia Niculescu and Camelia Şerban and Andreea Vescan
Faculty of Mathematics and Computer Science
Babeş-Bolyai University, Cluj-Napoca,România
{vniculescu, camelia, avescan}@cs.ubbcluj.ro

***Abstract*—This Research to Practice Full Paper presents a study regarding applying cyclic learning strategy with a special focus on object-oriented programming and states our findings, emphasizing both the advantages and disadvantages.**

The research considers as a use-case the teaching activity in the Faculty of Computer Science of Babeş-Bolyai University. The analysis takes into consideration several disciplines that compass a set of interconnected teaching objectives and aspects: (1) fundamental concepts and mechanisms (F) defined by object-orientated programming paradigm, (2) design principles, heuristics, and rules (D) that act as strategies implied in object-oriented design, and (3) functional and non-functional requirements related to software architecture (A).

The study is directed by a statistical analysis of the grades obtained by the students at different courses that treat the (F, D, A) aspects, and also the results obtained at the Bachelor's final exam that evaluates the level of the acquired fundamental knowledge. Their evolution and correlation during a period of several years are analyzed, and together with an analysis of the degree of absorption of the students in the IT industry form the base of the study.

***Index Terms*—cyclic learning, object-oriented programming, software design, courses, undergraduate, IT industry.**

I. Introduction

Object orientation in programming and design is from more than two decades the main approach in software development. Even if the last years trend emphasizes the importance of combining with other approaches - e.g. functional programming, reactive programming - still the object orientation remains the main and the most used mainstream approach. This is why a careful analysis of the teaching methods applied in studying this domain remains a topic of great interest.

The main goal of Object Oriented Programming (OOP) paradigm is to develop software systems that meet quality attributes like reusability, maintainability and reliability. But, as in chess, only knowing the rules (i.e. OOP mechanisms in our case) is not sufficient to meet quality attributes. We need strategies, expressed in terms of design principles, heuristics and rules, together with software development methodologies that define software architectures, as a macro granularity level in the development of software systems.

Therefore, three important knowledge areas - **FDA** - are needed to be taken into consideration when we aim to learn how to develop a software system based on object oriented paradigm:

1) The main **F**undamental concepts, principles and mechanisms defined by OOP paradigm - **F**.
2) **D**esign principles, heuristics and rules that act as strategies implied in designing the system - **D**.
3) Software **A**rchitectures that aim to define the main constituent elements of the system architecture and to establish rules to wire them together in order to build the system having satisfying both functional and non-functional requirements - **A**.

The knowledge related to previously enumerated three main areas could be introduced to the students in sequential, disjunctive stages, or using a cyclic learning driven approach.

Lately the *cyclic learning* driven teaching approach is chosen very often from the primary school level to the academic one. The idea is to meet a concept several times, using a cyclic manner, starting from a basic definition of it, considering a particular case, and going to the definition of the general case, when that concept is supposed to be well understood, reaching a maturity level of generalization, the learner being able to work with that concept at an advanced level.

The three introductory courses for teaching object oriented programming that are being taught at our faculty have as a main objective in their syllabus the following one:

> "Students have to be able to develop small to medium applications using the main concepts and mechanisms defined by object orientation programming paradigm, together with design strategies expressed in terms of principles, heuristics and rules, and use/build well defined software architectures for these applications.

This is also an objective found in the syllabus of most of the introductory set of courses that teach object oriented programming.

In order to reach the above mentioned objective of developing medium application, following an *Incremental Software Development Methodology* [18], [2], [7] (i.e. the functionalities are split in Iterations, each iteration being a functional version of the system that client could use it), a cyclic learning approach comes as a necessity.

It is worth to be mentioned that many of the students that begin to study Computer Science at our faculty already have some knowledge about the fundamental mechanisms of object oriented programming (achieved in high-schools). This knowledge is more oriented on practical application, and also the levels of this pre-existent knowledge are very different. (This situation is very probably encountered in many other universities.)

Since it is important to attract all the students even from the beginning, we needed to include into the first programming introductory courses also elements that are absolute new for the students, such as design and software architectures.

The above mentioned reasons and the ACM Curricula [1] recommendations lead to the decision of introducing a cyclic learning driven approach for teaching object oriented paradigm at our faculty starting with 2011. It has been a collective effort and the method has been improved each year.

The goal of this paper is to present how cyclic learning approach could be introduced for teaching object oriented programming and also to analyze the impact of introducing cyclic learning driven approach.

The paper is organized as follows: The next section briefly describes the cyclic learning approach. Section III presents how the teaching of the object-oriented programming has been changed in our faculty, and Section IV describes the conducted analysis, both formal and informal, and the corresponding results. The conclusions are emphasized in the last section.

II. Cyclic Learning

The learning cycle has been first described as a teaching approach based on three distinct phases of instruction [21]:

1) **the exploration**, which provides students with first-hand experiences with science phenomena;
2) **the concept introduction**, which allows students to build understanding of science concepts;
3) **the concept application**, which requires students to apply their understanding to situations or new problems.

In time, several different models, including different numbers of phases, have been proposed. A popular version of the learning cycle is the 5E Model: Engagement, Exploration, Explanation, Elaboration, and Evaluation [6]. It incorporates the three original learning cycle phases while adding two more.

Educational taxonomies are useful tools in developing learning objectives and assessing student attainment. The most widely cited in the literature is the Bloom's taxonomy [8]. Bloom's taxonomy has six categories, where each category builds on the lower ones:

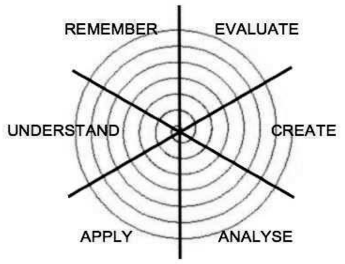


Fig. 1. The Bloom's taxonomy as a Spiral Taxonomy that it is used to evaluate a spiral learning process – image taken from [19].

1) Knowledge;
2) Comprehension;
3) Application;
4) Analysis;
5) Synthesis Abilities and skills, and
6) Evaluation.

Bloom's taxonomy has been revised by Anderson et al [22], [5], which changed the nouns listed in the Bloom's model into verbs, reversing the order of the highest two levels. These taxonomies do not define a sequence of instruction but define levels of performance that might be expected for any given content element. A learner performing at a higher level is expected to be able to perform at the lower levels in the cognitive hierarchy. A key difference between the revised taxonomy and the original taxonomy is that the type of knowledge elements are also defined: Factual knowledge, Conceptual knowledge, Procedural knowledge, Metacognitive knowledge. This provides a matrix into which learning objectives are mapped.

The revised Bloom's classification [8] defines the following six categories:

1) **Remember** – Recognizing, Recalling;
2) **Understand** – Interpreting, Exemplifying, Classifying, Summarizing, Inferring, Comparing, Explaining;
3) **Apply** – Executing, Implementing;
4) **Analyze** – Differentiating, Organizing, Attributing;
5) **Evaluate** – Checking, Critiquing;
6) **Create** – Generating, Planning, Producing.

The implied way of teaching is very similar to the spiral process of teaching which is described in [19] in correlation to Bloom's taxonomy and which is well emphasized by the Figure 1. This taxonomy is considered appropriate for Computer Science teaching.

Simplified variants of Bloom's taxonomy have been considered, either by grouping each two neighbor levels into one [25], or by extracting the most influential three categories [20].

This last variant is also in direct correlation to ACM classification of the level of knowledge [1]:

1) **K** = Know the term (˜ ACM: **Familiarity**);
2) **C** = Comprehend so as to illustrate (˜ ACM: **Usage**);
3) **A** = Apply it in new context (˜ ACM: **Assesment**).

The advantage of using cyclic learning approach lies in the fact that the students return to previously learned concepts with regularity, and each time they have the opportunity to extend and deep in their knowledge related to them. Thus, in order to learn new concepts, programming methodologies, principles, and rules, we resume those already studied and try to understand the need for new ones.

The following section links cyclic learning approach to the object-oriented paradigm, describing the way it is implemented at our faculty in teaching object-oriented programming.

III. From sequential to cyclic learning based approach in teaching OOP

The main courses from our curricula that were influenced by the introduction of cyclic learning approach are the following:

- FP: *Fundamentals of Programming* (semester 1);
- OOP: *Object-Oriented Programming* (semester 2);
- APM: *Advanced Programming Methods* (semester 3);
- SE: *Software Engineering* (semester 4);
- SDI: *System for Design and Implementation* (semester 4);
- PDC: *Parallel and Distributed Computing* (semester 5);
- VVSS: *Verification and Validation* (semester 6).

The first three courses from the above list are considered introductory courses for FDA and the last four courses have the prerequisites that students should be able to use those elements defined by FDA at the *Apply (Assessment)* level. We aim to reach this level even from the APM course for most of the FDA elements. The main list containing bibliographic references for the first three courses is defined by the following books: [13] , [14], [16], [15], [17].

Our faculty imposes a two-fold Bachelor's exam consisting of a written exam that evaluates the acquisition of the fundamental knowledge and a graduation thesis that should be defended. The written exam comprises three fundamental areas: operating systems, database, and algorithms & programming. The courses we analyze (FP, OOP, APM) are connected to the last area, and we will denote this part of the Bachelor's final exam with **BF**. The knowledge students need to acquire for this exam come from the first two types of FDA elements, most of them are from Fundamental topics and some are from Design topics. Also, the exam subject is linked with *Data Structure and Algorithms* course (semester 2); a sample of such a subject is given in the Appendix 1.

We recall here the fact that the proposed cyclic learning approach aims to develop the necessary abilities for students in order to build medium applications using the main concepts and mechanisms defined by object orientation programming paradigm, together with design strategies and defining an architecture for the developed software. The development methodology is iterative and incremental using *Feature Driven Development* methodology [24]. An iteration groups a list of features (functionalities) that are split in tasks. Even more, *Test Driven Development* [4] approach is used to design and test each feature.

Additionally, by teaching the FDA elements we aim not only to develop students' skills for developing a software system, but also to emphasize the importance of designing a system architecture that meets quality factors such as: maintainability, reliability and reusability.

The time stream in attaining the above mentioned goals in teaching object-oriented paradigm is as follows: we start from the FP course, using Python as programming language, then continue at the OOP course in C++ programming language and aim to reach the *Apply* level, for most of the discussed concepts, at the APM course, using Java and C# programming languages. The concepts learned using the cyclic learning approach are re-enforced during the first three courses: starting from a basic definition of a concept, considering a particular case, and going to the definition of the general case, when that concept is supposed to be well understood. Further, it reaches a maturity level of generalization; the learner thus being able to work with that concept at an advanced level.

Next we will highlight the way these concepts are introduced, discussing the encountered problems in teaching them. The presentation is directed by the areas these concepts belongs to: Fundamental, Design, and Architecture.

A. Analysed Items: OOP Fundamentals Concepts

F_0 - Data Abstraction
F_1 - Encapsulation: visibility, protection
F_2 - Inheritance: subtyping
F_3 - Composition
F_4 - Polymorphism: subtyping vs parametric
F_5 - Overriding vs. overloading
F_6 - Abstract classes, Interfaces
F_7 - Programming by interfaces
F_8 - UML: class diagrams, relations:associations, generalization/specialization, dependency

The first fundamental OOP concept that is introduced is *Abstraction* - viewed as the fundamental way used to understand and comprehend a complex issue.

In the context of object-oriented programming Booch [9] offers us the following definition of an abstraction: "An abstraction expresses all the essential characteristics that make an object different from some other object; abstractions offer a precise definition of the object's conceptual borders from an outsider's point of view."

This concept is defined at FP and OOP courses by means of ADT - Abstract Data Type. The students follow the *Domain Driven Design* approach [12] to identify the domain entities and to define them by means of abstraction. This comprises the conceptual model of the application. At the APM course data abstraction concept reaches A level relative to the cyclic learning approach, the students being able to define easily the conceptual model of the developed system or to identify other involved abstractions.

Another important fundamental concept of object orientation is *Encapsulation*. This is strongly connected with data abstraction. Abstraction is the process that defines the interface of the object while the encapsulation defines the object's representation (structure) together with the interface implementation. Encapsulation is used for separating the "contractual" interface of an abstraction from its implementation, and this also leads to information protection. It is emphasized that an object has two distinct parts: the object's interface and the implementation of this interface.The fact that the concept of *Interface* is initially introduced with a very simplified definition that is meant only to capture some aspects of a class leads to some difficulties in completely understanding the encapsulation concept. It is often confused with "information hiding".

A group of abstractions often form a hierarchy and by identifying this hierarchy we can greatly simplify the understanding of the problem. *Inheritance* defines a relation among classes in which a class shares its structure and behavior with one or more other classes (we talk about simple and multiple inheritance). In this context, inheritance implies a hierarchy of generalization/specialization type in which the class that derives is specialized, by changing or adding new behaviour or new attributes while reusing the existing ones.

The difficulties that arise from here are related to the fact that at the FP course the inheritance concept is viewed only as a code reuse mechanism. However, this concept gradually increases its level of understanding from the FP course (K - level) to APM course (A -level).

Beside generalization/specialization relations between classes, association relations are also introduced using the cyclic learning approach. They are introduced starting with FP course, emphasizing also their importance in code reusing. Aggregation and composition are defined as strong association relations. From a semantic point of view, aggregation indicates a "part of" relationship. Composition is also taught as an important relation between classes and it is discussed in the context of an important design rule: "Favor composition over inheritance". Even if it seems simple, this is a rule that usually is not followed by the first years students; they only reach C - level at APM course. But this subject is re-enforced at *Software Engineering* (SE) and *Systems for Design and Implementation* (SDI) courses (semester 4).

Interfaces are fundamental in object-oriented systems. Beside their theoretical definitions, *Interfaces* and *Abstract classes* are exemplified in the context of the above mentioned three courses in the laboratory's assignments by means of generic interfaces for data validation, services and also for CRUD repositories. For example, the students are asked to switch between different kind of repositories starting from in_memory repository, to file repository and then to xml repository. Also, the concept of interface is manipulated to a great extent into the APM course by means of *Service Oriented Architecture* and also by placing more emphasis on the design of Graphical User Interface APIs.

One of the concepts that need more attention in order to be well understood by the students is *Polymorphism*. The general definition, which states that the specific behaviour of an entity depends on its state, is difficult to be followed from the beginning and this is why the polymorphism concept is introduced first by using concrete examples. It is presented starting from FP course, but it is explained and exemplified in detail at OOP course. Since many design patterns are based on it, polymorphism is also re-enforced in the next courses. The concept of polymorphism is one of the fundamental concepts that is difficult to be deeply understood by students. Our teaching experience proves that there are many students that only reach C level for this concept. They manage to understand it but not many of them build software that really relies on it.

A similar situation is encountered when building interface oriented software (F7).

Summarizing the above mentioned discussions regarding the way the fundamental concepts are introduced and based on our experience in teaching these courses, in Table I we define the Bloom's taxonomy levels our students attain at the considered courses for Fundamental Concepts.

TABLE I
FUNDAMENTAL CONCEPTS - BLOOM TAXONOMY LEVELS OF LEARNING

Courses	**Fundamental Concepts**								
	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
FP	C	C	K	K	K	K	K	K	K
OOP	A	A	C	K	C	A	C	C	K
APM	A	A	A	C	C	A	A	C	A

B. *Analysed Items - Design principles and design patterns*

- D_0 - Low Coupling Design Principle (LoC)
- D_1 - High Cohesion Design Principle (HiC)
- D_2 - Single responsibility Design Principle (SRP)
- D_3 - Open Closed Design Principle (OCP)
- D_4 - Dependency Inversion principle (DIP)
- D_5 - Liskov substitution principle (LSP)
- D_6 - Programming by interfaces (PI)
- D_7 - Design Patterns: Observer, Singleton, Adapter, Decorator, Factory Method, Strategy, Command, Composite, Delegate (DP)

This section aims to emphasize the idea that the previously discussed fundamental concepts are linked to Design principles and patterns in approaching a cyclic learning based methodology for teaching object-oriented programming.

Having in mind the definitions of fundamental concepts, we can state that these concepts are strongly connected with the principles of "high cohesion", "open-closed" and "single responsability". A good data abstraction ensures a highly cohesive class, whereas encapsulation (separating the object interface from the object's representation) allows modifying programs efficiently, with a limited and localized effort, without affecting the various clients in any way because these depend on the server object's interface and not on its implementation.

Defining the classes hierarchy and the way the objects communicate (classes' relations) we take into account in teaching object oriented design the above mentioned design principles and patterns. Thus, these principles and patterns are not discussed independently by the way the fundamental concepts are taught, they appear as an objective of object-oriented paradigm to reach quality attributes as maintainability, testability, reliability etc. We are driven by the quotation stating that: "good internal structure assures good external quality". Thus, as we mentioned earlier it is not enough to know only the rules of the game, we need strategies to be applied. In object-oriented design paradigm, these strategies are related to design principles and patterns.

Summarizing the above mentioned discussions regarding the way the Design principles and design patterns are introduced, and based on our experience in teaching these courses, in Table II we define the Bloom's taxonomy levels our students attained at the considered courses for Design principles and design patterns.

TABLE II
Design Principles and Patterns - Bloom taxonomy Levels of learning

Courses	**Design principles and Patterns**							
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
FP	K	K	C	K	K	K	-	K
OOP	C	C	A	C	K	C	K	K
APM	A	A	A	C	C	A	C	C

C. *Analysed Items - Application development: Architecture*

- A_1 - Architectural patterns: Model-View-Controller and the variants, Three-Tier Four-Tier - multi-tiers: domain layer, presentation layer (UI), data access layer (DAO), busyness layer (BL)
- A_2 - UML: Packages, Deployment
- A_3 - Development Approaches: Feature-Driven-Design, Domain-Driven-Design, Service orientation
- A_4 - Test driven development: Unit testing, Agile/XP.

The fundamental concepts of object orientation together with the design principles, heuristics and rules are used as the building blocks to define the macro element of an object oriented system, namely its *Architecture*.

Even from the first course – FP, the students have to design a four-tier architecture for the system that they need to build for their laboratory assignments by using *Feature-Driven-Design* approach. This means that each feature of the application is developed from the presentation layer (UI) and finished it with Data Access Layer (DAO). The

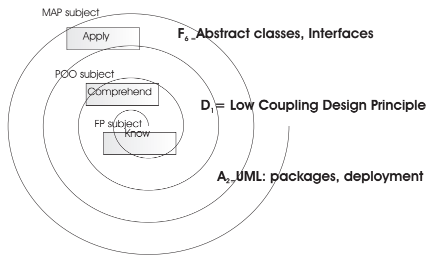

MAP subject
Apply
F_6 = Abstract classes, Interfaces
POO subject
Comprehend
D_1 = Low Coupling Design Principle
FP subject
Know
A_2 = UML: packages, deployment

Fig. 2. The Bloom's taxonomy as a Spiral Taxonomy - for F_6, D_1 and A_2 in our approach.

concept of service is somehow immature defined at FP and OOP course, but starting with APM, *Service Oriented Architecture* is more highlighted in the design and also the Model-View-Controller (and the similar variants) design pattern reaches the A-level.

In Table III, we define the Bloom's taxonomy levels our students attain at the considered courses. The architecture concepts are at the K level in FP course and then elevated to C in POO subject to A level in APM subject, whereas the others achieve only C level at APM subject.

TABLE III
Architecture - Bloom taxonomy Levels of learning

Courses	**Architecture**			
	A_1	A_2	A_3	A_4
FP	K	K	K	K
OOP	K	C	C	K
APM	C	A	A	C

In order to better emphasize the cyclic learning approach in teaching FDA concepts, Figure 2 describes the way the level of FDA elements is increased from K to A. We have exemplified only three of the FDA concepts, one for each area.

IV. Analysis

This section presents in details the conducted investigation regarding the impact of applying cycling learning method, formally stating the research questions and the performed analysis, both quantitative (formal analysis) and qualitative (informal analysis).

A. *Objective*

Our objective is to establish what impact had the application of cyclic learning transformations on the teaching results.

Firstly, we emphasize the research questions:

Research question 1: Is cyclic learning approach effective in teaching objected oriented paradigm?

Research question 2: Does cyclic learning approach enforce the knowledge transfer of the three main considered categories: F (Fundamentals), D(Design), A(Architectures)?

We performed both a formal analysis and an informal one to answer the above questions, thus the obtained grades of the involved disciplines and the Bachelor degree final exam (BF) in our study were used, and also the direct observations on the students‘s feedback and of the experienced professors.

Since the purpose of the Bachelor degree final exam (BF) (subject related to Algorithmics and Programming) is to evaluate especially the *Fundamental* (F) knowledge it is very useful to use the results of this exam in this analysis, even if it also reflects knowledge in *Data Structures and Algorithms* area.

The integration in the IT industry depends in great measure on the Software Engineering knowledge and this represents an important evaluation pillar, too.

B. Formal Analysis

The quantitative analysis is mainly based on the statistic analysis of the students results at different courses and at the final bachelor degree exam.

For the students that starts their studies in 2010 and finished them in 2013 the sequential didactic method was used. Their results are compared with the results of the students that finished their studies in the next years. We considered the years 2015, 2017, 2018 as being the representatives. The reason why the specific selected years were reasonable choices as being representative years, lie in the fact that these years represents the start for teaching these courses following a cyclic learning approach. Thus, every one of the above years added some improvements compared with the previous ones. Also, the subject of the final examination evolve during these years. But, before applying the cyclic learning approach these courses already reached a maturity level, being taught using the classical approach for 10 years.

The new approach introduced imported dependencies between the three analyzed courses (FP, OOP, APM), and this is also reflected by the correlations between them. (See Table IV for details.)

TABLE IV
Correlations between the courses FP and (POO, APM)

Graduates classes	**Pearson correlation**	
	POO	***APM***
2010-2013	0.633	0.558
2012-2015	0.639	0.502
2014-2017	0.709	0.552
2015-2018	0.705	0.498

As we specified before we will use (BF) exam in order to evaluate the impact of using cyclic learning in (F) type knowledge acquisition.

Table V contains in the first column the values relating to the Computer Science specialization, i.e. the number of graduated students for each class. It also contains the values for the mean and standard deviations for the (BF) exam. We can notice that the highest value from the means is for the class of 2012-2015.

TABLE V
Mean and standard deviations of (BF) results for each considered graduation class.

Graduates classes	**Mean and Standard Deviation**		
	N	***Mean***	***SD***
2010-2013	99	8.14	1.72
2012-2015	93	8.84	1.03
2014-2017	105	7.40	1.32
2015-2018	138	7.04	1.09

We have performed various analysis regarding the available data set, i.e. investigating the correlations between the results obtained at the three main analyzed courses (FP, POO, APM) and the Bachelor's degree exam (BF), the independent t-test and Cohen's d effect size, for the Computer Science specialization

Table VI contains in bold the correlations that are significant at $p < 0.05$ between the BF exam and the results/grades of the disciplines (FP, POO, DSA, APM, SE) for the Computer Science specialization.

TABLE VI
Correlations between the courses (FP, POO, APM) and (BF) grades.

Graduates classes	**Pearson correlation**				
	FP	***POO***	***DSA***	***APM***	***SE***
2010-2013	**0.56**	**0.56**	**0.45**	**0.57**	**0.54**
2012-2015	**0.47**	**0.41**	**0.34**	**0.45**	**0.55**
2014-2017	**0.51**	**0.47**	**0.45**	**0.51**	**0.42**
2015-2018	**0.28**	**0.37**	0.06	**0.33**	**0.36**

Consulting Table VI regarding correlations between the grades obtained at the disciplines and at the BF exam, we may conclude that in general they correlate, the grades of the disciplines impact the grade obtained at the BF exam. There is only one strong correlation (> 0.6) between FP and BF, majority are moderate to strong correlations (> 0.4), and several are of type weak to moderate (> 0.2). Thus, for all graduation classes the correlations are in general moderate to strong correlations.

We have performed independent t-test, comparing the class of 2010-2013 before applying cycling learning with the classes after applying cycling learning (2012-2015, 2014-2017, and 2015-2018).

An independent t-test [11] is used to compare the means of two independent groups in order to determine whether there is statistical evidence that the associated population means are significantly different.

The hypotheses may be stated as:

- H_0: $\mu_1 = \mu_2$.
- H_a: $\mu_1 \neq \mu_2$.

Table VII contains the p values: for the values in bold the obtained values are significant, i.e. we reject the null hypothesis and we accept the alternative hypothesis, meaning there is a difference between the obtained grades when applied sequential learning and when applying cycling learning.

TABLE VII
BF grades – independent t-test between the class (2010-2013) and the following classes (2013-2015, 2014-2017, 2015-2018).

Graduates class	**Independent t-test**		
	p value < 0.05?		
	2012-2015	***2014-2017***	***2015-2018***
2010 - 2013	**0.001000**	**0.000167**	**0.022000**

The APA Publication Manual [3] states that that it is "almost always necessary to include some index of effect size or strength of relationship in your results section". The general principle to be followed is to prepare and provide not only the information about statistical significance but also with enough information to assess the magnitude of the observed effect or relationship. Practical significance is generally assessed with some measure of effect size.

A common measure of effect size is d, known as Cohen's d effect sizes [10]. This can be used when comparing two means (in our case for the t-test), and is simply the difference in the two groups' means divided by the average of their standard deviations. Cohen [10] suggested that $d = 0.2$ be considered a 'small' effect size, 0.5 represents a 'medium' effect size and 0.8 a 'large' effect size. This means that if two groups' means don't differ by 0.2 standard deviations or more, the difference is trivial, even if it is statistically significant.

TABLE VIII
BF exam results Cohen'd effect size between the class (2010-2013) and the following classes (2012-2015, 2014-2017, 2015-2018).

Cohen's d effect size	**Cohen's d effect size**		
	d		
	2012-2015	***2014-2017***	***2015-2018***
2010 - 2013	0.486518	0.524742	0.354602

Table VIII contains the values for the Cohen'd effect size. Investigating our findings related to the obtained d values, we can see that the effect size is relatively medium.

C. Informal Analysis

The informal analysis was based on direct observations on the students and the professors feedback, and also on the information received from the IT industry, about the degree of students absorption.

In order to answer to the first question we consider the Bloom's taxonomy levels our students attain at the considered courses, i.e. the levels provided in Tables I, II, III, and the students' grades. Analyzing the content of the tables we can conclude that for some concepts the approach is effective but for others concepts there is still a need to continue teaching these concepts at the last three courses from the considered list. Although we aimed to reach a maturity level at APM course, the reality is that for some concepts students still encounter difficulties in using them. Despite the fact that there are several courses that treat a concept, there are still some basic concepts that do not reach a mature level.

Since in the first programming course (FP) many concepts and methodologies are introduced, the instructors are forced to use template applications for the laboratory works. The students are provided with skeletons of the application that they have to build.

This has the advantage of offering the students the possibility of developing real applications of medium complexity that increase their level of satisfaction.

Using these types of examples based on iterative development, students with a good observational skills could also notice the direct application of some design methodologies and approaches. On the other hand they are not involved directly in their creation and development, and many students do not deeply analyse these design techniques.

A disadvantage of this approach is related to the fact that for the following two courses (OOP, APM) the complexity level of the practical applications is not very much increased (similar types of applications are used in order to re-enforced some concepts and methodologies) and some students consider that nothing new is introduced.

Also, another drawback is given by the fact at the beginning of the degree students should receive templates/skeletons based on which they build their applications (of medium complexity) just by adding the specific functionality. This is mandatory because otherwise they could not manage their complexity at that incipient stage of learning. In some cases students operate with these skeletons adaptation without completely understanding all the details, and on the other hand, they could become some familiar with one type of skeletons that they start to consider that the first used type is the only possible one, and so they are not so open to other paradigms and approaches.

For the next courses – SDI, PDC, VVSS – that follow the three introductory programming courses cyclic learning approach was not used, but they have as prerequisites the FDA knowledge at the level specified in the first courses objectives. The experience of the last years emphasised the fact the majority of students managed to attained the requested level of knowledge, and in addition, it has been noticed a better understanding of the aspects related to design and systems architectures.

From the informal analysis we may conclude with the following overall advantages and disadvantages brought by this approach based on cyclic leaning:

Advantages:

- Students are familiar with more terms and concepts from the beginning; e.g. design patterns, application architectures, test-driven development. These could help them to have a better view about the domain of object-oriented programming and design.

- Students can be involved in applications development that require capabilities to follow a feature driven development, understanding not only the building blocks of an application, but also its components and how these components are wiring together, their dependencies and connections.
- Resuming a concept to the next courses leads to the maturity of that concept and its deeper understanding.
- The approach facilitates the access of the students much sooner on the IT jobs market place.

Disadvantages:

- Spreading too much the information could lead to the impossibility of assuring a deep enough level of learning – for example not to attain an "A" level for some important concepts and principles; e.g. encapsulation, create correct hierarchy and composition of classes, polymorphism, interface based programming, SOLID principles.
- There is a risk for some students that the familiarity with a term to give them the impression that they already achieved a complete knowledge about its meaning and application, and so their interest in a deeper associated learning could decrease.
- There is a class of students that understand in a deep extend the concepts, attaining the last level (A) from Bloom taxonomy, even from the first course. Applying cyclic learning based method for these students means to hold them back.
- For some students there is a risk to become oriented on building software applications just by applying some templates, using frameworks, or composing some already created components, without understanding very well the fundamental concepts and principles, and so to become low level executors.

Abstraction could be considered a core skill that is important for many areas of computer science. This is illustrated by Kramer in [23]. He argues that getting students to the cognitive stage in which they could apply *formal operational* is a prerequisite for the students studying many aspects of computing. Still, the nowadays development of the IT industry emphasize the fact that this ability is mandatory only for advanced classes of programmers.

D. Results

Based on the presented analysis we can answer to the two research questions.

Response for Research question 1. Cyclic learning approach is effective in teaching objected oriented paradigm. The results obtained by our analysis acknowledge the efficiency of cycling learning approach.

Response for Research question 2. We cannot state that the cyclic learning approach enforces the knowledge transfer of the F(Fundamentals) since the results obtained at BF exams decreased (but in a small measure), but we can state that the knowledge in the other two categories has been enforced D(Design), A(Architectures).

V. Conclusions

We have analyzed the impact of the introduction of cyclic learning approach in teaching object oriented programming in a Computer Science specialization.

Several courses that form an interconnected chain of learning programming were transformed during a period of seven years when iterative improvements have been done. The applied changes were directed by the ACM Curricula Report [1], too.

This impact has been evaluated by applying statistical analysis over the grades obtained by the students to each course from the interconnected chain, but also based on the final graduation exam, that in our faculty, give insights about the degree of acquired fundamental knowledge, too. Statistical analysis shows that the degree of the acquisition of this fundamental knowledge was not essentially degraded and the curricula adaptation has been improved each year.

The informal analysis is also very important since it emphasizes some advantages and disadvantages that are directly connected to the courses content and their impact on the students' implication and interest.

This approach introduces from the first course various knowledge about programming, but also about design and architectures; this increases the students interest in studying software application development from the beginning of the learning cycle.

Even if we don't have a formal statistical analysis of the impact in the level of adaption of the students for IT job market, we may noticed an important improvement proved by the increasing number of students that find a job before or immediately after their graduation. These results are especially important as Computer Science programs prepare graduates for contemporary workplaces.

As a further work we intend to conduct an analysis, which, based on well defined questionnaires, to better evaluate the students' feedback regarding the cyclic learning approach and also to formally evaluate the impact of it on their adaptation level to the IT jobs market.

Acknowledgment

The authors gratefully acknowledge the financial support by the Advanced Fellowships Babeş-Bolyai University, Institute for Advanced Studies in Science and Technology, Romania.

References

[1] *** *ACM Curricula 2013 Report.* https://www.acm.org/education/CS2013-final-report.pdf. (2013), pp 144- 154.

[2] Agile Alliance. *Manifesto for Agile Software Development.* http://agilemanifesto.org/, (2001).

[3] American Psychological Association. *Publication manual of the American Psychological Association (6th ed.)*. American Psychological Association, Washington, DC, US, 6, (2010).

[4] Astels, D. R. *Test-Driven Development: A Practical Guide*. Prentice Hall, 2003.

[5] Anderson, L.W. and Krathwohl, D.R. & all . *A taxonomy for learning, teaching, and assessing: A revision of BloomâĂŹs Taxonomy of Educational Objectives.* (Complete edition). New York: Longman.(2001).

[6] Ballone Lena and Duran E.. *The 5E Instructional Model: A Learning Cycle Approach* The Science Education Review, 3(2), (2004).

[7] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, (1999).

[8] Bloom, B. S., Engelhart, M. D.; Furst, E. J., Hill, W. H.; Krathwohl, D. R. *Taxonomy of educational objectives: The classification of educational goals.* Handbook I: Cognitive domain. New York: David McKay Company.(1956).

[9] Booch, G., *Object-Oriented Analysis and Design with Applications.* Benjamin Cummings, Redwood City, 2 edition, 1994.

[10] Cohen, J., *Things I have learned (so far)*, American Psychologist, 45(12), 1304-1312 (1990).

[11] Dyer, C., *Beginning Research in Psychology: A Practical Guide to Research Methods and Statistics*, ISBN: 9780631189299, Publisher Wiley pp. 482 (1995)

[12] Eric E., *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison Wesley, (2003).

[13] Beck, K, *Test Driven Development: By Example. Addison-Wesley Longman, 2002.*

[14] Fowler, M. *Refactoring. Improving the Design of Existing Code. Addison-Wesley, 1999.*

[15] Fowler, M. et.all. *Patterns of Enterprise Application Architecture, Addison-Wesley Professional; 1 edition (November 15, 2002).*

[16] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing, 1995.*

[17] Kleinberg and Tardos *Algorithm Design. Pearson Educational, 2014.*

[18] Fichman, R.G. and Moses, S.A. *An Incremental Process for Software Implementation*, Sloan Management Review, No. Winter, (1999), pp. 39-52.

[19] Fuller, Ursula and all. *Developing a computer science-specific learning taxonomy.* ACM SIGCSE Bulletin 39(4):152-170. (2007).

[20] Huitt, W. *Bloom et al.'s taxonomy of the cognitive domain.* Educational Psychology. (2011).

[21] Karplus, R., and H.D. Thier. 1967. *A new look at elementary school science.* Chicago, Rand McNally and Company, 1967. pp. 204.

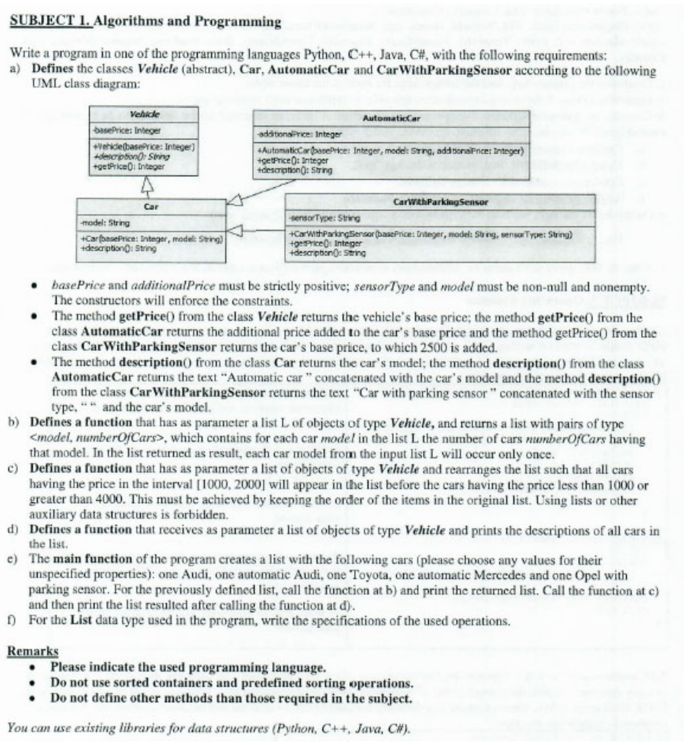
[22] Krathwohl, David R. *A Revision of Bloom's Taxonomy: An Overview.* Theory Into Practice, Vol. 41, No. 4, Taylor& Francis. (2002), pp. 212-218.

[23] Kramer, J. *Is abstraction the key to computing?* Communications of the ACM 50, 37-42, (2007).

[24] Palmer, S. R., Felsing, J. M., *Practical Guide to Feature-Driven Development*, Prentice Hall, 2002.

[25] Schaffer, H. E., Young,K. R., Ligon, E. W., Chapman, D. D. *Automating Individualized Formative Feedback in Large Classes Based on a Directed Concept Graph.* Frontiers in Psychology, 8, (2017) pp. 1-11.

Appendix



SUBJECT 1. Algorithms and Programming

Write a program in one of the programming languages Python, C++, Java, C#, with the following requirements:

a) **Defines** the classes ***Vehicle*** (abstract), **Car**, **AutomaticCar** and **CarWithParkingSensor** according to the following UML class diagram:

Vehicle
-basePrice: Integer
+Vehicle(basePrice: Integer)
+description(): String
+getPrice(): Integer
AutomaticCar
-additionalPrice: Integer
+AutomaticCar(basePrice: Integer, model: String, additionalPrice: Integer)
+getPrice(): Integer
+description(): String
Car
-model: String
+Car(basePrice: Integer, model: String)
+description(): String
CarWithParkingSensor
-sensorType: String
+CarWithParkingSensor(basePrice: Integer, model: String, sensorType: String)
+getPrice(): Integer
+description(): String

- *basePrice* and *additionalPrice* must be strictly positive; *sensorType* and *model* must be non-null and nonempty. The constructors will enforce the constraints.
- The method **getPrice()** from the class ***Vehicle*** returns the vehicle's base price; the method **getPrice()** from the class **AutomaticCar** returns the additional price added to the car's base price and the method getPrice() from the class **CarWithParkingSensor** returns the car's base price, to which 2500 is added.
- The method **description()** from the class **Car** returns the car's model; the method **description()** from the class **AutomaticCar** returns the text "Automatic car " concatenated with the car's model and the method **description()** from the class **CarWithParkingSensor** returns the text "Car with parking sensor " concatenated with the sensor type, " " and the car's model.

b) **Defines a function** that has as parameter a list L of objects of type ***Vehicle*,** and returns a list with pairs of type <*model, numberOfCars*>, which contains for each car *model* in the list L the number of cars *numberOfCars* having that model. In the list returned as result, each car model from the input list L will occur only once.

c) **Defines a function** that has as parameter a list of objects of type ***Vehicle*** and rearranges the list such that all cars having the price in the interval [1000, 2000] will appear in the list before the cars having the price less than 1000 or greater than 4000. This must be achieved by keeping the order of the items in the original list. Using lists or other auxiliary data structures is forbidden.

d) **Defines a function** that receives as parameter a list of objects of type ***Vehicle*** and prints the descriptions of all cars in the list.

e) The **main function** of the program creates a list with the following cars (please choose any values for their unspecified properties): one Audi, one automatic Audi, one Toyota, one automatic Mercedes and one Opel with parking sensor. For the previously defined list, call the function at b) and print the returned list. Call the function at c) and then print the list resulted after calling the function at d).

f) For the **List** data type used in the program, write the specifications of the used operations.

Remarks

- **Please indicate the used programming language.**
- **Do not use sorted containers and predefined sorting operations.**
- **Do not define other methods than those required in the subject.**

You can use existing libraries for data structures (Python, C++, Java, C#).