Agile and Cyclic Learning in Teaching Parallel and Distributed Computing

Virginia Niculescu
Adrian Sterca
Darius Bufnea
vniculescu@cs.ubbcluj.ro
forest@cs.ubbcluj.ro
bufny@cs.ubbcluj.ro
Faculty of Mathematics and Computer Science, Babeş-Bolyai University
Cluj-Napoca, Romania

ABSTRACT

Agile and cyclic learning are methodologies that have been recently proposed to be used in teaching Computer Science. This paper investigates their usage for the undergraduate studies on parallel and distributed computing (PDC). The aim of this analysis is to evaluate their effectiveness, and also to evaluate to which extent we have to go with the knowledge related to PDC at the undergraduate level. Also, we intended to find out the pace in which agile and cyclic learning enforces the best knowledge transfer of PDC concepts. The analysis takes into consideration several courses spread on the entire curricula, students auto-evaluation based on questionnaires, and grade results. The analysis emphasizes the fact that the tendency is to introduce more and more information and this is facilitated by an agile approach, but in the same time this should be moderated if the final goal is to assure also a good and deep understanding of associated knowledge.

CCS CONCEPTS

• **Applied computing** → **Collaborative learning**; • **Computing methodologies** → **Parallel computing methodologies**; • **Software and its engineering** → **Agile software development**.

KEYWORDS

agile methodologies, cyclic learning, knowledge levels, undergraduate studies, parallel and distributed computing

1 INTRODUCTION

The ACM/IEEE Curricula 2013 Report [7] and the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [13] emphasized the clear need for the undergraduate computer science education to be aware of the role that parallel and distributed computing plays in the computing landscape.

These recommendations led to the decision of our faculty to introduce much more topics about Parallel and Distributed Computing (PDC) even from early courses taught for Computer-Science specialization. This was a process that transformed the curricula, step by step, using an agile approach. Before this processes started, parallel and distributed computing topics have been studied at our faculty especially at master level programs. The PDC concepts are now spread across several courses, and these transformations applied to the courses led implicitly to a cyclic learning driven approach combined with an agile approach for teaching PDC. It has been a collective effort and the method has been improved each year.

Through cyclic learning a concept is introduced starting from a basic definition of it, considering a particular use-case, and then by returning iteratively, it arrives to the general definition or application, when that concept is supposed to be well understood at a high level of generalization; this has been proved to be very efficient in computer science teaching[4].

Agile methodology is derived from the IT business industry [1, 2]. In universities, using this is not only about assisting the learning process, it can also be used to manage an entire course either in respect to its content (curricula) or in respect to the people and processes involved in teaching that course.

This paper presents an investigation conducted in order to analyse the effectiveness of using cyclic learning combined with agile methodologies for PDC teaching, and also to analyse in this context to which extend new and more complex PDC knowledge could be introduced to the undergraduate students.

The paper is organized as follows: The next section briefly describes the cyclic and agile learning approaches, and the research objective is specified in the following section. Section 4 presents how the teaching of PDC is spread on several courses in the curricula of our faculty, and Section 5 describes the conducted analysis, and the corresponding results. The conclusions are emphasized in the last section.

2 AGILE AND CYCLIC LEARNING

Very important and useful tools in developing learning objectives and assessing student attainment are represented by the educational taxonomies. Bloom's taxonomy [3] is the most widely cited in the literature. The initial Bloom's taxonomy had six categories, where each category builds on the lower ones: Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation. Bloom's taxonomy has been revised by Anderson et al [8, 10], which changed the nouns listed in the Bloom's model into verbs, reversing the order of the highest two levels. The revised Bloom's classification defines

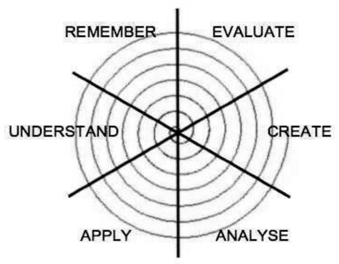

Figure 1: The spiral teaching process inspired by Bloom's taxonomy (image taken from [4]).

the following six categories: Remember, Understand, Apply, Analyze, Evaluate, Create.

These taxonomies do not define a sequence of instructions, but define levels of performance that might be expected for any given content element. This is very important when we are interested in evaluating abilities in creating products - as it is the case of software development. Simplified variants of Bloom's taxonomy have been considered, either by grouping each two neighbor levels into one [14], or by extracting the most influential three categories -[6]. This last variant is also in direct correlation to the ACM classification of the level of knowledge [7]:

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

2.1 Cyclic learning – CL

Cyclic learning defines a way of teaching which is very similar to the spiral process of teaching which is described in [4] (Figure 1) in correlation to Bloom's taxonomy. This implies that more than one iteration is needed in order to attain the highest level, and as stated in [4], this approach is considered appropriate for Computer Science teaching.

In essence, the cyclic learning approach presents the advantage of the fact that students return to previously learned concepts with regularity, in different contexts, and each time they have the opportunity to extend and deepen their knowledge related to them.

2.2 Agile learning – AL

Agile learning (AL) as a methodology is relatively new and has been only recently introduced in the academic area, being derived from the IT business industry [5, 12]. An adaptation of the main principles emphasized in the Agile Manifesto [1] ([https://www.agilealliance.org/agile101/the-agile-manifesto/]) is given in Table 1. As in the IT business case, the course holder should also assume the role of a course manager (i.e. project manager), managing also all the course activities and not only teaching it.

AL as a teaching methodology uses incremental steps and completing work through an iterative design process to meet a desired curriculum. The increments that add new functionality in short cycles correspond to the continuous increase in trainees' abilities in the agile learning/teaching process. This method is also based on working in teams and on continuous tracking of progress. AL imposes that learning objectives are modular, incremental and easily adaptable to changes [9]. But, agile instructional design could refer to any approach of training development that focuses on speed, flexibility, and collaboration.

Table 1: Agile software development ⇒ Agile learning Principles

P1	Individuals and interactions over processes and tools	Working in teams Collaborative analysis of the results - students evaluate other students, - group analysis, Enhanced student-professor interaction
P2	Working software over comprehensive documentation	Orientation on practical skills Allow software development based on frameworks/APIs/components that are not yet fully understood, but could provide fast practical results (products)
P3	Responding to change over following a plan	- Collaborate with students - Change/adapt the requirements if needed
P4	Customer collaboration over contract negotiation	Allow course adaptation and changes after the syllabus delivery (e.g. change the order in which the subjects are presented)

The CL and AL approaches have several similarities:

- build knowledge using several iterations;
- adding new functionalities/knowledge at each iteration.

Still they have their particularities:

- AL is more oriented on:
  - working in teams and encouraging cooperation,
  - producing results very fast,
  - abilities to build concrete products.
- CL is more oriented on:
  - achieving certain levels of knowledge (Bloom);
  - reiterate the same concept in order to deep the understanding.

They do not exclude each other, rather they are complementary, cycling learning being enhanced through agile methodologies.

3 RESEARCH OBJECTIVE

Our objective was to study the impact of applying cyclic and agile learning on teaching PDC topics, and to determine the amount of new information that should be introduced at each stage.

From an educational point of view, related to PDC teaching, we established three research questions:

Research Question 1: What is the pace in which cyclic learning enforces the best knowledge transfer?

Research Question 2: In which measure applying agile methods help the knowledge transfer?

Research Question 3: To which extent should we introduce PDC topics at the undergraduate level?

The first two questions are directly connected to each other referring to the same output, but they are also connected to the third because if the answer to the first questions emphasises an accelerating knowledge transfer pace, this would imply the answer to the third one: – new, and more complex PDC knowledge could be introduced.

Analysed courses

The teaching of PDC topics are spread on courses of all the three years of study, and the mandatory courses from our curricula that include these topics are enumerated in Table2.

Table 2: Undergraduate courses addressing PDC topics

Course name	Semester	ECTS	Hours per week (course,seminar,lab)
Operating Systems (OS)	2	5	2,1,2
Computer Networks (CN)	3	6	2,0,2
Advanced Programming Methods (APM)	3	6	2,2,2
Systems for Design and Implementation (SDI)	4	6	2,1,1
Web Programming (WP)	4	6	2,1,1
Parallel and Distributed Programming (PDP)	5	6	2,1,2

4 CYCLIC AND AGILE INTEGRATION

The global structure of PDC teaching is directed by cyclic learning approach. Concepts are introduced in different courses, most of them being reiterated in order to give a deeper understanding, and various usage contexts. At the courses' level, agile oriented techniques are applied in order to improve the knowledge transfer.

4.1 Cyclic learning aspects

The OS course introduces classical PDC concepts like Unix IPC (Inter Process Communication) for concurrent processes: pipe channels, named pipes (i.e. FIFO channels) and communication between processes using shared memory. Then the course moves to POSIX synchronization mechanisms for concurrent processes and threads, as: semaphores critical sections, conditional variables, or Read-Write locks. The OS course also briefly touches deadlock detection and prevention. All the programming examples are developed in C programming language under Linux.

Following, the CN class reviews concurrent processes and threads from the concurrent network server perspective: a TCP or UDP server serves remote clients concurrently and all client handlers are processes or threads sharing a common global state. The programming is still done in C/C++ programming language.

The APM course introduces more advanced PDC concepts, mainly in Java and some in C#. Here the asynchronous tasks are introduced using futures, callable, and executors, and then it moves to usage of some synchronization mechanisms for Java threads. They are discussed in the context of creating efficient applications of medium complexity, and the students are more oriented on using the mechanisms and not so much on their understanding.

SDI course introduces frameworks and APIs that facilitates distributed and web applications development. It discusses RPC, RMI, Protocol buffers & gRPC, Spring Remoting, and Restful web services.

The WP course also briefly touches asynchronous programming in Javascript covering Timeouts, Intervals, callbacks, AJAX calls and Promises.

Finally, the PDP course introduces new topics as: MPI, OpenMP and CUDA, but also reinforces, theoretically and practically, the knowledge related to threads, synchronization through semaphores, conditional variables, monitors and barriers; asynchronous tasks through futures and promises. Task partitioning with performance evaluation are discussed, analysed and applied in practical assignments. A view on the main parallel and distributed patterns is presented, too.

In the paper [11] we presented the estimation of the levels attained for many PDC items at each discussed courses. Since the interest in using PDC in different areas increased, then the focus on the related items also increased on each of these courses. Through the present analysis we intend to see also the improvement obtained by applying agile techniques.

4.2 Agile learning aspects

Various agile learning techniques are applied, especially at the micro-level, during all enumerated courses.

In general, at the laboratory classes students receive different projects or homework having a two-week deadline. This time interval can be ideally mapped to a sprint, each sprint ending with a retrospective in which students' projects or/and homework are evaluated. Based on the students results at the current assignment and on the professors' feedback, new decisions are taken regarding the next assignments (which can be adapted based on this feedback) or team up students in a different way. This corresponds to applying P1 and P3 principles from Table 1.

In what it follows, we will emphasize other several concrete use-cases in order to provide an overall view. Working with tasks, futures and executors is introduced at APM without discussing before about concepts such: threads pools, asynchronous tasks, futures and promises, etc. The students directly start to use their high level implementations, in order to improve the performance of their applications. Similarly, the students use Java parallel streams without discussing about the mechanism through which the parallelism is achieved in this case. When the associated concepts are introduced and discussed at PDP course, the fact that the students are already familiar with their potential usage and advantages, facilitates a lot the conceptual understanding.
This corresponds to applying P2 principle from Table 1.

For the WP class students team up in group of 3 or 4 in order to deliver their assignments. After each sprint, an individual evaluation of student's knowledge is performed alongside with the evaluation of the group assignment. This double evaluation allows to properly identify student's individual knowledge level and his/her contribution to the group assignment. Students with lower results have to repeat certain assignments in future sprints, while teams with lower results are broken, being reconsidered in such a way that at least two students with good results team up together with one with lower results. This assures a long time horizontal knowledge transfer and sharing between students within the same team, alongside with the teacher to student transfer. Since student's working time doesn't represent such a constraint as in a software production environment, the delay due to the repetition of certain assignments by some students could be overcame by teams and requirements adaptation. This corresponds to applying P1 and P3 principles from Table1.

Changing the order of the topics presentation is not very common in university courses, but it could emphasize the fact that sometimes, what the professor could consider as a natural order it is not always the best from the students point of view. In the initial PDP setting, MPI(Message Passing Interface) presentation was scheduled after talking about multithreading (concurrency, synchronous vs. asyncronous, etc), and also after OpenMP and CUDA (which are also based on multithreading). In the last year, we decided to change the order and start with MPI, then discuss more details about multithreading, and then introducing OpenMP and CUDA. This proved to be a better choice, this being emphasized by the students' feedback. This corresponds to applying P4 principle from Table 1.

In order to facilitate learning CUDA programming, that could rise many difficulties also because of the specific architecture and programming constraints, the associated laboratory work was set to be done in teams. The feedback was extremely positive, students managing to overcome the difficulties by working together. This corresponds to applying P1 principle from Table 1.

At PDP course students were also encouraged to follow an alternative evaluation path. The implicit path includes, besides the practical assignments, a theoretical written exam at the end of the semester. A project-based learning alternative is proposed, too. It relies on solving a 'challenging' problem, to implement a solution, but also analyse it theoretically from the design decisions and also performance points of view. Still, the students are allowed to return to the classical exam is they don't succeed with this challenging project. This corresponds to applying P3 and P4 principles from Table 1.

5 ANALYSIS

The goal of the conducted investigation was to evaluate the outcomes obtained at the end of the educational stream, and this is why the evaluation has been done mainly during the PDP course, which is the last in this stream. The analysis was quantitative but also qualitative, based on questionnaires and grades evaluation.

Analysed Items: We focus our investigation on the items that are introduced during the cyclic learning stream, with the help of agile methods. These items were presented in several courses in different contexts and use cases:

- Threads and processes – general knowledge.

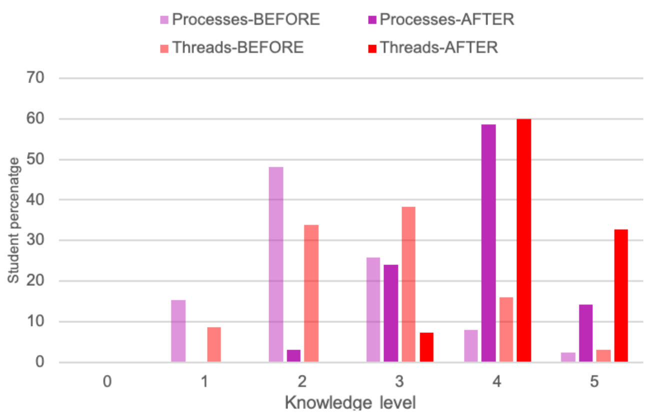

Figure 2: The before/after knowledge levels for general use of threads and processes; based on students auto-evaluation

- Synchronisation – race conditions, deadlock, etc.
- Asynchronous computation.
- Client-Server applications.
- MPI/OpenMP/CUDA programming.

5.1 Questionnaire based investigation

In order to have a reliable feedback from the students we invited them to fill up a questionnaire in Google forms (the complete results could be consulted at: [http://www.cs.ubbcluj.ro/~vniculescu/PPDquiz.pdf]). It represents students' auto-evaluation and open feedback.

We considered 6 levels of knowledge, the first specifying the state of not knowing anything about the concept under the analysis. The other 5 levels correspond to Bloom's revisited taxonomy, where the last two levels are grouped together.

This questionnaire was sent to the students from Computer Science specialization, which is formed of about 200 students; the number of students that filled up the questionnaire was 162. Since we were interested in cyclic learning evaluation of the knowledge associated to the analysed items, we asked them to evaluate their knowledge of each studied PDC item *"Before"* attending the PDP course and *"After"*. The reason for this was to evaluate not only the progress but also the previous level of knowledge – based on the students self-assessment. This facilitated the answer to the first and the third research questions.

We plot in Figure 2 the detailed results based on students auto-evaluation for threads and processes general understanding and usage, and in Figure 3 the understanding of theoretical concepts of semaphores and monitors. Figure 4 summarizes the results for more topics, but showing only the average of the auto-evaluation. It can be seen that, for each topic, the students specified that a pre-existent knowledge existed (obtained through previous iterations), which has been improved (as it was expected).

During the PDP course the students were asked to deliver 5 laboratory works (practical assignments) on the following themes:

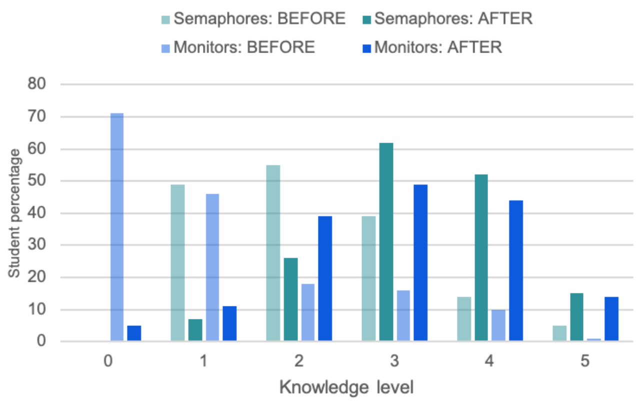

Figure 3: The before/after knowledge levels for general use of semaphores and monitors; based on students auto-evaluation

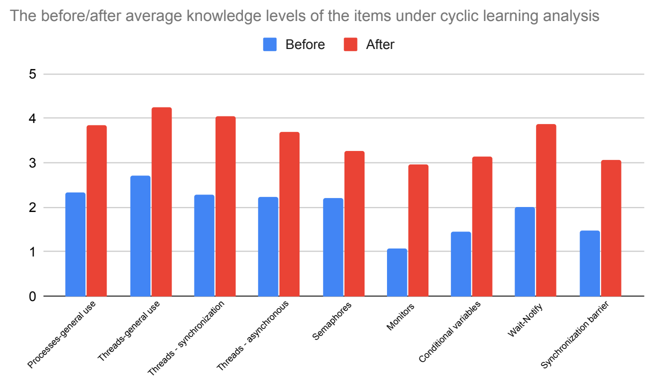

Figure 4: The before/after average knowledge levels of different topics under cyclic learning analysis; based on students auto-evaluation

(1) Multithreading: Adding and multiplying big numbers.
(2) MPI: Adding and multiplying big numbers.
(3) Adding large size polynomials represented using linked-list - different variants.
(4) Client-Server project with asynchronous tasks.
(5) CUDA project.

The students have been asked to specify the most difficult assignment (based on their personal opinion), and the results are given in Figure 5.

It is interesting to notice the fact that most of the students considered the assignment 4 - *Client-server* application to be the most difficult one. Client-server applications were discussed before PDP at several courses (CN - low level context, SDI and WP at a higher level context) and from this auto-evaluation we may deduce that the acquisition level before PDP course is at most at "Applying/Analysing" level. At the PDP course the "Client-Server" was only specified in the general discussion about patterns of distributed computing, but it was included into the assignments in order to follow the cyclic learning stream.

The first laboratory involves multithreading knowledge that was previously introduced in several iterations (algorithmic thinking was involved, too), but still there were 10% of the students that considered it the most difficult one .

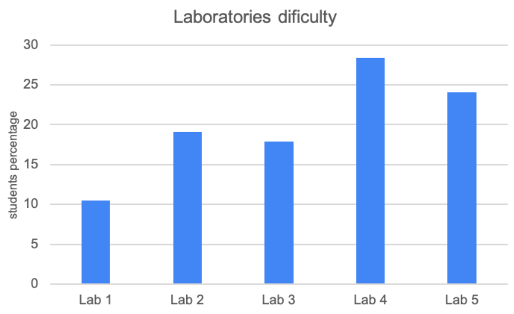

Figure 5: The students auto-evaluation regarding the most difficult laboratory theme.

Informal qualitative feedback: From free informal answers we found out that CUDA enjoys a large interest from students, but also MPI and OpenMP. This is promising since initially (when the course was introduced in 2015) the interest for MPI and OpenMP was not so high [11]. Also, MPI proves to be helpful also for deepening some more understanding of processes.

Some students expressed the fact that the PDC topics are difficult to understand, while very few others expressed their opinion that there is unnecessary repetition of some items.

5.2 Grade analysis

The evaluation for the PDP course is calculated as a weighted average between grades obtained on:

- Laboratory works (as describes in section 5.1);
- Theoretical written exam.

The theoretical written exam evaluates the understanding of the main synchronisation concepts and mechanisms together with the connected possible problems (deadlock, livelock,...), asynchronism through futures and promises, theoretical evaluation of the performance and task partitioning, parallel and distributed patterns.

It should be specified that the results for theoretical exam are lower than those that evaluates code interpretation or practical skills. Figure 6 emphasises these results.

Client-server applications is still considered difficult, even if it passed through several iterations. Beside the questionnaire answers, also the grade results emphasizes this - Figure7. A solution would be to apply the agile method of working in teams - similarly to the approach applied for CUDA laboratory (for this the results are also shown in Figure 7).

We have also analysed the grade results of MPI laboratory in two successive years: 2018 vs. 2019. The difference was that in 2019 the MPI presentation was done at the beginning

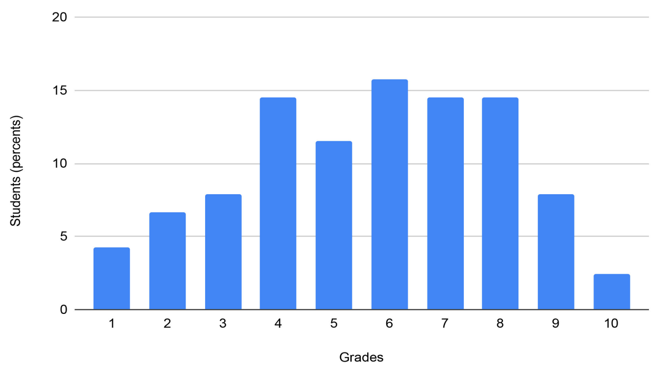

Figure 6: The grade results for the written exam.

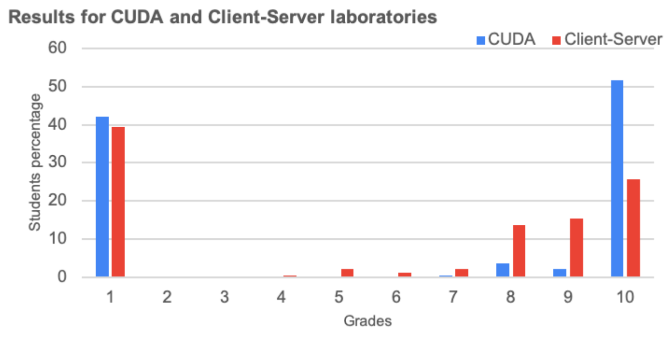

Figure 7: The grade results for CUDA and Client-Server laboratories.

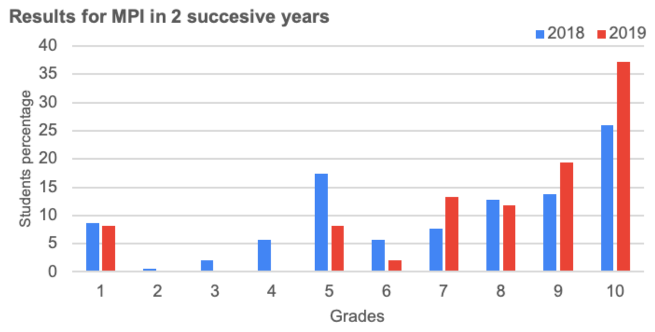

Figure 8: The grade results for MPI laboratory on 2 successive years (2018 vs 2019).

of the course – as explained in section 4.2. We noticed an improvement of the results, and the explanation could rely on the fact that starting with MPI that is a completely new topic (multithreading is there at the third iteration) increases the students' interest.

5.3 Results' interpretation

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

Response for Research Question 1. Since for PDC the knowledge acquisition needs repeated reinforcements, cyclic learning proved to be an effective and efficient method to be applied for teaching corresponding topics. This is due to the fact that the concepts and mechanisms of PDC are not very easy to be completely understood and assimilated (one course wouldn't be enough).

Response for Research Question 2. The analysis of the used agile methods shows a very good improvement in knowledge transfer, and also an increase of the students' interest in learning. These approaches should be used even much more in concrete use-cases.

Response for Research Question 3. The response to the first research question, even in the context of the positive answer of the second one, implies that we cannot increase very much the pace of introducing PDC topics at the undergraduate level. We have to assure the fact that the fundamentals are well understood and assimilated. Still, the elective courses could enlarge very much this knowledge for the students that have a special interest in this domain.

6 CONCLUSIONS

We have analyzed the need and the impact of using the cyclic and agile learning approaches in teaching PDC in the undergraduate Computer Science studies.

Several courses that form an interconnected chain of learning programming were transformed recently based on the new recommendations (ACM/IEEE and NSF/IEEE-TCPP) by introducing more information related to PDC topics. The correlation and well integration between these courses together with applying agile learning techniques are essential in assuring a good knowledge transfer.

Besides the quantitative analysis, the qualitative analysis is also 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 lead to some important educational research questions that could imply improvements of the educational process. We have conducted an analysis based on questionnaires and grade evaluation for more than 150 students, and the results show that PDC topics are considered difficult, and for teaching them it is appropriate to use the cyclic learning approach that is enhanced with agile leaning methods.

REFERENCES

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

[2] Scott William Ambler and Mark Lines. 2017. *An Executive's Guide to Disciplined Agile: Winning the Race to Business Agility*. CreateSpace Independent Publishing Platform. 225 pages.

[3] Benjamin S. Bloom, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain.* David McKay Company, New York.

[4] Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L. Lewis, Donna McGee Thompson, Charles Riedesel, and Errol Thompson. 2007. Developing a Computer Science-Specific Learning Taxonomy. *SIGCSE Bull.* 39, 4 (Dec. 2007), 152–170. https://doi.org/10.1145/1345375.1345438

[5] Poul H. Kyvsgård Hansen, Manuel Fradinho, Bjørn Andersen, and Paul Lefrere. 2009. Changing the Way We Learn: Towards Agile Learning and Cooperation. In *13th International Workshop of the IFIP WG 5.7 SIG*. Eidgenössische Technische Hochschule Zürich, Laboratorium für Lebensmittel-Verfahrenstechnik, 151–160.

[6] William Huitt. 2011. Bloom et al.'s taxonomy of the cognitive domain. *Educational Psychology Interactive* (2011).

[7] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA. 144–154 pages. https://doi.org/10.1145/2534860

[8] David R. Krathwohl. 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice* 41, 4 (2002), 212–218. https://doi.org/10.1207/s15430421tip4104_2

[9] J. Longmuß, B.P. Höhne, S. Bräutigam, A. Oberländer, and F. Schindler. 2016. Agile learning: Bridging the gap between industry and university. A model approach to embedded learning and competence development for the future workforce. In *Proceedings 44th SEFI Conference*. Tampere, Finland. Conference date: 12-15 September 2016.

[10] Anderson LW, Krathwohl DR, Airasian PW, Cruikshank KA, Richard Mayer, Pintrich PR, J. Raths, and Wittrock MC. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, New York.

[11] Virginia Niculescu and Darius Bufnea. 2018. Experience with Teaching PDC Topics into Babeş-Bolyai University's CS Courses. In *Euro-Par 2017: Parallel Processing Workshops*. Springer International Publishing, Santiago de Compostela, Spain, 240–251. https://doi.org/10.1007/978-3-319-75178-8_20

[12] Ingrid Noguera, Ana-Elena Guerrero-Roldán, and Ricard Masó. 2018. Collaborative agile learning in online environments: Strategies for improving team regulation and project management. *Computers & Education* 116 (2018), 110 – 129. https://doi.org/10.1016/j.compedu.2017.09.008

[13] Sushil K. Prasad et al. 2012. NSF/IEEE-TCPP Curriculum on Parallel and Distributed Computing - Core Topics for Undergraduates - Version I. http://cs.gsu.edu/~tcpp/curriculum/ Accessed: 15-Apr-2020.

[14] Henry E. Schaffer, Karen R. Young, Emily W. Ligon, and Diane D. Chapman. 2017. Automating Individualized Formative Feedback in Large Classes Based on a Directed Concept Graph. *Frontiers in Psychology* 8 (2017), 1–11. https://doi.org/10.3389/fpsyg.2017.00260