A Case Study of an Integrated Programming Course Based on PBL

Ayala L. Ribeiro
UEFS – State University of Feira de Santana
Feira de Santana, Bahia
Brazil 44036–900
ayalaedavi@gmail.com

Roberto A. Bittencourt
UEFS – State University of Feira de Santana
Feira de Santana, Bahia
Brazil 44036–900
roberto@uefs.br

Abstract—This Research to Practice Full Paper describes a case study of a course that uses problem-based learning (PBL) to integrate the subjects of Object-Oriented Programming, Data Structures and Software Design. We aimed to explore student learning when this teaching-learning approach was applied in the second term of a Computer Engineering undergraduate program. Data collection and analysis was qualitative, using semi-structured interviews and both open and axial coding. Results were organized in terms of qualitative themes describing course organization, problem conception, learned concepts and skills. This thorough exploration of such issues allows course developers to reflect on the impact of PBL and curriculum integration in student learning.

I. INTRODUCTION

Lack of motivation, failure and dropout are frequent issues in undergraduate computing courses, especially in introductory programming courses (CS1 and CS2). These courses often present high failure and dropout levels, making it hard for students to progress to their graduation [1]. Among the possible factors that contribute to those levels, we may list: i) lack of abstraction and logical reasoning skills to develop algorithmic solutions; ii) lack of student motivation, who often see the course as a major obstacle to overcome; and iii) teacher-centered learning approaches, which may not arouse student interest [2].

Some computing curricula introduce programming through the imperative paradigm in CS1 and later present the object-oriented paradigm in the sequence in CS2. This transition creates an additional problem for the cognitive conflict it raises in learners, which is usually time-consuming to solve [2], [3]. Associated with the previous factors, the curricular organization increases the complexity to acquire programming skills, generating lack of motivation and, consequently, difficulties to learning and knowledge retention.

Object-oriented programming (OOP) has become, in the last decades, the most influential programming paradigm, widely used both in education and in industry. However, learning OOP is not usually easy [4]. The difficulties may be caused by the complexity of the concepts to be learned in a short period of time, by the intrinsic complexity of these languages and by professional integrated development environments, aggravated by the use of teacher-centered learning approaches.

The use of active learning approaches is an alternative proposed by the community to address the difficulties of learning programming. Such is the case of Problem-Based Learning (PBL), a student-centered learning approach where an important part of the study happens in small groups that gather to solve problems that trigger curiosity and motivate students [5]. In PBL, learning is self-directed, based on reflection and on supporting raised questions. The use of PBL contributes to the acquisition of soft skills such as autonomy, initiative, communication, critical thinking, problem solving, group work, as well as the retention of knowledge and its application in different contexts [6], [7].

Some papers report the use of the PBL approach as an alternative to teach object-oriented programming [3], [8]–[10]. However, while those papers report their experiences in detail, an in-depth scientific assessment of the use of a PBL approach to teach OOP has not yet been pursued, especially with regard to learning outcomes.

The goal of this work was to evaluate student learning in an approach to teach object-oriented programming in the Computer Engineering program of the State University of Feira de Santana (UEFS), in an integrated Programming course in the second term of this program. The evaluation was carried out through qualitative research methods and using a case study research approach.

From the above goal, we try to answer the following research questions:

1) What elements of the PBL approach in an integrated programming course are fundamental to student success?

2) What results from the use of the PBL approach in an integrated programming course in terms of concept and skill acquisition?

The results of this investigation show that the curricular organization of the course, in which theory and practice go hand in hand, associated with the PBL approach, offer remarkable advantages for student learning. The conception of problems are among one of the biggest challenges. The practice experienced by solving the problems during PBL tutorial sessions as well as in lectures in the theoretical modules provide a smooth acquisition of both OOP concepts.
and programming skills. In addition, the approach allowed the acquisition of soft skills as well.

II. BACKGROUND

Here we describe issues related to learning OOP, foundations of PBL and its use in computing education.

A. Learning Object-Oriented Programming (OOP)

Learning programming may differ according to language paradigm [11]. Object-oriented languages usually pose more difficulties to novices than imperative languages when problems are longer, especially due to OOP learning curves.

Using object-oriented languages as the second paradigm to be learned has been debated by the community [12]. Authors argue that the natural complexity of OOP as well as the complexity from professional integrated development environments (IDEs) lead to important learning difficulties. Thus, they defend starting learning with imperative languages, followed by object-oriented languages.

The debate, however, is controversial. While Burton and Bruhn suggest that there is no evidence that CS1 courses taught with OO languages facilitate learning the craft of programming [13], K¨olling defends that starting with OOP concepts is beneficial, as long as appropriate teaching tools are used instead of professional IDEs [4].

K¨olling wrote a paper describing requirements for a beginner-appropriate programming language [4]. The language must be purely OO, easy to understand, simple, secure and high-level. Concepts learned should be easily transferred to other languages, and some issues relevant to professional software developers (e.g., efficiency) are irrelevant in the learning context. What learners need is a language supported by adequate teaching environments where learners focus on the craft of programming itself.

Environments such as Greenfoot [14], BlueJ [15] and Alice [16] are research results that aim to increase student engagement and reduce their difficulties. They also add a playful touch, making the learning of programming a fun experience.

B. Problem-Based Learning (PBL)

Problem-Based Learning (PBL) is one of the most popular active learning approaches in higher education [6]. It was developed at McMaster University in the context of medical education [17]. After some decades, its adoption in other fields of higher education has increased, especially in the fields of science and technology [6], [18]. In PBL, instructors behave as facilitators, while students actively pursue solutions to a given ill-defined problem inspired in the real life [6].

In PBL, students need to develop self-directed learning skills [7], [18] that lead them to problem solutions. They choose their own information sources, generate hypotheses for solving problems, discuss their learning issues in groups, and produce solutions according to their gained knowledge. Instructors, on the other hand, act as guides in group discussions, point learning resources that may help students, organize a group discussion structure, and aid students to remain on track.

Barrows developed a learning cycle where students try to understand the problem, generate hypotheses for solution, learn new knowledge and apply it to solve the problem [17]. Our Computer Engineering program has been using a variation of this learning cycle since 2003 [5], [19]. Basically, an ill-defined, inspired in the real life, problem triggers student motivation to learn: 1) A problem is handed out to students with no prior learning or preparation; 2) A group meeting led by students and facilitated by a tutor is used for brainstorming ideas and recalling facts; 3) Learning issues are raised by students, usually dealing with knowledge they do not have yet; 4) Students propose learning goals in order to test their hypotheses and solve their learning issues; 5) Students perform independent study between group meetings; 6) Students reconvene to discuss whatever they have learned; and 7) They repeat the previous steps in group meetings to navigate their way to problem solution. The cycle repeats according to schedule and problem duration, until they produce a solution to the problem.

PBL leads students to reach higher levels of cognition, moving from the lower levels of remembering, understanding and applying into the levels of analyzing, synthesizing and evaluating [6]. With PBL, students develop attitudes and values that are hard to acquire in a typical teacher-centered setting [7]. Moreover, the real-world context from PBL problems favors knowledge and skill transfer from school to the professional world [20]. Finally, a direct consequence of using the approach is the development of soft skills, such as problem solving, critical thinking, group skills, management, self-management and meta-cognitive skills [5], [6], [18].

C. PBL in Computing Education

The use of PBL in computing education has been previously reviewed by the community [21], [22]. O’Grady systematically reviews PBL in computing courses in a study that selected 63 primary studies from different computing disciplines [21]. He noticed, though, that apart from the large number of disciplines reached, PBL adoption is still superficial in computing curricula. Oliveira et al. realized, in a systematic mapping study that selected 52 papers, that most uses of PBL in computing happen in either software engineering or programming [22]. They also concluded that, regardless of PBL benefits, there are yet challenges to be overcome to reach more widespread adoption of the approach.

One of the first initiatives that used PBL to teach OOP happened in Australia [9]. Authors changed the CS1 course by moving from Pascal to Java, and adding a PBL approach. They summarize their three-year experience with the following results: students acquired both technical skills, such as programming in an OO language, and process skills, such as autonomy, critical thinking, group work skills, communication skills, independence and problem solving.

An integrated curriculum is usually mixed with problem-based approaches. Previous work in our university reports an experience of an integrated course of software design,
data structures and OOP that uses PBL as learning approach [3]. Authors describe two course offerings, and report the main learned lessons: acquisition of soft skills, course design respecting a gradual process related to learners’ capacities of assimilation, and the pitfalls of large inter-related projects leading to maintenance issues. Different from this paper, their focus is on the experience report, while here we focus on research questions related to learning with the PBL approach.

III. METHODOLOGY

Here we present the scenario, details about the case study and procedures for data collection and analysis.

A. Scenario

The case study was performed with Computer Engineering students from an undergraduate program in the State University of Feira de Santana, Bahia, Brazil. This program adopts PBL as its main learning approach since 2003. Furthermore, there is an integration between courses in its curriculum. Courses with content relations are offered together, as co-requisites, in the same academic term. They share challenges and opportunities, as well as student work [23].

The curriculum defines an integrated curricular component, named Integrated Study (IS), composed of one or more Theoretical Modules (TM) and an Integrator Module (IM). PBL unfolds in the IM course, working on contents similar to the TMs’ contents. In the IM, students are challenged with one or more comprehensive, real-world, ill-defined problems. To solve them, they must acquire knowledge relative to the TMs. On their turn, TMs are approached in a lecture-discussion format. In the TMs, students focus on learning content while, in the IM, they focus on learning deeper skills.

Our Computer Engineering curriculum has two integrated studies that deal with programming concepts. The first, the Algorithms IS, integrates concepts of algorithms, structured programming and basic data types (primitive variables, arrays and registers), and it is offered to freshmen. The following, the Programming IS, offered in the second term, integrates concepts of software design, OOP and advanced algorithms and data structures [3], [24]. At the end of the Programming IS, students must be capable of designing and developing object-oriented systems, by using appropriate data structures and algorithms, as well as simple software design concepts.

B. Participants

The case study was conducted during the second half of 2017. Classes were heterogeneous, made up of 46 freshman and sophomore students. Two instructors conducted the three theoretical modules (TMs), and four other instructors conducted four different Integrator Module (IM) groups.

C. Case study

The Programming IS is made up of three TMs and one IM. The TMs are Software Design, Data Structures and Object-Oriented Programming. Each TM has 30 hours of lectures/discussions in ordinary classrooms. The IM has 60 hours of groups meetings in special tutorial rooms. The TMs and the IM happen in parallel, with two weekly hours for each TM and four weekly hours for the IM, split into two 2-hour groups meetings. In the IM group meetings, students gather for self-directed group discussions, and follow the PBL learning cycle, while the tutor acts as a facilitator.

Each TM is split into three units. Instructors typically follow an order of concept presentation that follows the typical order students might follow in the IM, when they are trying to solve the problems. Nonetheless, there is no need for strict correspondence between them.

The Software Design TM deals with UML, domain models, design class diagrams, interaction diagrams, inheritance, composition, abstract data types, polymorphism, and some simple design patterns.

The Data Structures TM discusses concepts and implementation of data structures such as linked lists, queues, stacks, hash tables, trees and graphs, plus search and sort algorithms.

The Object-Oriented Programming IM introduces OOP in the Java language, involving the basics of classes and objects, additional concepts of exceptions, composition, inheritance, interfaces and polymorphism, as well as simple APIs such as collections, events and GUI programming.

The Programming IM, on the other hand, follows a plan dependent on the designed problems. Problems are conceived to respect a spiral of complexity that increases with each problem. Usually, there are four problems each term. In each of them, students learn new subject matter from the related TMs and develop technical programming skills, and additional soft/process skills. In this case study, the PBL problems are described in Table I.

The full planning and schedule for the IS, including the four used problems, are described in an experience report paper [24]. We also present, in the box in the following page, an instance of PBL problem used in this case study.

D. Data Collection and Analysis

To meet ethical issues and preserve the anonymity of participants, an informed consent term was signed by the students and instructors who wished to participate in this research.

The first author of this paper, who is not in the teaching team, used interviews as qualitative research instruments. Additionally, she made observations in class, following an observation protocol, taking notes based on research goals and questions, identifying student and teacher behavior, interaction between participants, and their interaction with the content. The main sources of analysis in this paper, however, are the semi-structured interviews with teachers that she conducted at the end of the term, using an interview guide of previously developed questions.

The data collected were analyzed using content analysis. We followed typical coding procedures, initially open coding and, then, axial coding, generating categories of interest. For each category, we wrote memos as results from the analysis. In this text, we use the sequential numbers P1, P2 and P3 to refer to the interviewed instructors.
Problem 4 – Road Trip Management with a Personal App

After some internships, you decide to leave for a business of your own. You’ve always wanted to start a business and you think the field of IT is ideal for that. Inspired by Google Trips (https://get.google.com/trips/), which is making the most success among travelers in the world, you remind all the trips you’ve ever done in your life. Being yourself a fan of the four wheels, you decide to make your own road travel app.

To gather requirements, you install and open Google Trips, and realize that it keeps various details about each city, such as name, latitude, longitude, area, population, city description, photos, things to do, places to eat and places to sleep. You think, “There are two types of information: the essential data about the city and the data associated with the city. I think I can start with the essential data by adding the associated data as I evolve my solution.”

Then you remember Google Maps, and recall that each city has several routes to allow you to transit between it and other cities. Worse yet, between each two cities, there are several possible paths, roads that fork, junctions connecting two or more highways. To simplify the whole thing, you decide to create a Point as the main entity, extending Point to two other classes: Intersection and City. An intersection would be a point that connects two or more roads. A city can also connect two or more cities and, in addition, has all those additional features that may interest travelers: places to eat, places to sleep, things to do. To make your life easier, you decide to work only with places to eat, leaving the rest for later.

You think a little more. Point is an abstract thing, which does not really exist. But intersection and city are things that really do exist. The distance between a point and a neighboring point is something easy to understand even though points are abstract things. Travellers would most likely ask the distance between an Intersection/City and another Intersection/City, but one can see them as points. Likewise, there are some things that only make sense to ask about cities (population, for example), whereas intersections may have additional attributes about the type of intersection (e.g., roundabout, crossing, traffic light) that do relate to a city. Your mind gets stuck and you decide to read more about abstraction in OOP.

Then, you are inspired once again be another project, EComp Maps, an open map project that wants to map all the streets and roads in every city in the world. The ideas from there help you create your app before you resolve to pay Google to use their maps. Of course, working with a smaller number of cities (maybe 15) and road stretches (maybe 100) to start testing your prototype. Looking at the data from EComp Maps, you see that there are two text files: one file with one line per city/intersection with location code, latitude and longitude, and another file with one line per road stretch, with the place of origin code, the place of destination code and the distance in kilometers between origin and destination.

One thing you notice is that the system administrator is the one who is going to register the data of each city, needing a graphical interface for that, as well as loading the map data and persisting the data of the whole system. Users, in turn, will just create their trips by making a very simple roadmap like Google Trips: choosing the first city, date of arrival and date of departure, adding the next city with arrival and departure dates for this city, and so on, until closing the trip with the last destination city. An user login would be a good choice, including a password hash, so that the password is not stored open in system files.

You think it would be great if the app offered the shortest route between every two cities in the route. You research a little and find out that someone has already thought about it, still in the twentieth century, inventing some cool algorithms for graphs. (Speaking of graphs, you have just read an article on a social network showing that an extremely efficient solution for representing graphs is by using hash tables to find the adjacent elements.)

After researching Google Trips once again to better understand the requirements, you decide to create some user stories and a domain model to help your developing your app.

In addition, before you start developing the graphical user interface, you decide to create the system façade and prepare the acceptance tests to ensure that everything works fine by letting the GUI access the system controllers.

Because your app need not be limited to the web, you decide to make the first prototype graphical interface by using one of the Java libraries to build graphical interfaces. You remember the various options (AWT, Swing, SWT) and decide which one will look more beautiful and be platform-independent.

IV. RESULTS AND DISCUSSION

We grouped the results into four themes, described in the following.

A. Course Organization

The organization of the Programming IS follows a spiral of increasing complexity, in which skills and knowledge are exercised, allowing the acquisition of skills. To realize that, contents are presented in the TMs to help students solve the problems proposed in the Programming IM. Simultaneously, the Programming IM presents practical problems based on real-world scenarios, involving concepts from the TMs. Such curricular organization allows the integration, where TMs and IM support each other. However, students face some difficulties. One of them is the excess of concepts to be learned in a short period of time, caused by the TMs’ small workload. Such difficulties may lead to student overload.

In parallel to the IM, the TMs present and discuss technical concepts that are usually linked to the theory needed to solve the problems, but also involving other concepts not addressed in the PBL problems. The concepts covered in the TMs help students to solve the problems proposed in the Programming IM. “The theoretical module helps a lot to introduce concepts. Students can have a much better perspective on where they need to focus. Then, in the IM, they focus on two or three hypotheses to solve. In the theoretical module, they enlarge horizons. Since they saw that in class, they can practice it in the IM...” (P2).

That also happens in the opposite direction. Students can learn contents that have not yet been discussed in the theoretical modules, but that are required in the IM problems. “It helps, I can realize that they are stimulated in the IM. And that consequently leads to learning in the theoretical course, because when they have to solve a problem, they have to look for a solution. Then, when they look for a solution, that stimulates to search new content, to learn a technique, to know how to solve a real problem in practice. Then, that stimulates them on demand.” (P3). This pedagogical organization in which theory and practice go hand in hand offers remarkable advantages, both from the point of view of enriching students’ cognitive structures and from the later recollection of knowledge learned, which makes learning a meaningful experience. “And here, in the IM, they kind of get in that mode of trial and error, trial and error, trial and error... This thing of making a mistake, making a mistake, helps in the process as well. Because when they see the right solution, they don’t forget it anymore. So, that helps a lot.” (P2).
### Course Plan for the Programming Integrator Module (IM)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Problem Topic</th>
<th>Learning Goals</th>
<th>Contents Addressed</th>
</tr>
</thead>
</table>
| 1       | Scheduling electoral Biometric Registration | Conceiving and developing a simple OO system from given software requirements and analysis. | 1. Classes and objects  
2. Fields, methods, and constructors  
3. Console Input and output in Java  
4. Interfaces in Java  
5. Basic MVC design pattern  
6. Domain model  
7. Design Class Diagram  
8. Abstract data types in data structures  
9. Linked Lists: Implementing Operations  
10. Iterator design pattern: implementation and use. |
| 2       | Electronic Auction Tool | Conceiving, developing and validating a software system based on reusing code, implementing data structures and designing software from given requirements and analysis. | 1. Constructors  
2. Method overload  
3. Object Composition  
4. Simple inheritance  
5. MVC design pattern  
6. Domain model  
7. Design class diagram  
8. Unit Tests  
9. Stacks, Queues, Priority Queues  
10. Sorting Algorithms |
| 3       | Stock Portfolio Manager | Conceiving, coding and validating a system based on advanced reuse techniques, appropriate data structures and disciplined design from given software requirements and analysis. | 1. Text files  
2. Binary files and serialization  
3. Handling Exceptions  
4. Façade design pattern  
5. MVC design pattern  
6. Unit Tests  
7. System and Acceptance tests  
8. Design Class Diagram  
9. Binary trees: representation and algorithms  
10. Balancing Binary Trees  
11. Documentation with Javadoc |
| 4       | Travel Management App | Analyzing, conceiving, coding and validating a GUI-based system from advanced reuse techniques, advanced data structures and disciplined software design. | 1. Graphical user interface components in Java  
2. Handling interface events in Java  
3. Collections from the standard Java library  
4. Abstract classes and polymorphic inheritance  
5. MVC design pattern  
6. Unit Tests  
7. System and Acceptance tests  
8. Design Class Diagram  
9. Hash Tables: Representation and Algorithms  
10. Graphs: representation and traversal  
11. Shortest path algorithm for graphs |

This curricular organization also poses some difficulties. One of them is the excess of concepts discussed in the theoretical modules in a short period of time. “And also, with short deadlines, it’s a dilemma that they face. They have to learn something in a short period of time, it’s a difficult rhythm to follow.” (P1). In general, the theoretical modules have a large amount of content for a relatively small workload. “The first thing that is important is that the Object-Oriented Programming module has a workload of 30 hours. That’s why theoretical part gets very condensed. So, since the content is very extensive, I have to focus especially in the theoretical issues, and I can’t ask for many practical exercises. If I did that, I couldn’t cover the theory.” (P3).

This problem is even more evident in the Data Structures TM that has an even greater content load when compared to the other TMs. “With Object-Oriented Programming, I think it’s smoother, you know, because things end up kind of repeating themselves. But, there, we work with programming techniques, which makes it easier, because the number of concepts they have to work on is even larger.” (P2). In addition, occasionally some concepts that are in the TMs’ syllabi are not as elaborated as they should for the lack of time. “The interaction diagrams, I mean, the communication diagram and the sequence diagram, I can’t work too deeply on those diagrams that are quite important to them, for mere lack of time.” (P2).

Those associated difficulties lead students to work overload. “I think it’s cool, I’m passionate about the IM, I’m passionate about PBL, actually, I am. I think this integration stuff is cool, even though I also think that the course workloads are too low, it ends up overwhelming students a lot.” (P2).
B. Problem Conception

Conceiving the problems is one of the biggest challenges because it can interfere with student motivation and, as a result, in student learning. The quality of the problems is directly related to some issues. The first issue is that the problem must be solvable as well as open-ended. In addition, the description of the scenario should be brief and with a level of complexity that allows group discussion. Finally, caution is advised in choosing the concepts and problem domain.

The learning goals chosen should be achievable with an open-ended problem, so that there is more than one workable solution. “I have an opinion about the problems. And each faculty has a different view. I am in favor of shorter, more open problem descriptions. Some instructors make the text longer, more detailed.” (P1). In addition, since problems are handled in multiple PBL tutorial sessions, they should involve more comprehensive contents.

According to instructors, the description of the scenario should neither be too long nor too short, it should have the ideal size. “The problem, when it’s leaner to read, gives more room for student decision. I prefer it this way.” (P1). Problems should have an appropriate level of complexity, sufficient to allow group discussion in the search for possible solutions. “I personally don’t like very much the complex problem type, because it gets stuck, students get very worried, much more than just understanding and meeting a number of requirements.” (P1).

The choice of concepts to be worked in each problem is a sensitive issue, and it must obey an increasing order of complexity. “In the beginning, in the early years here, we would put the graphical user interface in the beginning. They would suffer a lot, as they say, but they would practice a lot.” (P1). Another issue to consider is the choice of problem domain. The domain of the problem must be contextualized with the real world, allowing students to relate problem content with previous knowledge from students’ real lives. “What I see is the following: the more you’re able to work out a problem closer to the students’ experience, the better. So, for instance, problems that resemble experiences they had on the internet are more well received.” (P1). The choice of domains with playful themes provides greater student engagement to solve problems, since they increase the motivational level. “So, there’s something already in industry that they could associate with. So, the ones that motivate the most are those closer to students’ lives. If you do a thing involving shopping, travelling, music, games, they get involved. When there’s a problem about games, they get interested. So, this is an interesting indicator to think through.” (P2).

The more unfamiliar the problem domain is to students, the less motivating. “So when you create a problem away from their reality, it arouses less interest because of those things. Maybe if it were in another country where stock exchange was something more sought after, as an investment, maybe this kind of application in the problem would get more interest. I think it really was a very strange thing to them.” (P1). On the other hand, despite difficulties, an unknown domain allows the acquisition of knowledge previously unknown to students. “How I should do a stock portfolio, that whole thing was an unknown problem. It was a strange thing for them, but it was interesting for the sake of information, for them to get acquainted.” (P1).

C. Learned Concepts

Regarding learning of OOP concepts in the TMs and in the IM, results suggest that trying to solve the proposed problems in the PBL tutorial sessions provided advances in student learning. At the end of the term, most of them were able to create classes to represent real-world objects, their associated attributes and methods, despite the initial burden to deal with high level abstractions in some concepts. They mastered well the concepts of encapsulation, inheritance and polymorphism. Nonetheless, in some contexts, they had trouble to employ the concept of polymorphism. At first, they also had trouble with the concept of exceptions, but they mastered them in the course of the activities.

Initially, students had some trouble to acquire some concepts. However, we realized that, through their practice on searching solutions to the PBL problems, associated to the theoretical classes, they evolved in their learning. “So the trend for them was to begin with concept assimilation in a somewhat low level. Over time, the learning curve improves considerably […] If you don’t understand the basic concepts of object orientation, it is hard for you to understand a pattern, for instance, because it’s based on the concepts of interface, abstract class and polymorphism, for example. They are super important […]” (P2).

Since they had learned imperative programming in the prior term, one of their difficulties at the beginning of the term was that it took them some time to adapt to OOP concepts. “The first issue is to understand why it’s important to break a program into classes. Because it’s important to split it. And they always want to do everything together, they never think of the splitting thing. The second issue is to know in which class to put the fields and associated methods. They have some great trouble to learn how to define the scope of a class” (P3).

Moreover, they learned relatively well the concepts of encapsulation, inheritance and polymorphism. “Encapsulation, I think they learned it very well. The concept of inheritance, they managed to learn it very well. And the concept of polymorphism, they can learn it very well too.” (P3). However, even though students might have understood the concept of polymorphism, we realized that most students had a hard time knowing when to use it, and the advantages of using it. “A topic that I approach a lot, but I think it’s hard for students to understand, is the concept of polymorphism and knowing when to use it. I think it’s always hard, they understand what it is, but they don’t know when to use it. Sometimes they even understand what it is, but they don’t know why they should use it. ’Why should I use that? What’s the advantage?’ ” (P3).

Students showed some issues with exceptions, i.e., errors arisen during program execution. However, from the second
problem on, most of them had already known how to handle exceptions. “Exceptions, for instance, that’s not very easy. It’s not elementary but, from the second problem, they started to use them until the end.” (P1).

In the Data Structures TM, as well as in the other TMs, the concepts most exercised during their practice with PBL problems are the ones better acquired by students. In general, students have learned basic data structures such as queues, stacks and trees, as well as the concept of graphs and the simplest sorting algorithms. However, they had trouble with recursive sorting algorithms such as MergeSort.

It is important to emphasize that students had had a programming course with the imperative paradigm in the prior term. Therefore, they carry a previous load of concepts acquired in that context. As an example, we cite the concept of linked lists. “The linked list topic, because they’ve already come from the previous term knowing it, and right by the end of the first term. They’ve already seen a little of it with the C language. That helps.” (P1). Moreover, the concept of iterator was one of the most demanded during the PBL problems. Because of that, those concepts were well-founded. “[...] linked lists and iterators, I think they learned them well because they were the most required.” (P1).

They learned to create binary trees. However, most of them had issues with the concept of deleting nodes and balancing trees. “Perhaps the greatest issue was when there was a larger structure, and they had to balance the tree.” (P1). “I think they can figure out how to create a tree, maybe not deleting or shifting nodes...” (P3).

Most of the students had an easy time learning the concepts of graphs, especially the Dijkstra algorithm, a concept required to solve the last problem. “I think they can understand it well. With graphs, they had an easy time. We only addressed Dijkstra, which is basically navigating a graph.” (P3). They understood the simplest sorting algorithms such as BubbleSort, InsertionSort and SelectionSort. Most students learned the QuickSort sorting method despite the complexity of details and the use of recursion. “QuickSort is recursive but it is very simple because of the binary search. So, because of the binary search, I think they can understand it well.” (P3). On the other hand, they had a hard time to learn MergeSort and HeapSort. “I think with some sorting algorithms, they don’t understand them as well. For instance, Merge Sort, because it’s basically recursive, it’s the most recursive of all. And HeapSort, I think they had some trouble with it.” (P3).

In the Software Design TM, results showed that students had a hard time to interpret problems. We divided these more general difficulties into three categories: problem abstraction, business rules and the use of UML artifacts. Problem abstraction is related to the issues to understand problem domain, to identify system features and to document it. Business rules comprises identifying the business rules and the constraints and/or premises to solve the problems. In addition, they also had trouble to use UML artifacts, such as class and interaction diagrams and their relations.

The first obstacle students met was dealing with subjectivity during problem interpretation. They are usually strongly focused on coding. However, the Software Design TM requires additional skills to abstract problems that simulate a real scenario and, later, to represent them through design models using a common notation. “Initially, they show some resistance because when they start the undergraduate program, they think all they are going to do is just coding, coding and coding... when they get at the Software Design course, they are faced with reading, interpreting text, which is part of the process of creating a solution based on a programming language [...] if you don’t understand the problem, you can’t deliver the solution [...]” (P2).

Students also faced issues related to understanding and using the UML artifacts. Most of them had issues to identify and describe flows and rules and, consequently, to create diagrams. “It varies a lot from student to student. For instance, we always ask for the domain model diagram... the design class diagram as well, and each one has a different formulation... I think everyone has evolved, but I don’t believe that it was their best concept.” (P1).

D. Learned Skills

One of the research questions this study seeks to understand is the impact of the approach in terms of knowledge and skill acquisition. The PBL approach allowed the acquisition of both technical and process skills, the latter also known as soft skills.

1) Technical skills: Technical skills comprise a set of skills and competences directly related to the technical exercise, and involve particular knowledge, processes, methods and procedures. The PBL approach allowed the acquisition of skills and competences related to the practice of computer programming in an OOP language.

Results revealed that students were able to understand object-oriented modeling as well as related basic concepts. “Thinking of an object-oriented program, I think they learned, for instance, they come from an idea that every program is in a single file. When we get to object-oriented programming, they start to separate classes into files and try to group those classes.” (P3).

Students were able to apply object-oriented data structures to model simple and complex data from real problems. “I suppose they have also learned how to manipulate files in Java. We address that in object serialization in Java.” (P3). “[...] to understand what an exception is, when to use this exception, both in the exception hierarchy and in everything that is related to exception handling.” (P3). “They know that programming for the interface needs some layouts and, depending on the delta, they can make an interface better than another.” (P3).

They learned to design and implement the main data structures, knowing when each one should be used in a real-world problem. “... they know how to do basic data structures, create generic structures” (P3). They also learned to estimate the complexity of basic algorithms. “They know a little bit of algorithm complexity, not much, but they can look at an
algorithm and see if it’s going to be slow or fast, they see several examples with algorithm complexity.” (P3).

They learned the main search and sorting methods, as well as their application in different particular situations. “They know how to do some sorting algorithms like sorting a large collection of data. They understand the importance of having good algorithms to solve a problem, the right algorithm for a given problem, that is, not any algorithm that solves the problem.” (P3). They also understood and learned to manipulate advanced data structures such as trees, graphs and hash tables. “[...] they can manipulate a graph, they can manipulate a tree, and understand how it works. Besides, they understand and manipulate graphs and are able to develop search algorithms in directed and undirected graphs.” (P3).

They were able to search and identify existing solutions that suit each particular problem. “Knowing how to look at the problem in a different way and knowing how to find ready, reliable and already tested solutions to that problem that they are facing, so they don’t have to rework to think of an efficient solution to the problem.” (P2). And finally, in the Programming IM, students acquired skills related to their own experience of exercising some stages of software development: requirements, analysis, design, coding and testing. “The IM helps with practice. The IM is the opportunity they have to practice, to code, to write, to test the recursion, to see a bigger, real problem.” (P3).

2) Soft skills: The learning approach based on problems and projects allowed to acquire a series of soft skills. Among them, the ones who stand out are autonomy, self-learning, pro-activity, communication and interpersonal relationships. These behavioral and process traits generate benefits for students, increasing their chances of success in both professional and academic lives, since they are used to solving problems and facing new challenges. The approach, then, prepares students to develop the professional profile required by industry.

Autonomy is developed with PBL, since students are the main responsible for their learning. That is the opposite of the traditional teacher-centered approaches, where students play a passive role in their learning process, depending very much on the teacher, who plays an active role to mediate knowledge. “Students acquire so... an independence in the way they study and build their own knowledge. They don’t depend on the teacher because in the traditional method, the teacher is the source of the information.” (P1).

Self-learning is stimulated because students are encouraged to learn something without having a teacher lecturing them. “PBL compensates for that in the following way. If they ever need that content they did not learn, they go after it. Then, in the end, they don’t lose it, because they know how to get the knowledge, even if they don’t have it. Knowing how to get it is more important.” (P1). With their particular efforts, students themselves intuit, seek and research the materials needed for their learning, exercising proactivity. “They seek knowledge, seek the references from the books [...]” (P1).

Furthermore, there is the development of oral communication provided by the very dynamics of the PBL tutorial sessions, in which students need to dialogue and discuss possible solutions to problems. “Then, students who have issues to communicate, and are sometimes withdrawn, shy or such, they have an opportunity to develop those skills, at least to improve those important qualities of communication in different ways, such as attending a meeting, exposing their ideas and their opinions [...]” (P1).

Finally, the PBL approach allows improving interpersonal relationships. Discussions on possible solutions to problems unfold within tutorial sessions and also outside of class. Students cooperate to build their knowledge, learn to express ideas and also listen to their peers, to accept their classmates’ opinions. With that, they develop mutual respect. (“Studying in groups is like that, we don’t have a follow-up tool, but we realize they do it. Another cool thing in PBL is that because this group study is stimulated, you end up with less individualism.” (P1).

V. CONCLUSIONS

This paper presented a qualitative case study of a PBL-based approach to teach OOP, data structures and software design to Computer Engineering undergraduates during the second term of 2017. The results are described from the perspective of four themes: course organization, problem conception, learned concepts, and learned skills, both technical and soft skills.

The results show that the curricular organization of the course, in which theory and practice go hand in hand, associated with the PBL approach, offer remarkable advantages from the point of view of enriching students’ cognitive structures, as well as from content retention, by making learning a meaningful experience. In addition, we found that conceiving problems as learning triggers is among one of the greatest challenges. The description of the scenario in a good problem should have the ideal size, and be ill-defined in a way that entails discussion, and making assumptions and decisions. In addition, the problem must have a viable solution in the available time, the introduction of new concepts must obey a gradual and increasing order of complexity, and the domain must be contextualized within students’ previous experiences.

The practical experience acquired when solving the problems in the Programming IM as well as theoretical knowledge acquired during lectures/discussions in the theoretical modules provided a gradual learning of the contents and skills. Despite difficulties with some concepts, students in general reached the general goal of the Programming integrated study. Students were able to design and develop object-oriented software, appropriately using algorithms and data structures, with mastery of the fundamentals underlying the methodologies and tools used. In addition, they developed process skills such as autonomy, self-learning, pro-activity, communication and interpersonal relationships. These skills add an important differential for these students in the marketplace.

In future work, we intend to analyze more quantitative aspects regarding student motivation as well as to measure the relation between student motivation and performance.
ACKNOWLEDGMENT
The authors would like to thank both faculty and students who voluntarily took part in this work. This work was funded by FAPESP, through grant 2015/24331-1.

REFERENCES