Object-First Approaches Programming Teaching

Rostislav Fojtik

Abstract: *Teaching algorithmization and programming has been recently going through big changes trying to react to the dynamic development of software industry. Methodical process, development models, or programming languages previously used do not conform to current requirements. The paper is based on practical experience of the authors teaching programming on secondary and tertiary level of education. The aim is to show suitable methods in teaching of object-oriented programming. It presents the most frequent problems teaching programming. It shows common misconceptions and offers solution for them.*

Key words: *Computer Science Education, Object-First Paradigm, Object-Oriented Programming, Programming Teaching.*

INTRODUCTION

Teachers often ask which programming language to start with. This approach is, however, unsuitable. The first question when creating a programming course should not deal with a programming language, but with selection of a suitable paradigm and the objective of the course. If the objective is to teach students to create algorithms and think logically, the most suitable, particularly for students of primary schools, seems to be microworlds, such as Karel or Logo. These tools are well-arranged, simple, and demonstrative. The students usually see their results of the advance in a graphical visualisation. The tools do not take long to learn the language rules and users can really focus on creating algorithms and simple programs and they do not wander in language complexities and concrete details. However, if the objective is to learn students to program so that their knowledge and skills could be used in practice, first of all we have to choose a suitable paradigm. Currently, the most spread software development approach is object-oriented.

SUITABLE METHODS FOR TEACHING PROGRAMMING

Teaching programming does not follow modern methods in most of secondary schools. Majority of the teachers teach the same or similar way they were taught at the time of their studies. The problem is that current requirements on program development have considerably changed. Once the most common procedural paradigm and approach imperative-first or algorithm-first are not enough and it is necessary to aim at other methodological educational procedures. It can be primarily seen in currently very frequently used object-oriented programming. Lots of teachers apply the classical procedural approach, and then they proceed to the object-oriented approach. Yet, this sequence makes students create unwanted habits, which are difficult to remove [1].

If we compare books on programming, syllabi, and educational plans of the subjects where programming should be taught, we find out that it mostly concerns education of a concrete language. The introductory passages of books or educational courses describe data types. Numerous pages enlist basic control structures, work with fields, access to files, creation of program modules, etc. Description of all rules of a concrete language takes dozens or hundreds of pages or several initial lessons. Analysis or design of a program, design patterns, testing, and suitable programming procedures are mentioned only marginally or not at all [5]. There is no space! Instead of programming coursebooks, those are rather reference manuals for a particular programming language. To the detriment of the subject matter, programming is often taught in the same way. The main emphasis is put on managing syntactic rules and difficulties of the selected programming language [3].

PROGRAMMING LANGUAGE SELECTION

Selection of a suitable programming language is an important step in preparation of educational course. It is necessary to select a language conforming to the object-oriented paradigm. Most languages do not have object attributes implemented very suitably (e.g. Pascal). On the other hand, there are languages that are of high-quality from the OOP point of view, but they are less used in practice (Smalltalk).

Programming languages can be divided into two groups:

- pure OOP languages
- hybrid OOP languages

Programming languages from the first group use only the object-oriented model. We cannot create independent functions that are not a part of the class. We also do not use global variables declaration. We can classify here languages Smalltalk or Eiffel. A disadvantage of programming language Smalltalk is less frequent use in practical projects. This language gives a basis for Objective-C, which serves as a basic tool of programming in operating system Mac OS X [2]. Unfortunately, it is unusable in other systems. In addition, syntax of these languages differs from a large number of popular programming languages. Objective-C code example:

```
MyClass *myObject = [[MyClass alloc] init];
[myObject setAttribute: 1];
```

C# or Java code:

```
MyClass myObject = new MyClass();
myObject.setAttribute(1);
```

On the other hand, hybrid programming languages use as a model object-oriented, procedural, and imperative models. Thus they merge various paradigms and it is up to the programmer which approach will be preferred. A typical representative of this group is programming language C++, which builds on a purely non-object approach of language C and its object extension.

Another division of programming languages depends on the extent of type check. So-called static languages are based on knowledge of data types and they perform more frequent type checks in the compilation time. C++ can be again taken as an example. Dynamic languages (such as Smalltalk or Ruby) have weaker knowledge of types in a part of variables declaration and they usually carry out checks during the programme runtime. Dynamic languages are usually interpreted.

Programming language C++ is not suitable for an introductory course. It concerns a relatively complicated language, which does not have strong control mechanisms. Work with pointers requires its profound knowledge. If not, the program can contain less visible errors. It concerns a language that enables the programmer almost everything, but it presumes that the programmer knows exactly what he is doing. Moreover it belongs to so-called hybrid languages, i.e. languages connecting procedural and object-oriented approach. Programs can be created according to one or the other paradigm, or we can even use both approaches in one code.

Suitable languages for an initial course of object-oriented programming can be languages Java or C#. Syntactic rules are very similar and currently used in other programming and scripting languages.

Usual mistakes of students learning a programming language by classical approach followed by object one:

- students think about the code – language, not about the problem solution
- unsuitable use of global variables:

```
Type globalObject;
class MyClass
```

```
{
      public void Method()
      {
          globalObject = change;
      }
}
```

- students tend to create monolithic solutions where they connect logics, structure and data storage with user interface
- trying to solve the most detailed problems right from the start
- creating a large class with lots of competences
- classes contain unsuitable competences

A typical example is creation of method for writing information on the screen for classes that should solve other functionality than a write on the screen:

```
class Vehicle
{
      private int enginePower;
      //other attributes

      public void WriteOnScreen()
      {
            Console.Write("Engi ne Power "=+ engi nePower);
      }
      //other methods
}
```

PEDAGOGICAL EXPERIMENT

At the turn of 2009 and 2010, the Department of Informatics and Computer Science repeated the experiment with education of programming of selected secondary-schools students. There were three courses prepared for the students. Each course was attended by 12-28 secondary-school students. Their knowledge of programming was at different levels. A third part of students had not learnt any programming language before. The rest knew basics of language C or Pascal. Basics of object-oriented programming were known by few students. Object-oriented approach was familiar only to students who used developing tool Borland Delphi at their secondary school. But they knew little about classes or objects as well. The main emphasis in their education was put on the use of graphical components and design of graphical interface of the application.

The objective of the educational courses was to familiarise the students with object-oriented programming and creation of programs based on this paradigm. Object-First methodology was used for the training [6].

When deciding about the selection of a programming language, Java was selected for two courses and C# for the third. Their characteristic conforms best to current education.

- It enables object-oriented approach
- The possibility to use the language in practice is motivating for students. Most students do not like to work with programming languages that are purely designed for learning or research
- The languages are easier than C++
- The syntax of the language is similar as in C, C++ and several others frequently used programming or scripting languages.
- Java enables to use environment BlueJ to graphically visualise classes and their relations

- Both languages have available free development environment. The following tools are used in the courses: NetBeans and MS Visual C# Express Edition.

The course had twenty lessons of present education. Students could also use LMS Moodle (examples, textbook, video, communication). The basis for qualitative education is a suitably designed structure of the lesson. The initial stage was always dedicated to revision of the last subject matter, or solving students' questions. Then there was an introduction to a new problem. This was not taught by means of a theoretical explanation by the teacher, but by solving concrete programs and problems. The students were presented with examples and tasks whose solution supported a topic under consideration. The aim was not to create complex, perfect programs, but to show possible procedures; often at the cost of large simplification of the problem and subsequent solution. Creation of complex and extensive codes would not have been suitable and it would have been difficult to realise in such a small time-allocation. On the contrary, big emphasis was put on discussion and communication among all participants. When solving individual tasks, they used a graphical visualisation of the problem and then they proceeded to code-writing itself. Graphical visualisation is very important and it makes understanding of the solution easier for the student.

The first lesson was mainly focused on explanation of basic characteristics of object-oriented programming and its differences from procedural approach followed by a brief description of programming language Java and C#, its brief history, characteristics, advantages and disadvantages, and an overview of certain development environments suitable for education and professional application.

The content of the course was the following:

- Class and object
- Methods
- Alternative and iteration
- Inheritance
- Assembling and composition
- Array, library, collection
- Program design

An important role is played by suitable chosen examples and problem tasks. Their preparation and primarily making up is really difficult. Examples that are solved by the students in the lessons must often meet contradictory requirements [4].

- They must be complex enough to sufficiently cover the solved area
- They must serve as a pattern for solution of real problems
- They must be at least a bit connected with real use. It should not be mere "examples for examples".
- Their solution must not be extensive and complicated so that they could not be solved in the lesson time.

The created solution must reflect a topic under consideration. Other parts of the code should be as simple as possible. The above-described requirements lay high demands on pedagogues. The solved examples and tasks can highly motive or totally put the students off. Despite the fact that the attendance at the seminars was not compulsory, attendance of majority of the participants was surprisingly good.

What did the students learn?

They did not perfectly learn the language code, all of its rules and laws. Is it an omission? Usual education at majority of secondary schools and at certain universities goes rather the opposite direction. The first lessons are usually focused on a large number of syntactic rules, overview of a huge number of data types, concrete functions, operators, libraries, etc. Students are then flooded by information in which they get lost. When they are to create a concrete program, they are not able to think about its solution, but most of the energy is devoted to handle the syntax and development environment. After a while they are able to write the code correctly, but not to assemble the algorithm, correctly define classes, objects, their relations. They can learn the technique of code creation on their own later.

CONCLUSIONS

The object-oriented approach in teaching programming can be successfully used not only at universities, but at lower level of educational institutions as well, and not only as a supplement, but as the principal programming paradigm. However, it is very important to set a suitable methodology of education. An important role in this process is played by clearness, adequacy, and quality of examples. Last but not least, students must be suitably motivated.

REFERENCES

[1] Bennedsen J., Schulte C. What does “Objects-First” Mean? An International Study of Teachers' Perceptions of Objects-First, 2008, http://crpit.com/confpapers/CRPITV88Bennedsen.pdf [online]

[2] Kochan S. G. Programming in Objective-C 2.0, Addison-Wesley Professional, 2009, ISBN: 0321566157

[3] Pecinovský R. Začlenění návrhových vzorů do výuky programování. Objekty 2005 - sborník konference. Ostrava 2005. ISBN 80-248-0595-2.

[4] Pecinovský R. Výuka objektově orientovaného programování žáků základních a středních škol, Objekty 2003 – sborník konference, Ostrava 2003, ISBN 80-248-0274-0.

[5] Pecinovský R. Výuka programování podle metodiky Design Patterns First**,** Tvorba software 2006, ISBN 80-248-1082-4

[6] Sajaniemi J., Chenglie Hu, Teaching Programming: Going beyond “Objects First”, ftp://cs.joensuu.fi/pub/Reports/A-2006-1.pdf [online]

ABOUT THE AUTHOR

Rostislav Fojtik, PhD, Department of Computer Science, Faculty of Science, University of Ostrava, Czech Republic, 30.dubna 22, Ostrava, rostislav.fojtik@osu.cz, Phone: +420597092173