Cognitive Aspects of Object-Oriented Programming

Andreas Schwill
Fachbereich Mathematik/Informatik - Universität Paderborn
D-33095 Paderborn - Germany
email: schwill@uni-paderborn.de

Abstract

Computer science education in schools is still mainly based on the imperative programming paradigm via Pascal, but there are also many proposals how to teach the functional, predicative or object-oriented style. All these approaches share the (reasonable) implicit assumption that one or two of the non-imperative paradigms have to be included into lessons in order to give students a correct view of current computer science activities. Most authors, however, leave open which paradigm to start with; moreover, they do not consider students' psychological prerequisites necessary to comprehend these paradigms. In this paper we will focus on the object-oriented paradigm solely from the learner's perspective and show that this paradigm is consistent with the natural way of human thinking, and therefore particularly suitable for teaching computer science at an introductory level.

Keywords: Cognition, Programming

Programming styles in computer science education - A new edition of the language controversy?

In the seventies it was disputed with passion which programming language is most suitable for computer science education in schools. This language controversy was settled by stressing in lessons the „correct way of thinking"; the final transfer of a solution obtained by the correct way of thinking into a programming language was considered merely a technical process that could not negatively influence the systematic progress of the students any more (ignoring the Sapir-Whorf thesis).

The current situation is somewhat similar to the former with the matter in dispute lying on a metalevel instead: What is the correct way of thinking embodied by the corresponding programming paradigm, i.e. is it imperative, functional, predicative or object-oriented thinking? Presumably, this controversy may be settled as before by shifting it to a metalevel and thus suspending it until again several metalevels have been identified and discussion concerning the best metalevel will be resumed.

While computer science education in school is still dominated by the imperative paradigm, there are several approaches to introduce also the other three paradigms. Most authors argue that these paradigms play or will play such an important role in software development that they have to be included, at least in part, into lessons in order to give students a correct view of the current evolution of computer science. This generally accepted claim, however, leaves open which language or

paradigm seems most suitable at the introductory level, since it does not take into account that different paradigms require different intellectual prerequisites and abilities of students as well as different curricular approaches which may hinder lessons to some extent, as shown by the following example.

Example
Introductory education in imperative programming using Pascal is often unsatisfying, since students have to learn many different language concepts in advance before they are able to write a nontrivial program. This situation makes great demands on the teacher in keeping students motivated.
With a predicative approach using Prolog this seems not a problem, since Prolog's few concepts will enable students to write powerful programs even at an early stage. However, Yazdani [6] has shown that programming in Prolog needs high cognitive abilities with respect to logical thinking, modelling parts of the real world by logic and comprehending Prolog's virtual machine.

In this paper we favor an object-oriented approach to introduce into computer science for two reasons. On the one hand this paradigm meets the demands for a modern education with powerful concepts, such as encapsulation, inheritance, evolutionary software development by extension, adaptation and reconfiguration of solutions already available etc., and on the other hand it seems in line with cognitive processes that are performed in the human brain during perceiving, thinking and problem solving, as will be shown in the rest of the paper.

Objects in mind and programming
Adults as well as little children show the typical human behaviour to judge things by what they can be used to. A screwdriver, for example, is a tool for handling screws; as an instance of a class with more general properties like „long" and „sharp" it may also be used as a crowbar, chisel, drill or stabbing weapon. Conversely, one distinguishes several subclasses with more specific properties: screwdrivers for cheese-head screws, for cross-head screws, with equipment to check the electric tension etc.
There has been much research on childrens' [5] and adults' [2] perception of objects and representation of knowledge in memory and how this knowledge is used to guide one's decisions and actions. All these results reveal that identification of objects seems to depend more on actions that are possible with them than on their nature such as color or shape. This behaviour of humans is illustrated in Tab. 1 which shows the analogy between concepts of object-oriented programming and a standard model of cognitive psychology based on *categories* [4] or *schemas* [1] being large, complex units that organize much of human knowledge and behaviour. From the programming perspective these units may be considered as classes.

Example: Duncker's candle problem and its object-oriented interpretation
The following experiment [3] illustrates these results: Several subjects alone in a room had to solve the problem to fix three candles at a wall and enlight them. On a table there were few things that the

subjects could use to solve the problem. Among these mostly useless things there were also some that could be used for the solution: tacks, matches and three little cardboard boxes about the size of a matchbox. Solution of the problem: Each cardboard box is tacked to the wall and serves as a basis for the candles. Then the candles are enlighted and fixed on the boxes with some wax.
Subjects had to solve this problem starting from two slightly different situations: For persons of the first group the cardboard boxes were filled with the experimental material, the first one with candles, the second with matches and the third with tacks. For persons of the second group the boxes were empty and the material was scattered on the table.
Surprisingly, the second group solved the problem more often and more quickly than the first group. Duncker explains this observation as follows: The first group perceives the boxes as containers for candles, tacks and matches. Subsequently, this function „container“ is so closely bound to the boxes that subjects can hardly loose their thinking from it and are unable to use the boxes for different purposes, namely as a place for the candles. Duncker calls this phenomenon *functional fixedness*. The second group perceives the boxes unbound to any particular function and is thus able to use them freely even for unusual purposes.
From the point of view of computer science subjects' thinking is obviously object-oriented: First of all objects are divided into classes depending on what operations they permit. A box is considered by the first group as an object belonging to a class „box“ with operations like „open“, „close“, „is empty“ or „is full“. These operations determine the subsequent thinking and inhibit this group to recognize that boxes also belong to a superclass with more general properties like „flat“ and „cuboid-shaped“ and operations such as „stack“ or „can things lie on it“. Fig. 1 shows a definition following Oberon, Fig. 2 a treelike representation of the class hierarchy of the boxes which subjects might probably have had in mind. Now, functional fixedness may be considered as the lack of intellectual ability to switch between different levels of the hierarchy.

Recommendations for computer science education

In this paper we wished to motivate that object-oriented programming might be a better approach for teaching computer science at an introductory level, since it reflects fundamental cognitive processes of humans. However, it is not enough to just start education with an object-oriented language. Learning outcome also depends heavily on the programming environment that should make visible the object-oriented approach, i.e. provide clear visualizations of objects which students can analyze interactively and playfully for what can be done with them, as well as manipulate, combine, reconfigurate and extend. Systems that come close to this philosophy are, for instance, HyperCard at the introductory level and Smalltalk-80 at a more advanced level.
Further empirical research with respect to the precise psychological prerequisites, the best language and its programming environment and the curricular approach are necessary.

References

[1] Anderson, J.R.: Cognitive psychology and its implications. Freeman 1980
[2] Bruner, J.S.; Goodnow, J.J.; Austin, G.A.: A study of thinking. Wiley 1956

[3] Duncker, K.: Zur Psychologie des produktiven Denkens. Springer 1966
[4] Glass, A.L.; Holyoak, K.J.; Santa, J.L.: Cognition. Addison-Wesley 1979
[5] Piaget, J.: The origins of intelligence in children. Intern. University Press 1952
[6] Yazdani, M.: Artificial intelligence, powerful ideas and children's learning. Computers, Cognition and Development (J. Rutkowska, C. Crook, eds.) (1987) 99-114

object-oriented view	**Example**	**psychological view**
class	dog	category/schema
definition by attributes:		definition by attributes:
- variables	has 4 legs	- perceptual attributes
- methods	can bark	- functional attributes
- inheritance		- relational attributes
-- single	A dog is not a cat	-- categories may be mutually exclusive
-- multiple	dogs and cats may be pets	-- categories may overlap

Tab. 1

```
type object = record
      has a spatial extension;
      you can take hold of it;
      has color and shape;
      ...
end;

type cuboid-shaped = record (object)
      has length, width, height;
      ...
end;

type flat-cuboid = record (cuboid-shaped)
      height≤5cm;
      can be stacked;
      you can lie something on it;
      ...
end;

type box = record (flat-cuboid)
      can be opened;
      can be closed;
      is empty or filled with ...;
      ...
end.
```

Fig. 1

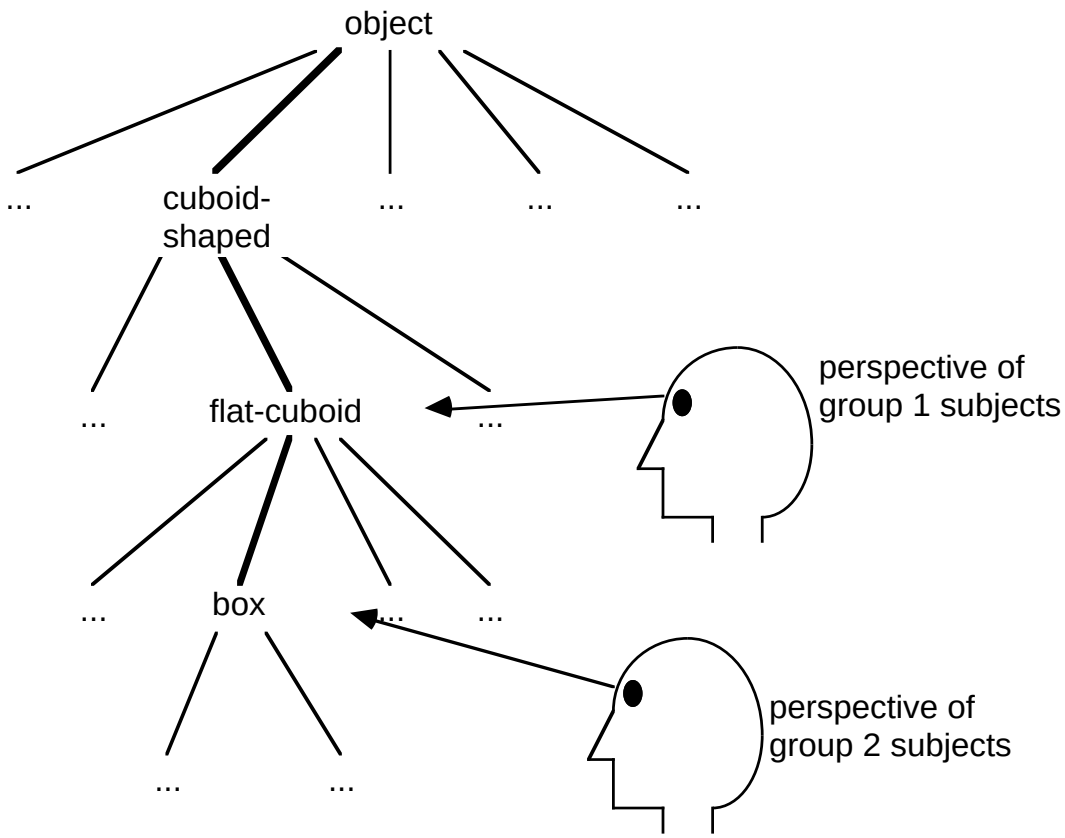
object
...
cuboid-
shaped
...
...
...
...
flat-cuboid
...
perspective of
group 1 subjects
...
box
...
...
...
...
perspective of
group 2 subjects

Fig. 2