A smooth way towards object oriented programming in secondary schools

Peter Hubwieser Peter.Hubwieser@in.tum.de
Fakultät für Informatik der Technischen Universität München, Boltzmannstr. 3, 85748 Garching, Germany

Abstract
The paper presents an analysis of the learning objectives that are to be achieved if Object Oriented Programming should start with a program that fulfils the demands of modern teaching approaches. The solution of the problem lies in the arrangement of many objectives in the parts of the course preceding OOP. The paper presents an appropriate teaching concept that will be carried out within the new subject, informatics that started in 2003 at the Bavarian Gymnasiums.

Keywords
Object oriented modelling, object oriented programming, informatics at secondary schools, learning objectives, didactical dilemma

INTRODUCTION
This paper might be regarded as the continuation of a series of international publications (Hubwieser et al. (1997), Hubwieser and Broy (1999), Müller and Hubwieser (2000), Hubwieser (2002), Hubwieser (2004), Schneider 2005, Hubwieser 2006) reporting about the recently started mandatory subject of informatics at the Gymnasiums (which is a type of secondary schools that directly leads to the University) in the state of Bavaria (which is one of the 16 states of the Federal Republic of Germany). The conception of this subject was worked out mainly by my team at the Technische Universität München.

Whereas the former publications mainly covered the curriculum details, we are going to shift the focus now towards the first steps of the evaluation of our meanwhile implemented concept. The goals of this evaluation process are to get insights into the quality of the implementation of our concept as well as to detect weak points of the concept itself. This means that we have to examine the learning outcomes of the students correlated to the learning objectives we have set to the educational process. Additionally we expect some general results about learning processes in informatics. As Bavaria is running about 400 Gymnasiums with about 290.000 students aged from 10 to 19 years, we have access to a very large sample.

One of the hardest challenges we faced during the conception phase was to design the part of the course that covers Object Oriented Modelling and Programming (from now on shortly OOM and OOP, respectively) in grade 10. The main obstacle was in the big number of learning objectives that had to be achieved simultaneously, if the programming course starts with a relatively complex program, which is postulated by modern teaching approaches.

We solved this problem by distributing the learning objectives over the parts of the course that precede the OO-part. In my opinion we found a way to teach the basic concepts of OOP step-by-step, one after the other and thereby avoiding confronting the students with too many unknown ideas simultaneously.

The description of the learning objectives that we want to achieve and their distribution over the course will be the focus of this paper. As the reader might not be familiar with the organisational structure and the basic conditions of the new
subject, the paper start with a short overview, although this information might be found in some of the preceding publications, (e.g., Hubwieser, 2006).

GLOBAL OBJECTIVES
First of all, we should explain our objectives, which correspond to the widespread perception of general education. Concerning the generality of the objectives we distinguish (following Anderson and Krathwohl, 2001) between

- Global objectives: “Complex, multifaceted learning outcomes that require substantial time and instruction to accomplish”;
- Educational objectives: derived from global objectives by breaking “them down into more focused, delimited form”;
- Instructional Objectives, with the purpose “to focus teaching and testing on narrow, day-today slices of learning in fairly specific content areas”.

During the course of lessons we designed the students should achieve the following global objectives. They should:

- Acquire the capability of independent opinion and responsible acting in the information society,
- Be able to act responsibly and efficiently in a world of work and profession that is ubiquitously penetrated by IT,
- Master effectively the tools and understand the limitations, chances and risks of information technology,
- Learn the responsible, efficient usage of information technology based on knowledge of the theoretical foundations and basic principles of the systems,
- Master complex systems, particularly being able to describe their structure and behavior and communicate about them in a competent way,
- Be prepared for the application of information technology in other school subjects,
- Be able to choose their career based on a sufficient knowledge of the possibilities and principal limitations of future IT developments.

Also following Anderson and Krathwohl (2001) we regard learning objectives as a combination of a certain type of knowledge and an observable behaviour (called cognitive process) concerning this type of knowledge, forming the two dimensions knowledge dimension and cognitive process dimension, of their Revision of Bloom’s taxonomy (see Table 1 below).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Factual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. Conceptual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. Procedural</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. Metacognitive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Revision of Bloom’s Taxonomy

A closer look at the global objectives of our course of lessons I listed above uncovers that students have to achieve learning objectives in all categories of the cognitive process dimension following the taxonomy of Anderson and Krathwohl (2001), particularly in the three highest ones:

- They will have to analyze problems in order to represent them properly by an object-oriented model.
- They will have to evaluate alternative models in order to choose one of them.
- They will have to create models and programs out of their models.
Concerning the knowledge dimension we face also objectives of all categories, listed with one example each:

- **Factual**: syntax elements of a programming language,
- **Conceptual**: structural elements of algorithms like conditional statement,
- **Procedural**: developing a Java program,
- **Metacognitive**: mastering the modeling and programming process as a whole.

Summarizing these considerations about the learning objectives we see that teaching OOP is a very demanding task which covers all the highest categories in the taxonomy of learning objectives.

**ORGANIZATIONAL OVERVIEW**

Basically there are four possible structural implementations of Informatics courses in school curricula:

- **Compulsory** subjects: all students of a certain grade or at least all students of a certain direction of study have to attend the course that is explicitly visible in the timetable. Examples are German Language, Mathematics or Geography.
- **Optional** subjects: the students are offered a variety of subjects additionally to the compulsory ones. They may choose none, one or more of them, e.g. modern dance, chorus or rock climbing.
- As a compromise between these two extremes there are **compulsory optional** subjects, from which the students have to take (at least) one, of the foreign languages, e.g., French, Spain or Italian.
- **Integration**: some topics might be taught integrated into other subjects, e.g. traffic education into the subjects physics and geography.

As pointed out in many publications (e.g., Hubwieser & Broy, 1999), we have definitely the opinion that it is necessary to teach informatics within a compulsory subject. In contrary to some misunderstandings, this should take place *additionally* to the widespread *use* of ICT (integrated into all other subjects). Some of our arguments are the following:

- Regarding the current organizational structures at German schools it seems absolutely impossible to assure proper teacher education at university level without a regular subject. Otherwise this would be restricted to pure "in service training".
- According to our experiences the students do not take integrated learning goals seriously enough. In particular they prefer to spend their learning time on regular subjects that lead to grades.

Thus the state of Bavaria decided in 2000 to introduce a mandatory subject called informatics. The following table gives a short overview over the organizational structure of the subject:

<table>
<thead>
<tr>
<th>Direction of study:</th>
<th>(Natural) science &amp; technology</th>
<th>Others</th>
<th>Starting in September</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade 6,7</td>
<td>1 lesson/week</td>
<td>1 lesson/week</td>
<td>2004</td>
</tr>
<tr>
<td>grade 9,10</td>
<td>2 lesson/week</td>
<td></td>
<td>2007</td>
</tr>
<tr>
<td>grade 11,12</td>
<td>3 lesson/week</td>
<td>3 lesson/week</td>
<td>2009</td>
</tr>
</tbody>
</table>

Table 2: Organizational structure
METHODICAL POSTULATES
Following the modern constructivist learning approach of the situated cognition movement (see Brown et al., 1989), particularly cognitive apprenticeship, we have to direct our attention towards the following methodical aspects:

- In order to motivate students properly and to offer them examples of realistic complexity we try to work on authentic problems and environments as early as possible.
- Concerning the structure of the lessons we try to arrange all learning objectives in and around large modelling and programming projects. By this way the students have much time on task left, sparing them from too much learning at home. The information input by the teachers should be as short as possible, usually not exceeding 5-10 minutes within every lesson.
- The projects are defined by use cases (respectively scenarios) that are told as stories, offering narrative anchors by this way.
- Intensive cooperation within task sharing working groups should offer sufficient occasions for social learning.

OOM AND OOP
We have pointed out in many of the publications mentioned above that following our opinion modelling is the most beneficial concept of informatics concerning school education. In fact our whole course of lessons is structured following a sequence of modelling techniques:

- OOM of documents in grade 6 and 7,
- Algorithmic modelling in grade 7 and 10,
- Functional modelling in grade 9,
- Data modelling in grade 9,
- OOM of real world systems in grade 10.

On the other hand we want all students to transfer their developed models as fast as possible into software systems in order to allow them to verify and evaluate their models. Thus we assigned one or more appropriate software systems to every modelling technique:

- OOM of documents: graphic systems, text and hypertext processors, presentation software,
- Algorithmic models: robot systems and programmable graphics systems,
- Functional models: spreadsheets and functional programming languages,
- Data models: relational data base systems,
- OOM of real world systems: Object oriented programming systems.

Despite our emphasis on modelling I will concentrate my considerations in this paper on Object Oriented Programming, because this is one of the most demanding educational objectives in the whole course of lessons. The modelling techniques are explained in detail e.g. in Hubwieser (2003) and Hubwieser (2006).

THE PROGRAMMING TOOLS
Despite the complexity and the didactical drawbacks of Java (e.g., the usage of the equality symbol for the assignment) we propose to use it as programming language, because:

- Many students have already some experience with it,
- It might be regarded as the most frequently used programming language currently (at least concerning new projects),
- It is open source,
- It seems likely that it will be supported technically at least for the next 10 years by nearly all hardware platforms and operating systems.
Following the arguments we have listed in Hubwieser (2006) we propose to our teachers to use the programming environment *BlueJ* (see Barnes and Kölling (2003) and www.bluej.org). Some of the main reasons for this proposal are that BlueJ: 

- Allows the students to work interactively with classes and objects before writing their first program, e.g. inspect the attribute values or invoke methods,
- Works with class diagrams and object cards,
- Saves the students from using (and the teachers from explaining) the cryptic "public static void main" method in their first program,
- Comes with sample projects that directly continue the sight on objects and classes we have presented in the former grades 6 and 7 of the course.

Nevertheless we refuse to prescribe or require the usage of a certain programming language or environment. During the development of the course we just had to assure the existence of at least one programming tool that enables the students to implement all the concepts of the curriculum as well as the didactical concepts we propose.

![Figure 1: The project Shapes in BlueJ](image)

As we have already explained in Hubwieser (2006), we propose to make use of the BlueJ programming system in combination with its ready-made sample project *shapes* that comes with the textbook (Barnes & Kölling, 2003). One of the advantages of this project is that it directly follows the object oriented view on vector graphic documents that the students have acquired in the former grades 6 and 7, even using the same notations (see Figure 1).

**THE DIDACTICAL DILEMMA**

As many experiences show, it is not easy to start the teaching of programming with OOP. Therefore many universities in Germany use another programming paradigm for the beginners, e.g., the functional one (with languages like Haskell, ML or Scheme). Additionally the modern teaching approach mentioned above postulates to pose authentic problems to the students that have relevance in the "real world" outside their school. Thus it seems absolutely not suitable to start programming with the development of a small, very simple program like one of the popular (but terribly dull) "Hello World" programs. In contrary it seems advisable to start with interesting, sufficient complex tasks that convince the students that the concepts they have to learn are really helpful in their later professional lives. Hence we have to start our programming course with quite complex programs that simulate processes that the students know from their everyday life.
On the other hand, if we start with such problems, we might ask too much from the students, because they will have to learn an enormous amount of new, partly very difficult concepts at one time. This might be regarded as the classical dilemma of teaching OOP and following modern teaching approaches.

But where should we take our problems from? As we have pointed out in Hubwieser (2006), we suggest introducing the state semantics of variables and attributes very carefully. Thus the students have to look at an object as a state machine. In order to realize this in a student oriented way, the students need to be able and to understand a simulation program of a typical state machine, e.g., a traffic light system. This issue is offering many problems and tasks that might be posed for the first programs.

Hence we will analyze in the following considerations the simulation of a two-lamp traffic light system by a Java program as an introductory example into OOP.

**INSTRUCTIONAL OBJECTIVES**

Concerning the generality of learning objectives (see “Educational Objectives” above), apparently the third category “instructional objectives” of Anderson and Krathwohl (2001) is just what we need in order to describe learning processes in a fine granularity as it is necessary to evaluate them.

In order to identify the instructional objectives the students have to accomplish if they want to really understand this program, we have a look at the (shortened) Java code of the class definition and associate the concepts the students apparently (regarding the program code) have to know in order to understand the program, which we number and mark with “L” (meaning “local objective”). We start with listing the pure concepts that represent the knowledge part of the objectives, thus belonging to the type conceptual, apparently. Concerning the cognitive process dimension we might be satisfied for the moment (for this first program) by putting all these objectives into the second category understand. Later in the course the students will have to achieve at least the 3rd category apply while using all these concepts in another context while developing own programs. Depending from the teaching method there may occur many more objectives that are not explicitly visible. One of them is the understanding of the singleton software pattern that becomes necessary if the Canvas class in the ready made BlueJ project (which is responsible for the display of the graphical objects) is no longer regarded as a black box concept (as we do).

Additionally to these local concepts the students have to understand some global concepts that are necessary to work with any program, marked with “G” denoting “global”. Nevertheless these global concepts lead to instructional or educational objectives and thus have to be distinguished from the global objectives that were set in the former section of this paper.

G1: Concept of Algorithm: representation, input and output concept, Structural elements;
G2: The total algorithm described by the program might be a combination of algorithmic parts within the methods and the global part formed by the cooperation of the methods;
G3: Usage of the programming environment,
G4: States of a program: coded, compiled, running,
G5: Description techniques for the syntax, e.g. syntax diagrams or BNF,
G6: State Semantics of imperative programs.
By closely looking at the concepts mentioned above, we detect that there are many other instructional objectives that the students have to reach before they are able to understand the concepts that are explicitly necessary. We will mark them with an “I” symbolizing “implicit objectives”. Some examples are:

I1: In order to understand the reference concept (L8) the students have to know that the program code is stored in the working memory of the computer as well as the data the program works with (“von Neumann principle”).

I2: If the students should be able to work with Boolean expressions (L12) (e.g. in conditional expressions L11), they have to have basic knowledge about logic, e.g. about combining expressions by AND or OR operators.
Apparently these objectives are partly ordered by a precedence relation $P$, where $(O_1, O_2) \in R$ or $O_1 \rightarrow O_2$ symbolizes the fact that objective $O_1$ has to be achieved in order to achieve objective $O_2$, in other words before $O_2$. Concerning our implicit objectives we see:

$I_1 \rightarrow L_8,$
$I_2 \rightarrow L_{12} \rightarrow L_{11}.$

If we regard all learning objectives that are necessary to understand our first program above together with all implicit and global objectives as the nodes of a graph and connect the nodes by directed edges, symbolizing the precedence relation, we are able to find out all possible “learning ways” to our program as the set of all paths through this graph from the starting point to the final objectives. This is particularly important with regard to learning objectives of e-Learning systems, because these systems should be constructed in such a way that the learner himself is able to choose one out of these “learning ways”. Therefore we work intensively on the digital representation of such graphs by developing a suitable ontology (see Staller, 2006). This might be helpful also while working with presence (or blended) learning, because even there is such a huge amount of learning objectives that it cannot be managed without electronic support.

**ADVANCED OBJECTIVES**

Additionally to the objectives that have to be achieved in order to understand the first program above, the students are faced with many other “advanced” objectives that occur during the continuation of the course. I restrict myself to list only some of them.

If the students (later) work with an ordinary Java program development environment (that means not with BlueJ), they will have construct a specific “Start-”-class, containing the famous main-method.

```java
public class Start {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        c1.makeVisible();
    }
}
```

This requires additional understanding of the concepts:

L20: (static) class methods and
L21: arrays (concerning the list of arguments).

Later they will have to design classes that are able to carry out mathematical calculations. This requires understanding of the concepts concerning input and output including side effects:

L22: returning the value of a method,
L23: input and output methods.

**GRADUALLY ACHIEVING THESE OBJECTIVES**

This analysis of the learning objectives proves that there is indeed (as already claimed in the section “Didactical Dilemma”) a big number of learning objectives to be achieved simultaneously, if the first object oriented program should fulfil the postulations of modern teaching approaches. We found that this is nearly impossible even for university students. Therefore we arrange as many of these objectives as possible in the course sections that precede the true OOP part. This requires the use of programming systems that allow a propaedeutic approach to programming. Three of the, were designed especially for our course of lessons (unfortunately all these systems are only available in German language currently):
• *Karol the Robot* (Krsko & Freiberger, 2004) which is based on the Karel robot system proposed by Pattis (1981). This system allows the programming of a virtual robot in a very intuitive (German) language without using variables or attributes and thus not requiring the understanding of state semantics (see G5 above).

• *ObjectDraw* (Pabst (2006a)) is a graphical drawing system that allows to produce vector graphics while visualizing the object and class structure of the produced document and approaching OOP by using a specific description language for the classes and objects.

• *EOS* (Pabst (2006b)) is a programming system that allow to draw and animate vector graphics by simple programs that are written in a very intuitive (German) language very similar to the Karol language.

By using these systems we are able to offer a smooth learning way towards true OOP that starts in the first grades of our course and ends up at the real OOP part in grade 10. In detail we arranged the objectives as shown in the table below.

<table>
<thead>
<tr>
<th>Grade</th>
<th>Issue</th>
<th>Local objectives</th>
<th>Global objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade 6</td>
<td>Vector graphics</td>
<td>L2, L3, L7, L9, L10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Text documents</td>
<td>L4</td>
<td></td>
</tr>
<tr>
<td>Grade 7</td>
<td>Hypertext systems</td>
<td>L8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Robots and Algorithms</td>
<td>L6, L11, L14, L18</td>
<td>G1, G4, G5</td>
</tr>
<tr>
<td>Grade 9</td>
<td>Functional modelling and</td>
<td>L12, L13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spreadsheets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left to grade 10</td>
<td>OOM/OOP</td>
<td>L1, L5, L16, L17, L19</td>
<td>G2, G3, G6</td>
</tr>
</tbody>
</table>

Table 4

Detailed descriptions of the teaching concepts that lead to the achievement of these objectives can be found in Hubwieser et al. (1997), Hubwieser and Broy (1999), Müller and Hubwieser (2000), Hubwieser (2002), Hubwieser (2004), Schneider 2005, Hubwieser 2006.

Additional information can be taken from the two classroom textbooks we produced until now: Frey et al. (2004), Hubwieser et al. (2007). The third book covering OOM/OOP is just in production.

**CONCLUSION AND FURTHER WORK**

As the conception of our new subject informatics is approaching its final state by finishing the curriculum for the grades 11 and 12 (which will be published in the following international publications), we will shift the focus of our working group towards the evaluation of the implementation of the concept from now on. Particularly we are interested in the following questions:

• How do learning processes proceed?
• How strong is the Influence of teacher education?
• Which Variables influence the learning success in informatics

After some preliminary work about possible questionnaires and the design of the study we are approaching the pre-test stage. The field study is planned to take place in 2008.
REFERENCES
Pabst M. (2006b) http://pabst.heim.at/objectdraw/
Biography

Peter Hubwieser is an associate professor at the Technische Universität München. He used to teach Mathematics, Physics and Informatics at the Bavarian secondary schools for 15 years before he switched to the university in 2002. He initiated and decisively designed the new mandatory subject of informatics at Bavarian Gymnasiums from 1994 to 2007.

This paper was presented at IMICT 2007: IFIP WG3.1 & WG3.5 Joint Conference on Informatics, Mathematics, and ICT: a ‘golden triangle’. College of Computer and Information Science, Northeastern University, Boston, Massachusetts, USA 27th – 29th June 2007

Copyright Statement B
This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs2.5 License. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/2.5/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.