Empirical Comparison of Objects-First and Objects-Later

Albrecht Ehlert  
OSZ Informations- und Medizintechnik  
Haarlemer Str. 23-27  
12359 Berlin  
Germany  
ehlert@oszimt.de

Carsten Schulte  
Freie Universität Berlin  
Königin-Luise Str. 24-26  
14195 Berlin  
Germany  
schulte@inf.fu-berlin.de

ABSTRACT
In this paper, results of an empirical comparison of objects-first vs. objects-later are presented and discussed. The study was carefully designed to align the two approaches so that the comparison is focused on the main difference between the two approaches: that is the different sequence in which topics are taught: object oriented topics from the beginning or not.

The study with a duration of one year was carried out in a secondary school. In the end, both groups showed the same increase in learning gain, but perceived the difficulty of topics differently.

We discuss study design, results and pedagogical implications.

Categories and Subject Descriptors: K.3.2 [Computers & Education]: Computer and Information Science Education – computer science education, information systems education.

General Terms: Experimentation, Human Factors.

Keywords: Objects-first, objects-later, empirical study, high school, KCS1, CS0, computing education research

1. INTRODUCTION
The debate on objects-first is still opinionated and acute ([13], [18]). A main reason for this is the complexity of the debate, as there are many pedagogical dimensions involved: learning goals, teaching examples, learning tasks, types of exams, tools, relevance of certain topics for introductory courses, evaluation techniques and perspectives. Therefore, empirical studies run the risk of comparing apples with oranges.

In the next section we discuss the current state of empirical research concerning objects-first vs. objects-later. Subsequently, our study design will be presented, followed by results, interpretation and a conclusion.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICER ’09, August 10–11, 2009, Berkeley, California, USA.  
Copyright 2009 ACM 978-1-60558-615-1/09/08...$10.00.

2. EMPIRICAL RESULTS ON THE OBJECTS-FIRST DEBATE
In 2005, Bruce published a report on the discussion on objects-first in the SIGCSE mailing list [4]. Due to – or despite of – the “increasing trend […] to introduce classes and objects very early” he sees an “increasing concern among some faculty about whether this is the right direction” ([4], p. 111). His conclusion is that more empirical evidence is needed to examine the success of innovative teaching approaches like objects-first.

Such a conclusion is easier to reach than to fulfill. This is due to the inherent complexity of the discussed topics. Although objects-first sounds like a concrete and specific pedagogical idea, there are a lot of different pedagogical dimensions involved. A list of paper titles contributing to the debate on objects-first gives an impression of the issues involved:

- Mental models and programming aptitude [5]
- From objects-first to design-first with multimedia and intelligent tutoring [21]
- Day one of the objects-first first course: what to do [3]
- Implications of perspective in teaching objects first and object design [7]
- An objective comparison of languages for teaching introductory programming [16]
- Methodology first and language second: a way to teach object-oriented programming [27]
- Relationship of early programming language to novice generated design [6]
- A study of the development of students’ visualizations of program state during an elementary object-oriented programming course [22]
- Adopting XP practices for teaching object-oriented programming [11]

In the study of Vilner et. al. [25], no significant differences were found between procedural-first vs. objects-first. Reges reports that procedural-first seems more promising [19]. In contrast, Decker concludes that “objects should be introduced early and emphasized after their introduction” ([8], p.245). In a summarizing report on research on introductory programming, Pears et.al
[17] conclude that “studying individual learning settings is no longer sufficient. Our literature review leads us to believe that larger-scale systematic studies are vital to a better long-term understanding of how to teach this key element of the field. […] They should provide strategically significant new empirical results, allowing the community to use research to begin resolve key questions regarding what to teach beginning programmers and how to teach programming” ([18], p. 211).

From the current state of research on objects-first vs. objects-later we can conclude that there are examples/variants of each approach that work, but there are hardly any generalizations available.

In our opinion, the major problem is to develop a setting which allows a fair comparison of the two opposing teaching approaches. For example in [25], two different languages are used; the two different learning groups are comparable, but only anecdotic. In [19], the different groups cannot be compared, as they are from different years. In addition, several other aspects also differ (teacher, material, exams). In [8], both groups mastered the ‘basic concepts’ equally well ([8], p. 243), but the objects-emphasized group was better in the exam that focused on object-oriented concepts ([8], p. 243f). It is difficult to control the important variables.

The problem is twofold: First, a point of view for the comparison has to be defined; i.e. an operationalization of the difference between objects-first and objects-later. This may be the focus of teaching content, a methodology, the examples used, etc. Second, all other important issues should be controlled, i.e. they should be made comparable so that their influence on the result is equal on both groups, or can be measured in order to estimate its influence on the overall result.

In the following section, we argue for our solution. The sequence of topics taught, was considered as the central difference between objects-first and objects-later.

3. OUR STUDY

From the discussion in the prior section two conclusions become immediately apparent. In order to produce reliable results, a fair comparison of two different teaching approaches is necessary. This involves the need to control the relevant variables. Otherwise the results are very hard to interpret. Second, given the many facets of the debate, such a comparison cannot be directed to compare all issues involved in the debate on objects-first and objects-later.

In the following subsection we firstly will define and operationalize objects-first and objects-later as OOP-first and OOP-later, which differ only in the sequence of topics in the course schedule. Then, we will develop the study design and define the variables we aim to control.

3.1 OOP-first vs. OOP-later

In a (non-representative) study [1], three general approaches to objects-first were distinguished:

1. Using objects: At the beginning of the course, the student uses objects implemented beforehand. When the student has understood the concept object, he moves on to defining classes by himself. Focus is on usage before implementa-

2. Creating classes: The student both defines and implements classes and creates instances of the defined classes. Focus is on the concrete-creative part of programming.

3. Concepts: This involves the teaching of the general principles and ideas of the object-oriented paradigm, focusing not just on programming but on creating object-oriented models in general. Focus is on the conceptual aspects of object-orientation.

This study follows the second mentioned approach to objects-first, in which the focus is on the concrete part of programming (for more details on the three different concepts of object-first courses see [1]).

Using the term OOP-first is supposed to illustrate this focus. The counterpart to OOP-first is – quite naturally – OOP-later. This term implies two important aspects of our study design: Firstly, it implies the same coverage and depth of object-oriented content; and secondly it shows the main difference of the two approaches: the sequence in which the topics are taught. In addition, we suppose that the “OOP-later”-approach closely resembles what people refer to when objects-first is defined in opposition to “traditional teaching”:

OOP-later is the traditional programming course with a common sequence of topics, and many (often also common) small programming tasks to solve for the students. Such a typical sequence starts with variables, assignments, is then enriched by selection, iteration and sub-programs (procedures). There might be small differences within this approach, but in general this is the typical sequence to be found in textbooks.

Surprisingly, this sequence is quite similar in textbooks on object-oriented programming. Hence, we can clearly define a list of topics to be included in both approaches – which differ only in the sequence regarding the object-oriented topics.

In summary, the list of topics is [10]:
- Overview on OOP, classes and objects
- Variables/attributes, constants, primitive data types
- Control structures: iteration and selection
- Sub-programs: procedures/methods
- Non-primitive data types
- Inheritance
- Association

Given this operationalization, it is obvious that the topics in OOP-first are comparable to the topics in OOP-later; see the following table:

<table>
<thead>
<tr>
<th>OOP-first</th>
<th>OOP-later</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Classes and objects</td>
<td>1. Data types (incl. Variables, constants)</td>
</tr>
<tr>
<td>2. Attributes (incl. data types)</td>
<td>2. Control structures</td>
</tr>
<tr>
<td>3. Methods (including control structures)</td>
<td>3. Procedures</td>
</tr>
<tr>
<td>4. Inheritance</td>
<td>4. Classes and objects</td>
</tr>
<tr>
<td>5. Association</td>
<td>5. Inheritance and associaton</td>
</tr>
</tbody>
</table>

Figure 1: Typical teaching sequences for OOP-first and OOP-later

In section 4, we give a more precise overview of the two different sequences used in the empirical study.
3.2 Study Design

We decided to focus on two main research questions:

1. Are the two approaches really different in terms of learning outcome?
2. Are the two approaches really different in terms of perceived difficulty (comfort level)?

In prior work, sometimes different outcomes were measured, sometimes not. In [24] and [25], it is reported that in-between differences in the learning outcome subsequently vanish. Therefore, it seems reasonable to restrict the comparison to the end of a longer period, for example to the end of a whole learning cycle in which all topics were taught. In our case, this meant a test at the end of one school-year plus a second test after the summer holidays to check for long-lasting effects. The test of learning-gain comprises all topics taught.

The second possible difference refers to general approaches to programming. In prior studies, several different themes connected to students’ attitudes are mentioned: self-efficacy, abstraction-ability, typical misconceptions, strategies to program, etc. We decided to focus on the students’ perception of difficulty and hence the comfort level of the course, which is considered as an important factor for course success [26]. As the issue of perceived difficulty is process-centered, it was monitored throughout the teaching sequence (retrospective at the end of each teaching module, see Figure 2).

In the test at the end of the course, two topics were included: dynamic OOP and object-oriented modeling (see appendix). One might argue that introduction to programming only, and even a focus on language constructs is not sufficient. Therefore, dynamic OOP asks for understanding the dynamics of an object-oriented paradigm. Modeling asks for the process of analyzing a problem and designing an object-oriented solution for a given problem. In the operationalization of the two different approaches for the study it might be that this kind of knowledge is missing.

Overall, post- and follow-up-tests comprise eleven sections (nine sections for each teaching topic and two additional topics on OOM and dynamic OOP). Each section contains a question as indicator of the learning gain specific to the module. Each of these nine was classified as either object-oriented or procedural – so the topics were used to define the different sequences of the OOP-first and OOP-later versions of the course, and to define the topics to test at the end of the course.

These topics were tested in pre-, post- and follow-up-tests. Students were given 100 minutes for answering questions to the 9 topics, and another 30 minutes for answering questions to the two additional questions concerning modeling and dynamic OOP. The pre-test was a shortened version of the whole test.

In the tests, OOP-related topics as well as non-OOP-related topics were represented equally (that is: approximately 50% each). Some of the questions were given as open-ended questions, some as multiple-choice questions (not more than 30% MCQ).

The pre-test was a small sub-sample of these questions. However, as the question of OOP-first versus OOP-later is focused on introductory programming, we did not expect any prior knowledge. Instead, giving students a big test at the beginning in which they are likely to fail completely could lead to negative attitudes. Therefore, we decided to not give the pre-test to the students involved in the study; instead, we provided this test to another group of students not involved in the study. In addition, we thought it is reliable enough to assume equality of the students regarding prior knowledge.

The following figure gives an overview of the instrument used throughout the study:

![Study design (main study) (arrows are indicating the nine short questionnaires students answered during the course, e.g. the comfort level)](image)

Our treatment variable was the difference in the sequence of topics. Other interfering variables should be controlled so that differences in learning outcome and/or students’ attitudes are very likely to be caused by the difference of OOP-first to OOP-later.

The experiment was noticeably influenced by two factors: the students and the implementation of the teaching concept.

Regarding the students, prior to the experiment all students had been divided into subgroups, so that both groups were as similar as possible with regard to major influencing variables. As such variables we chose gender, age, prior exam grades in German language and mathematics. However, the school the study was done at (OSZ IMT), starts in grade 11 – so prior grades were assigned to the students (who previously attended a variety of different schools) by different teachers. Especially the math grade is considered as a good indicator of course success (see e.g. [26]).

The subgroups were assigned as classes, so that all subjects were taught in these different classes. It was thus possible to add an additional check: both classes were assigned to the same math teacher. Given the same prior grades in math, both classes were expected to have similar math grades at the end of the school year.

Regarding the concrete teaching environment, several measures were taken. In both groups, teaching time, exams, programming language and tools were the same. The individual assignments, homework tasks and examples used had to differ, though: In order to measure effects of different ‘teaching sequences’, we wanted to be able to make use of prior knowledge gained in each group. For example, teachers could and should point out similarities of new concepts to prior learned concepts or possibilities to use a new concept in context of a prior learned concept (like using iteration (a new concept) in order to step through an array, which was learned beforehand).

However, the teaching style itself had to be the same (the role of examples, the complexity of assignments, etc.). Therefore, the teachers involved had regular meetings to coordinate their teaching style and exchange their materials.
Of course it is possible that the different characters of the teachers involved have affected the study – in order to control this as much as possible, regular meetings were held, and questions regarding the teacher were included in the questionnaires the students answered after each module.

This empirical design was tested in a pre-study, which like the main-study also spans an entire school year.

The pre-study revealed some minor issues, e.g. in the OOP-first version the topics were shorter, so that the coverage and depth of topics for each topic was refined for the main study. In addition, in the OOP-later version topics changed positions [9]. Lastly, the empirical instruments were refined for the main study (see section 0).

4. THE STUDY
The study was conducted at Oberstufenzentrum Informations- und Medizintechnik (OSZ IMT) in Berlin within the years 2006/07 (pre-study) and 2007/08 (main study). OSZ IMT is Berlin’s largest secondary school (grades 11 to 13) specialized in Computer Science. On average, the students attending the school have no prior experiences in programming and are likely to be medium or low achievers (this estimation is based on specifics of the German school system, where high achievers are sorted out early and attend different types of schools (Gymnasium)). The aim of the introductory sequence is to introduce into the object-oriented paradigm and into programming basics of Java. A school year has 40 weeks; three "hours" a week are dedicated to computing (45 minutes each hour).

4.1 Implementation of the Study
The pre-study was conducted in school year 2006/07, the main study in 2007/08. During the main study, the students were on average 17 years old. All students were new at the school (the schools starts with class 11).

The figures below give an overview about the sequence of topics for both classes; a more detailed description of the topics can be found in the appendix.

<table>
<thead>
<tr>
<th>topic</th>
<th>1</th>
<th>T1: Introduction in the OOP with BlueJ (Introduction into concepts, using objects)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T2: Introduction in IDE, variable, constant, data types, sequences (here: in context of attributes)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T3: Class and object (Implementation of classes and instantiation of objects)</td>
<td></td>
</tr>
<tr>
<td>4/5*</td>
<td>T4: Methods (in context of classes)</td>
<td></td>
</tr>
<tr>
<td>5/4*</td>
<td>T5: Selections including Nassi-Shneidermann-diagrams</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T6: Loops including Nassi-Shneidermann-diagrams</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T7: Arrays and Strings</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T8: Inheritance, abstract methods</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>T9: Association, static members</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Sequence of topics taught so far in the OOP-first group
(*left digit: Sequence pre-study, right digit: sequence main study)

As mentioned above, both approaches should make use of possible advantages due to the chosen sequence of topics. Therefore there are some differences between the topics, for example in topic T3 (class and object): In the OOP-later version of this topic, data types were in the subtopic attributes included. This could not be done in the OOP-first group, because here data types were taught afterwards. On the other hand, in topic T2 (data types), the OOP-first version included use of data types for attributes.

<table>
<thead>
<tr>
<th>topic</th>
<th>1</th>
<th>T2: Introduction in IDE, variable, constant, data types, sequences (here: without referring to attributes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T5: Selections including Nassi-Shneidermann-diagrams</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T6: Loops including Nassi-Shneidermann-diagrams</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T7: Arrays and Strings (here: Strings were introduced as special data type, not as class)</td>
<td></td>
</tr>
<tr>
<td>5/7*</td>
<td>T3: Class and object</td>
<td></td>
</tr>
<tr>
<td>6/5*</td>
<td>T1: Introduction in the OOP with BlueJ</td>
<td></td>
</tr>
<tr>
<td>7/6*</td>
<td>T4: Methods</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T8: Inheritance, abstract methods</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>T9: Association, static members</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Sequence of topics taught so far in the OOP-later group
(*left digit: Sequence pre-study, right digit: sequence main study)

As in the pre-study some problems occurred in synchronizing both courses, several measures were undertaken to change this for the main study: Throughout the year, both teachers had regular and more frequent meetings in order to ensure a close match of teaching style; e.g. by using teaching methods like group work in the same ways and for the same topics, by giving learning tasks and worksheets the same structure.

At the end of the year (topics 8 and 9), learning tasks and materials used were identical. Both teachers also cooperated to give each module the same length (the maximum difference was 3 hours). In addition, for each topic the learning goals were defined in advance and served to guide the teaching process.

5. EMPIRICAL RESULTS AND INTERPRETATION
This section presents selected results of the study, including results of the pre-study, and offers some interpretations.

5.1 Learning Outcome
The overall learning outcome certainly is of major interest. Do the two approaches lead to different learning gains?

We start by discussing results from the pre-study, summarized in the following table.

Note, the following table presents results of the pre-study, in which the post-tests covered only six topics (instead of eleven in
the main-study). The learning outcome was the same for four of these six topics. Differences were found in two of them. These two topics were object-oriented, and the OOP-later group showed a significantly better performance (more information on the pre-study in [9]):

<table>
<thead>
<tr>
<th>topic</th>
<th>points</th>
<th>OOP- First</th>
<th>OOP- Later</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control- and Data- Structures</td>
<td>20</td>
<td>61%</td>
<td>59%</td>
</tr>
<tr>
<td>Methods</td>
<td>20</td>
<td>67%</td>
<td>64%</td>
</tr>
<tr>
<td>Nassi-Shneidermann-Diagrams</td>
<td>20</td>
<td>66%</td>
<td>64%</td>
</tr>
<tr>
<td>Object-oriented Programming (static)</td>
<td>20</td>
<td>72%*</td>
<td>79%</td>
</tr>
<tr>
<td>Unified Modeling Language (UML)</td>
<td>20</td>
<td>72%*</td>
<td>85%*</td>
</tr>
<tr>
<td>Addition: OOP (dynamic)</td>
<td>10</td>
<td>50%*</td>
<td>67%*</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>110</strong></td>
<td><strong>65%</strong></td>
<td><strong>70%</strong></td>
</tr>
</tbody>
</table>

Figure 5: Overview of the assignment at the end of the pre-study
(*significant results: bold print)

However, during the year of the pre-study, the OOP-first class progressed much faster so there was time for additional topics at the end of the school year. This was not the case for the OOP-later group which only learned the OOP-topics at the end. Consequently, the results may be ascribed to the fact that OOP-first was not taught at the same level of detail (but covered more topics), and the OOP-later students were better in those topics they learned last.

The overall interpretation of this result is difficult. One conclusion was to check in the main study more precisely that both classes learn the same topics at the same length. In addition, these differences might be vanishing some time later on. One interpretation through is the feeling that OOP-first approaches might ‘succeed’ the teacher to proceed (too) quickly.

In the main study, the post-test (as well as the follow-up test) consisted of one section for each topic, plus two additional topics concerning OOM and dynamic OOP.

The following table shows the results of the post-test in the main study.

Please note that T1 and T3 were combined into one section of the test because it is hard to test BlueJ and the initial concepts (class, object) only (for more details see appendix). Therefore, the concepts were tested in context with implementation of classes (incl. setter, getter, constructor ...).

Similar to the pre-study both classes generally had comparable results, in two topics one class was significantly better; but here it was the OOP-first group instead of the OOP-later group. In addition, the results in the T4 (methods) are noteworthy, although not significantly different.

Overall, the OOP-first class shows better learning results not only in the post-test but also in the math grade (see Figure 6). The meaning of this result is not clear, though. It turned out that the OOP-first group had a better (more intense) preparation for the post-test than the other group.

<table>
<thead>
<tr>
<th>topic</th>
<th>points</th>
<th>OOP- First</th>
<th>OOP- Later</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2: Data structures and control structure sequence</td>
<td>10</td>
<td>6,8</td>
<td>7,0</td>
</tr>
<tr>
<td>T5: Control structure selection</td>
<td>10</td>
<td>6,3</td>
<td>7,1</td>
</tr>
<tr>
<td>T6: Control structure loops</td>
<td>10</td>
<td>5,8</td>
<td>6,0</td>
</tr>
<tr>
<td>T7: Arrays and Strings</td>
<td>10</td>
<td>5,7*</td>
<td>3,7*</td>
</tr>
<tr>
<td>T4: Methods</td>
<td>10</td>
<td>6,1</td>
<td>4,8</td>
</tr>
<tr>
<td>T1/T3: Introduction in the OOP / Class and object</td>
<td>20</td>
<td>14,6</td>
<td>14,9</td>
</tr>
<tr>
<td>T8: Inheritance</td>
<td>10</td>
<td>7,6</td>
<td>6,9</td>
</tr>
<tr>
<td>T9: Association</td>
<td>10</td>
<td>6,2*</td>
<td>4,9*</td>
</tr>
<tr>
<td>Addition: OOP (dynamic)</td>
<td>10</td>
<td>7,1</td>
<td>7,0</td>
</tr>
<tr>
<td>Addition: OOM</td>
<td>10</td>
<td>6,7</td>
<td>5,9</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>110</strong></td>
<td><strong>72,9</strong></td>
<td><strong>68,2</strong></td>
</tr>
</tbody>
</table>

Figure 6: Overview of the assignment at the end of the study (main study, post-test)
(*significant results: bold print)

It seems rather interesting, that in the follow-up test all significant differences vanished. The follow-up test was conducted approximately eight weeks after the post-test, which means after the six-week summer break.

<table>
<thead>
<tr>
<th>topic</th>
<th>points</th>
<th>OOP- First</th>
<th>OOP- Later</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2: Data structures and control structure sequence</td>
<td>10</td>
<td>6,4</td>
<td>7,2</td>
</tr>
<tr>
<td>T5: Control structure selection</td>
<td>10</td>
<td>6,5</td>
<td>7,4</td>
</tr>
<tr>
<td>T6: Control structure loops</td>
<td>10</td>
<td>5,7</td>
<td>5,4</td>
</tr>
<tr>
<td>T7: Arrays and Strings</td>
<td>10</td>
<td>3,6</td>
<td>3,8</td>
</tr>
<tr>
<td>T4: Methods</td>
<td>10</td>
<td>5,3</td>
<td>4,6</td>
</tr>
<tr>
<td>T1/T3: Introduction in the OOP / Class and object</td>
<td>20</td>
<td>13,4</td>
<td>13,0</td>
</tr>
<tr>
<td>T8: Inheritance</td>
<td>10</td>
<td>5,8</td>
<td>6,0</td>
</tr>
<tr>
<td>T9: Association</td>
<td>10</td>
<td>2,5</td>
<td>1,6</td>
</tr>
<tr>
<td>Addition: OOP (dynamic)</td>
<td>10</td>
<td>5,3</td>
<td>5,1</td>
</tr>
<tr>
<td>Addition: OOM</td>
<td>10</td>
<td>5,3</td>
<td>6,3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>110</strong></td>
<td><strong>59,8</strong></td>
<td><strong>60,4</strong></td>
</tr>
</tbody>
</table>

Figure 7: Overview of the assignment at the end of the study (main study, follow-up-test)
(*significant results: bold print)

Concerning the overall result as well as the individual results in individual topics we could not confirm a consistent difference between the two approaches.

Interpreting these results we came to the conclusion that in terms of learning outcome the two approaches are not(!) different.

This result is comparable to [24] and [25], where no differences were found either or the differences vanished in later teaching and learning.

The following figure compares the overall percentages of pre-, post- and follow-up-test. In the pre-test not all topics were
given, therefore the learning success of 100% applies to topics T1 to T8 (association, dynamic OOP and OOM is missing).

![Learning Success (in %)](image)

**Figure 8: Learning success (main study, topics 1–8)**

Students start without prior knowledge, the 9.8% in the pre-test is close to the probability of guessing the correct answer (Note, multiple choice questions were used). Gained Knowledge is surprisingly quite constant between pre-test and follow-up-test (results differ by only 5.8%).

### 5.2 What Is Difficult, What Is Easy?

For a detailed analysis of the results we ranked the topics according to their overall average grade.

<table>
<thead>
<tr>
<th>topic</th>
<th>max</th>
<th>points</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5: Control structure selection</td>
<td>10</td>
<td>7,0</td>
</tr>
<tr>
<td>T2: Data structures and control structure sequence</td>
<td>10</td>
<td>6,8</td>
</tr>
<tr>
<td>T1/T3: Introduction in the OOP / Class and object</td>
<td>20</td>
<td>13,2 (6,6)</td>
</tr>
<tr>
<td>T8: Inheritance</td>
<td>10</td>
<td>5,9</td>
</tr>
<tr>
<td>Addition: OOM</td>
<td>10</td>
<td>5,8</td>
</tr>
<tr>
<td>T6: Control structure loops</td>
<td>10</td>
<td>5,6</td>
</tr>
<tr>
<td>Addition: OOP (dynamic)</td>
<td>10</td>
<td>5,2</td>
</tr>
<tr>
<td>T4: Methods</td>
<td>10</td>
<td>5,0</td>
</tr>
<tr>
<td>T7: Arrays and Strings</td>
<td>10</td>
<td>3,7</td>
</tr>
<tr>
<td>T9: Association</td>
<td>10</td>
<td>2,1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>110</td>
<td>60,3</td>
</tr>
</tbody>
</table>

**Figure 9: Results ordered by score (main study, follow-up-test, both groups together)**

The table shows a ranked list of the mean grade for both groups. Easier topics are listed at the top, more difficult topics at the bottom. In addition we indicated three levels of difficulty. In each of the three levels of difficulty we find OOP-topics as well as non-OOP-topics.

Interpretation:

Neither OOP nor non-OOP-topics are more difficult than topics from the other approach. Therefore it is not possible to justify OOP-first or OOP-later with the difficulty of topics of the paradigm (whether arguing for teaching difficult things first or later). Please note that the difficulty is measured in both groups; it is independent of the used teaching paradigm.

In the next step we check for effects of the teaching sequence. The following table maps the difficulty of topics to the teaching sequence:

<table>
<thead>
<tr>
<th>Sequence of topics, colors indicating difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
</tr>
<tr>
<td>OOP-first</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>OOP-later</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Figure 10: Comparison of teaching sequence with measured difficulty of the topic. Easiest topics marked in white, middle in light grey and the most difficult topics in dark grey**

The OOP-first approach is close to an approach from easy to difficult topics. Only topics T7 and T8 disturb this pattern. Arrays and Strings (T7) were taught before Inheritance (T8).

The OOP-later approach also starts with easy topics, but then teaches control structure loops (T6) and arrays (T7) before introducing two easy topics with OOP overview (T1) and objects and classes (T3).

Interpretation:

Both sequences progress in a reasonable way from overall more easy to more difficult topics.

For the OOP-later approach it is the defining characteristic to introduce object-oriented concepts later in the course. It may be motivating for students to start learning the new paradigm with some easier topics.

Topic T7, however, the introduction of strings and arrays, is a difficult topic touched upon quite early (i.e. before the OO topics); but this is due to the close connection of iteration and arrays, allowing the integration of concepts and interesting learning tasks.

The OOP-first approach moves straightforward from easy to difficult topics. Here it might be an option to change T7 (arrays) with T8 (inheritance), so that the most difficult topics (T7 and T9) are taught at the end of the course; but here as well arrays were taught after loops for the same reasons as discussed for the OOP-later approach.

Let us now turn to the question why T7 and T9 are the most difficult topics.

In section 5.5, the question of difficulty is discussed from the students' perspective.

### 5.3 Why Are Association and Arrays the Most Difficult Concepts?

This section presents typical questions used in the test to determine students’ learning outcome regarding arrays and association.

We discuss some possibilities in order to clarify why these topics are the most difficult.

In the following table a typical multiple-choice-question is given for the topic arrays:
int[] x = new int[25];
Which of the following statements is/are correct?

O x.length has the value 24
O x[0] has the value null
O x[25] has the value 0
O x[24] is not defined
O None of them
1 Point

Figure 11: Question concerning arrays

In order to answer this question correctly, the student has to know about the array construct in Java, i.e. about the numbering of elements (starting with zero), as well as about the execution of the statements, that is how new fields are initialized (also starting with zero).

For another question concerning arrays, the student has to know how to handle arrays with loops:

Please initialize the following array with the help of a loop to the values 15, 35 and 55
short[] numberArray = new short[3];

Figure 12: Question concerning construction of arrays

The association exam comprises two tasks: drawing the UML class diagram (showing the associations between the classes) and writing the Java code to implement the associations.

Take a look at the following example:

Given is the following scenario: A company consists of several departments. Each department has many employees and exactly one employee as a leader. Please complete the following source code (4 lines), so that the relationship between the department and the employees is implemented. Hint: the private members get the references through the constructor. The class employee exists.

public class Department { 4 Points
  /*1*/
  /*2*/
  public Department(Employee departmentLeader, Employee[] employeeList) {
    /*3*/
    /*4*/
  } // Constructor
} // Department

Figure 13: Question concerning the implementation of an association

The students have to construct a reference to one department leader and a reference to a list of employees.

This task is very difficult for most students in the study.

public class Department {
  private Employee departmentLeader;
  private Employee [] employeeList;
  public Department(Employee departmentLeader, Employee[] employeeList) {
    this.departmentLeader = departmentLeader;
    this.employeeList = employeeList;
  } // Constructor
} // Department

Figure 14: Possible Answer to the question in Figure 13

Interpretation:
The topics "association" and "arrays" were the most difficult ones. Arrays may be seen as more complex because the data type gets more complex from integer to an array of integer; e.g. iteration is needed to access the member variables of a field. 1:n-association is complex for similar reasons, e.g. it usually is implemented using an array.

The most difficult topics taught can be regarded as both procedural (arrays) and object-oriented (association of classes). In other words, there is neither an argument for OOP-later nor for OOP-first claiming a certain sequence is easier because it prevents the teacher from having to teach difficult topics too early in the course – or the other way around, claiming the only way to teach difficult topics is to teach them from the beginning so that there are more chances they are properly learned.

Instead, the difficulties we have measured in the post-test are outside the area of OOP-first and OOP-later. First, they appear regardless to any sequence, and they are beyond this distinction, as we try to argue: Arrays were taught in combination with loops, because one has to iterate through an array in order to make use of the concept.

Associations are also implemented by arrays, in other words, one has to iterate through an 1:n-association in order to make use of the concept.

The difficulty is to integrate concepts, not only language constructs but also data structure and algorithms. Hence the need arises

- to comprehend 'larger entities instead of details' [17];
- to understand the whole instead of parts, which is more difficult according to the SOLO-taxonomy [14], [13];
- to relate different parts of the program, which is more difficult than understanding entities only [23].

In summary, the most difficult topics are not affected by changing from OOP-later to OOP-first or vice versa.

5.4 The Role of OOM in OOP Courses

In some discussions, the ability to translate a problem into an object-oriented design, the ability to create an object-oriented model, is seen as natural or even essential part of introducing the object-oriented paradigm (compare e.g.: [21], [7], [27], [1]).

Although OOM was no central or official learning goal in one of the modules, throughout the courses, students were introduced to aspects of OOM: For example, in order to introduce inheritance, examples were given, discussed, designed and then implemented using the new concept. In the topic association, relationships between objects were discussed and abstracted to design associations between classes.

Both classes gained experiences in OOM via learning tasks or questions like these:

- What attributes and methods should be assigned to the class?
- What is the relationship between these objects?
- Is there a relationship between classes which can be expressed by inheritance?
- Are there any associations between classes?
For these type of tasks usually simple UML-like diagrams (class diagrams, object diagrams) were used and produced. The term OOM in this paper refers these afore mentioned aspects.

Given is the following scenario: Humans can own valuable items. For example, Mr. Mueller owns a special car with a maximum speed and a certain value. Mrs. Meyer owns a certain house with a defined size of living space and a certain value. Please model this scenario with the help of a UML-OOA-class-diagram.

**Figure 15: OOM-Question**

<table>
<thead>
<tr>
<th>Human</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>gender</td>
<td>value</td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Car</th>
<th>House</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxSpeed</td>
<td>size</td>
</tr>
</tbody>
</table>

**Figure 16: Possible answer for Figure 15**

On average, students reached 5.8 of 10 possible points (see Figure 7 and Figure 9). Students thus gained some knowledge in OOM.

This result is interesting for two reasons:

- It suggests that OOM can be learned in an OOP-centered course as well. Or, more cautiously, it seems that learning OOM is not hindered by learning OOP.
- It sheds light on the debate on the so-called paradigm shift between object-oriented design and procedural design (expressed in more familiar terms: the discussion on object-centered thinking or object-oriented thinking). Cautiously phrased: The example suggests that such OOM-competencies can be learned by students attending an OOP-later course as well.

### 5.5 Perceived Difficulty, Comfort Level

In this section we discuss the subjective views of the students on difficulty as perceived during the courses.

After each module, the students were asked to judge the difficulty of the last topic.

The following table summarizes the results.

Overall, the mean of the perceived difficulty was 2.32 in the OOP-first course, and 2.06 in the OOP-later course. This difference is statistically significant, but it has no effects on the learning outcome (see section 6).

<table>
<thead>
<tr>
<th>Topic</th>
<th>OOP-First</th>
<th>OOP-Later</th>
<th>Both groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>T8: Inheritance</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T3: Class and object</td>
<td>2,1</td>
<td>2</td>
<td>2,1</td>
</tr>
<tr>
<td>T1: Introduction in OOP with BlueJ</td>
<td>2,1</td>
<td>2</td>
<td>2,1</td>
</tr>
<tr>
<td>T7: Arrays and Strings</td>
<td>2,4</td>
<td>1,9</td>
<td>2,2</td>
</tr>
<tr>
<td>T4: Methods</td>
<td>2,2</td>
<td>2,1</td>
<td>2,2</td>
</tr>
<tr>
<td>T9: Association</td>
<td>2,4</td>
<td>2,2</td>
<td>2,3</td>
</tr>
<tr>
<td>T2: Data structures and control structure sequence</td>
<td>2,7*</td>
<td>1,9*</td>
<td>2,3</td>
</tr>
<tr>
<td>T5: Control structure selection</td>
<td>2,5</td>
<td>2,1</td>
<td>2,3</td>
</tr>
<tr>
<td>T6: Control structure loops</td>
<td>2,5</td>
<td>2,2</td>
<td>2,4</td>
</tr>
</tbody>
</table>

| Average                | 2,32*     | 2,06*     | 2,19        |

**Figure 17: Perceived difficulty level of topics (main study 2007/2008) ordered by mean values of both groups (**Topic 2 and average: differences are extremely significant**)

The biggest difference between the two courses is in topic 2, which was perceived as easier by the OOP-later students. This difference is presumably due to the sequence of tools used.

Here we have found a sequence effect – but maybe one which is due to perceived comfort. In the OOP-first class, students started with BlueJ in the first module, but then continued with a text-driven editor in topic 2.

Please note, the second tool, a specific Java-editor called JavaEditor (see appendix), is especially designed for use in secondary school; but it is a text-editor without BlueJ’s ability to visualize objects (object bench and object inspector). A simple conclusion is to avoid switching from comfortable to less comfortable tools too early. The second tool offers many more comfort functions for editing source code, but presumably this is needed later only.

Overall, this difference can be considered an artifact of the study design where each topic was designed to be as similar as possible for both classes. A solution might be to let the students program in BlueJ. A later switch to JavaEditor with syntax highlighting and code completion is then probably no obstacle any more.

The general conclusion of this is that pedagogical enhancements of the two course concepts are likely to be of such issues, and are not a question of switching the ‘teaching paradigm’ – that is from first to later or vice versa.

A second difference in perceived difficulty can be found in topic 7; although the difference is not significant. In both classes arrays were taught after loops (topic 6), but apparently topic 7 was perceived as more difficult in the OOP-first class.

Arrays are one of the two most difficult topics in the entire course, students of the OOP-later underestimated the difficulty. We do not have any interpretation why.

Figure 17 also presents a ranking of perceived difficulty. The first three topics, the easiest ones, are all object-oriented topics –
and the three topics perceived as most difficult are all control structures, i.e. more procedural. Note that the list is ordered by the means of both groups, but the object-later group has a smaller mean variation, therefore the order is due to the order of the objects-first group. In summary, the object first group regards object-oriented topics as easier (and for the objects-later group the variances are too small to conclude anything).

[12] and [17] also analyzed perceived difficulty of topics, albeit on university level. [12] asked students who had one or two programming courses in Java or C++ at university to describe the perceived difficulty of programming constructs. For the topics that were taught in [12] as well as in our study, these are (from easy to difficult): selection, loops, variables, parameters, arrays. Arrays was also one of the most difficult topics in the post-test, but was judged a little less difficult.

[17] also included object-oriented concepts in their study. The order of perceived difficulty for the topics found in both studies was from easy to difficult ([17], p. 58): variable, basic function calling, operators and precedence, conditional operations, loops, scope of variables, classes and objects, arrays, encapsulation, string handling, inheritance, function over-riding (in inheritance), dynamic allocation of memory (with new). In this study, class and objects are perceived more difficult.

Generally, it is problematic to compare perceived difficulty across several studies. In the studies mentioned above, the students' perceived difficulties were compared with the opinions of teachers.

Our study enables us to compare the perceived difficulty with the results of the post-test. Students' perceptions of difficulty sometimes differ from teachers' views, as well as from the measured difficulty in a test. In our study for example occurred a strange mismatch in topic 2, which was measured as very difficult in the post-test but perceived (of the OOP-later group) as very easy.

Possibly perceived difficulty is more focused on the learning process, and the tests of learning gain are more focused on the result of this process – and maybe also teachers' opinions are more focused on the results than on the process.

Still, perceptions of difficulty, i.e. students' assessment of the learning process, are interesting to capture. The main result here is the difference of the two groups in general: this difference is still significant without topic T2. Not in any detail, but in general students perceived the OOP-first version of the course as more difficult. We discuss this difference in terms of comfort level.

The perceived difficulty is connected with the teaching and learning climate. It might improve the learning climate and motivation when the topics taught are perceived as being easy. Wilson and Shrock [26] found that from 12 success factors, "Comfort Level in the computer science class was the best predictor for success in the course" ([26], p.187). They measured the construct comfort level by asking for the perceived difficulty.

Hence, when there are differences between the overall comfort level of OOP-first and OOP-later, these should be due to the sequence of topics. Figure 18 compares the level of difficulty of the different topics.

![Figure 18: Subjective topic difficulty level for the pupils (main study 2007/2008) ordered by topics](image)

*Figure 18: Subjective topic difficulty level for the pupils (main study 2007/2008) ordered by topics (Topic 2: difference extremely significant)*

Overall, both sequences show different patterns in the flow of perceived difficulty, see Figure 19 and Figure 20.

![Figure 19: Subjective topic difficulty level for the OOP-First-pupils (main study 2007/2008) ordered by sequences](image)

*Figure 19: Subjective topic difficulty level for the OOP-First-pupils (main study 2007/2008) ordered by sequences*  

In the OOP-first course, students perceived the difficulty of topics in a somewhat alternating sequence: A more difficult topic is followed by an easier topic, and so on.

![Figure 20: Subjective topic difficulty level for the OOP-Later-pupils (main study 2007/2008) ordered by sequences](image)

*Figure 20: Subjective topic difficulty level for the OOP-Later-pupils (main study 2007/2008) ordered by sequences*  

In the OOP-later course, students perceived the difficulty of the first three topics (T2, T5, and T6) as slowly increasing. The following topic 7 (arrays) is perceived as easier; at this point begins a second slow increase of perceived difficulty (see the corresponding lines in figure 20). Topic 7 is an exception, as it is perceived much easier then it was according to the post test. As mentioned above, we do not know why students in this group rated arrays as being so easy. However, if we put aside this topic, a pattern in perceived difficulty becomes apparent: The first slow increase in topics belongs to procedural topics. Then – beginning with topic 1 – object-oriented concepts are
introduced, which is perceived as an easier topic and here starts the second slow increase of perceived difficulty.

In general the pattern in the OOP-later groups looks more solid, but we are not sure whether this involves pedagogical advantages.

In addition, the perceived difficulty in both courses is on a reasonable level around the medium between "too easy" (1) and "too difficult" (5).

6. DISCUSSION

The main result of the study is that there are no differences between OOP-first and OOP-later with regard to learning gain.

Differences were found, however, concerning the perceived difficulty and learning climate, but as the overall results are the same, the study does not reveal any effects of these differences in perception of difficulty.

Based on our discussion of OO-first vs. OO-later and our concept that the main difference is the sequence, we therefore conclude that OO-first vs. OO-later is of less importance than the debate suggests. Instead, the debate should focus on the details!

A more detailed analysis mostly supported our general conclusion:

- Most difficult topics remain unaffected by the sequence – they do not become easier when a different teaching sequence is used (section 0).
- The two most difficult topics of the courses are arrays and association. It is not the case that object-oriented or procedural concepts are more difficult in general. The difficulties itself lie beyond OO vs. non-OO (section 0).
- The results in questions on OOM and dynamic OOP (section 0) give some support to the notion that the results of this study on OOP-first vs. OOP-later can be generalized or transferred to other instantiations of objects-first. Such a generalization is also supported by some other studies.

Three questions remain for further studies – they are all related to the difference in measured difficulty via tests and the perceived difficulty, as indicated by the students in questionnaires after each topic was taught. Possibly perceived difficulty is a measure of the learning process itself, whereas the tested difficulty is a measure of the learning outcome. However, there is some evidence that perceived difficulty of learning is part of the comfort level and as such influencing the learning outcome. This leads to three questions for further studies:

- We cannot explain why the perceived difficulty does not match the ‘objectively’ measured difficulty of topics in the post-test.
- We cannot explain the differences in perceived difficulty of the OOP-first and OOP-later group. We have some ideas concerning this difference, though, which could be tested in further work. It may be that OOP-later is a teaching approach leading to teach in a stepwise fashion, where each step is separated from the other – and OOP-first may be an approach where it is more likely that topics are taught in a more integrated fashion, with two effects: It leads to the impression to be able to move forward more quickly, and it can lead to the impression on the learners' side that the topics are difficult.
- We cannot explain why the gap in perceived and tested difficulty is extreme for the topic arrays, and why it occurs only in the objects-later group.

A last issue refers to interpreting the results, the construction of empirical instruments in the current study, and for further work in this area. The issue can be named as a lack of (explanatory) theories. Such theory would be useful to frame the discussion, in order to examine the above mentioned educational issues with more depth, and to allow more effectively to link current research to the existing body of knowledge. Some work in the field might be useful in this regard: the Block model [23], aiming at a general description of dimensions and levels of program understanding and learning programming; the Object Interaction Hierarchy [1], describing competence level of understanding the interaction of objects, and the SOLO-taxonomy, describing general levels of understanding[14], [15] (see also the short discussion of results in section 0).

7. REFERENCES


8. APPENDIX

8.1 Explanation of Topics (Syllabus)

T1 (Introduction in the OOP with BlueJ): Introduction into concepts, instantiating objects from pre-defined classes, using methods, changing attribute values.

T2 (Simple data types): Introduction in IDE (JavaEditor, http://lernen.bildung.hessen.de/informatik/javaeditor/index.htm) variable, constant, data types, control structure sequence (sequences of statements).

T3 (Class and object): Design and implementation of classes with attributes and set- and get-methods, constructors, instantiation of objects. UML-class diagram and UML-object diagram.

T4 (Methods): Method-bodies (with some more algorithmic aspects compared to T3), actual and formal parameter, return values.

T5 (Selections): Selections (if, if-else, case) including Nassi-Shneidermann-diagrams.

T6 (Loops): Loops (do-while, while, for) including Nassi-Shneidermann-diagrams.

T7 (Arrays and Strings): one- and two-dimensional arrays with different primitive data types, from char-array to String-class.

T8 (Inheritance): Design and Implementation of class-hierarchies, abstract methods and abstract classes. UML-class diagrams.

T9 (Association): Design and Implementation of relationships between classes (1:1 and 1:n), static attributes and static methods. UML-class diagrams.

8.2 Explanation of test questions

T1 and T3: Why T1 (Intro to OO with BlueJ object inspector) and T3 (class and object) together? These two topics were tested in one single block of questions, in order to avoid asking questions specific to the BlueJ environment. Understanding of concepts (T1) was tested by asking questions related to T3.

Additional dynamic OOP: With static OOP we refer to T3 (design and implementation of classes, instantiation of objects – but with ‘primitive’ set- and get-methods only). With dynamic OOP we refer to instantiation of more than one object, message passing, dynamic change of object structures. Static OOP is OOP without interaction of objects, dynamic OOP is OOP with interaction of objects including nested method calls, iteration though object structures built by associations between several objects (expressed in terms of the Object Interaction Hierarchy [1], static OOP is on a level below 1, and dynamic OOP is on level 2).

Additional OOM: Task was to design in UML a class model including inheritance and association.

Validity of the test: Overall the test questions evolved from regular exam questions, where experiences show that these types of questions are understandable by the students. The test questions were constructed in several steps: First for each topic (T1 to T9) learning goals were precisely defined. Second, for each goal test questions were constructed as indicators for the goal (based on the given experiences of exam questions). Third, the each question was assigned to a difficulty level (based on Bloom’s taxonomy) and checked that each Block of questions (for each Topic) in sum had the same level of difficulty.