Minimally Invasive Programming Courses –
Learning OOP With(out) Instruction

Marc Berges
Technische Universität München
Fakultät für Informatik
Boltzmannstr. 3
D-85748 Garching, Germany
+49 (89) 289 17353
Berges@tum.de

Peter Hubwieser
Technische Universität München
Fakultät für Informatik
Boltzmannstr. 3
D-85748 Garching, Germany
+49 (89) 289 17350
Peter.Hubwieser@tum.de

ABSTRACT
In this paper, we describe a research project that investigates how far freshmen at the University (without any programming background) are able to learn object-oriented programming with as little (human) instruction as possible. We designed specific tasks for programming assignments and supporting worksheets that contained the only information input that the students received during the courses. We examined the program code the students produced in order to assess the quality of their products. The surprising result was that most of the students were able to write quite satisfying programs. Additionally, a cluster analysis of the results showed that there are two different types of students: the ones that accept and apply the object-oriented concepts quite willingly, while the others prefer to program in a more traditional, procedural style.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education] Computer science education

General Terms
Algorithms, Human Factors, Languages, Experimentation

Keywords
Programming projects, higher education, programming novices, self-conducted learning, object-oriented programming

1. INTRODUCTION
In order to investigate the limitations of self-learning in the context of object-oriented programming, we invited the freshmen of our Faculty of Informatics to participate in small programming courses that took place before the start of the regular courses of their first semester. The basic idea was that there should be no direct human instruction at all. The only information the students would receive would be written on three worksheets.

Every year, our Faculty welcomes about 300 freshmen that start their computer science studies. Unfortunately, most of them don’t have any previous experience in programming. Obviously, this causes many serious problems during the first year’s courses. On the one hand, these students often have problems following the more theoretical lessons, because they don’t have any idea how to apply all the theory. On the other hand, there isn’t enough time left to practice programming in order to improve their skills. Much too often, they end up dropping out of the studies. Due to this situation, many freshmen welcome our invitation to the programming courses.

Our projects started shortly before the academic year 2008/09. Preceding the academic year 2009/10, we organized a second generation of introductory programming courses (called “pre-projects”). Altogether about 300 students or 50% of all freshmen participated in these courses. Additionally, we offered similarly organized courses with more challenging tasks to the students after their first semester (“post-projects”). This second type of projects will be described in another publication.

The program code written by the students helps us to investigate the learning processes of programming novices. Additionally, we will conduct a longitudinal research study by comparing the results of the projects over several years. This might show how the programming skills of our novices will vary over several generations of students. Particularly interesting will be the result of the academic year 2011/12, because at that time we will welcome the first student generation that has attended the new compulsory subject of computer science at the secondary schools of our state, whose curriculum was designed mainly by our working group [5].

In this paper, we describe the design of the pre-projects and present the first “ad-hoc”-results in order to show how successful the students have worked. Additionally, we show some possible research questions that might be answered by more closely investigating the programs of the students.

2. RELATED WORK
In the last few years the engagement of the students in the introductory courses was a central topic for many CS instructors. After the improvement of the lectures, the focus was shifted to programming issues. In 2009, Hansen and his colleagues analyzed a student survey about the engagement and frustration in
programming projects that attended their CS introductory courses. They tried to find out, which of their assignments are considered to be engaging. [8]

Another major discussion that is fundamental for our work is the question about the role of programming in introductory computing courses. Henry Walker emphasizes the need of programming skills as an “essential vehicle” to achieve the problem-solving skills [9]. This is one of the central ideas behind our courses.

We are trying to figure out how much instruction is necessary to understand and use OOP. One of the first projects dealing with “minimal invasive” instruction was the “Hole-in-the-Wall”-project of Sugata Mitra. In [10] he introduced a model on how children acquire computing skills without supervision. We transferred this idea to programming skills.

The working group around Jürgen Börstler evaluated OO Example Programs for CS1 [11]. The evaluation of the program code that the students produced is one of the future goals of our work.

3. THE DESIGN OF OUR COURSES

The students were divided into small groups of 10-12 individuals that shared a room and a tutor, but still each student worked individually on his/her own assignments. Unlike the problems encountered by novice pair programmers, described in [12], we wanted to investigate the limitations of self-guided learning in OOP.

Each group was coached by a tutor, usually an experienced student of the 5th or higher semester. The coaches were advised to help the students with practical tips or by explaining the worksheets or the IDE, but were strictly told not to give any instructions beyond this. Besides the tutor, the students had access to instructional sites on the internet. Forced by organizational circumstances, we had to limit the working time of each student group to two and a half days. During this very short period, every student had to complete his/her own small program. Due to the requirements of the curriculum of our faculty, we had to use Java as the programming language. Following didactical guidelines, we suggested to use the BlueJ-IDE to develop and test the first classes. After the first steps, the students had the choice to continue their project work using Eclipse.

The courses took place four weeks before the regular semester started. At the time of registration for their projects, the students had to self-assess their programming skills. They were asked to assess their programming experience in one of three levels: (1) “I have no experience at all”, (2) “I have already written programs”, and (3) “I have already written object-oriented programs”. The working groups were composed homogeneously in line with these levels, as we had designed a specific type of assignment for each level. The groups of level 1 were asked to program a “Mastermind” game, while the groups of level 2 were required to write a tool for managing game results, e.g. within a football league. The groups of level 3 were given the task of creating the dice game “Yahzee”.

This paper is focused on the results of the students of level 1 (“Mastermind”), because this group represents the most interesting type of participants for us: the “absolute beginners”. For this group, the preconditions are very clear. The students of the other two levels had written some programs before and thus had already worked with various programming languages.

Therefore, their prior knowledge is very difficult to assess. The “Mastermind”-task was formulated as follows:

One of two players sets a four-digit number. The other player tries to find out which four digits from zero to nine are building the number. Therefore, she/he places a four-digit number as a guess. After each guess, the first player responds (1) how many digits are correct and (2) how many digits are correct and additionally in the correct position. The goal of the game is to guess the right number as fast as possible, but at least in twelve steps.

4. THE WORKSHEETS

The main idea of the pre-projects is the absence of any instruction by a teaching assistant, because we wanted to figure out the limitations to self-guided learning on the topic of object-oriented programming. We also did not want to interfere with the theoretical introductory courses of the regular curriculum of our faculty. Despite this, we had to give the freshmen just sufficient information to enable them to complete their projects. Therefore, we created three worksheets [13], which contained all relevant information they required for this purpose.

In their paper [2] Schulte and Ehler compared the “objects-first” and the “objects-later” strategies and defined a list of object-oriented concepts, that form a typical teaching sequence leading to the ability of object-oriented programming: Classes and objects – Attributes (incl. data types) – Methods (incl. control statements) – Inheritance – Association. We designed our worksheets following this sequence. The items “Inheritance” and “Association” were omitted, because we did not expect the students to understand them without instruction.

The first worksheet gives a short introduction to the basic concepts of object-orientation: objects, classes, attributes and methods. Furthermore, it presents the classification of objects and the representation of classes in UML. One of the most important issues of the first worksheet is encapsulation or rather information hiding in object orientation. This is one of the central principles, which the freshmen should internalize as soon as possible.

The second sheet presents the implementation of the basic concepts of object orientation in Java, which are described textually, as well as by syntax diagrams, as shown in [1] (see Figure 1). It explains the concepts of classes, constructor methods, attribute declaration and initialization, method declaration with or without parameters, data types and value assignments. The explanation of declaration, initialization and the usage of arrays are the most important issues of the worksheet. Schulte and Ehler argue in [2], that arrays are one of the most difficult concepts of programming in general.

![Figure 1: Syntax diagram of Java class structure](image-url)
The third worksheet presents the concept of algorithms and their control structures (sequential and conditional processing, repetition) as well as their implementation in Java. Additionally, the call of methods and the creation of instances of objects were explained. Finally, the application of a "main"- method and the standard input and output mechanisms in Java were described. For the implementation of user input we offered a specific framework to the students.

Due to the very short time the students had left to learn programming we had to focus on the most important concepts of object-oriented programming in Java. That was the main purpose of our worksheets. If we had presented a textbook to the students, as it is practiced in the regular introductory course in the first term, the students would hardly have been able to figure out what information is relevant for their projects. Using the worksheets, we could define exactly what information the students would receive.

5. GATHERING OF DATA
The first and most obvious question was, whether the absolute beginners would be able to write a "really working" Java program at all during the very short time of two and a half days. The students were asked to fill out a questionnaire in order to get their personal variables.

5.1 Drawing the sample
The complete population of the second generation of the courses contained the code of 93 programs. As we wanted to compare the code of programs from students with similar prerequisite knowledge, we restricted our investigation to a sample of 37 projects by the following three criteria.

The first criterion was the supposed previous subject domain knowledge in computer science (beyond pure programming skills), which turned out to be very heterogeneous. To ensure that all members of the sample had similar previous knowledge, we restricted the sample to all freshmen that had stated that they had not attended any prior courses in computer science or informatics at school and that graduated from a school within our state, where computer science or informatics was not mandatory for the students that graduated (or will graduate) before 2011. This way we assured that all members of the sample had the same prior school education in computer science (namely none). Indeed, this was the majority of our population.

The second restriction was formed by the self-assessed experience in programming. We restricted the sample to all students that classified themselves to the programming experience level 1 ("I have no experience at all", see above), as the previous programming knowledge of the students of level 2 and 3 was very different, e.g. concerning the different programming languages and styles they had already used. The restriction to the (most interesting) "absolute beginners" made the sample quite homogeneous.

One of the problems encountered in the first generation of the projects in 2008 was the assignment of the prerequisite knowledge of a certain student to its program. Without knowing the prior knowledge it is hardly possible to evaluate the results the students produced.

This problem was solved in the second generation of the projects in 2009 by numbering the questionnaires and inserting this number in each class file as a comment. By using this method it was possible to match the program code to the answers on the questionnaires and thus to the corresponding prerequisite knowledge without sacrificing anonymity.

5.2 The questionnaire
The personal questionnaire was composed of four sections, where the second section contained three subsections. For this paper, only the first section with the personal data and the third section with the self-assessment of the students are relevant. The second section asks for the opinion about the organization and the fourth section is open for individual comments and critical remarks.

We collected the following personal data: gender, age, previous knowledge, prior school education, and state or country of origin.

In the self-assessment section, we asked the following three questions:

SA1) “How much did you learn in the pre-projects?”
SA2) “How do you estimate your Java knowledge after the pre-projects?”
SA3) “Did you understand the concept of object-orientation?”

6. EVALUATION OF THE PROGRAMS
After drawing our sample described above, we inspected the program code of the corresponding 37 Java projects.

6.1 Syntactical Correctness
The first aspect we investigated was the syntactical correctness of the programs. For this purpose, the programs of each student were compiled as Eclipse-projects, including all necessary libraries.

32 of the 37 examined programs could be successfully compiled after importing them into Eclipse. Two programs were missing the corresponding class for the input framework. After inserting this class, these two programs were also compiled successfully. One program even contained a first attempt to implement a graphical user interface. For this purpose a class from a BlueJ example project was used. After importing this class into Eclipse, this program was also compiled without any problems. Two programs could not be compiled: one because of a small syntax error, the other because it was just fragmental code without making sense.

Although the time for programming was very short and there was a complete lack of instruction by a teacher, all but two programs from our sample could be compiled, some with minor modifications to the code. Nevertheless, we have to take into account, that modern code editors like Eclipse offer many hints that allowed the students to solve syntactical problems following a “trial and error” strategy.

6.2 Fulfillment of the specifications
The fact that nearly all programs could be compiled is only a very weak indicator for the success of the pre-projects. Another more convincing argument could arise from the correctness of the “Mastermind” programs.

Therefore, we tested the program code with two exemplary inputs, one with a special case of four identical digits, e.g. “1111”, the second with a quite general form of user input (four different figures to guess, e.g. 3456). A big challenge for the programmers was the calculation of the number of right positions in the special
case. If the user guesses only one of the four (equal) digits in the number, the program has to return correctly only one right digit position. The rest of the digits are still not guessed correctly. Only 18 out of the 37 programs could handle numbers with identical digits. In some programs, numbers with two or more equal digits were not accepted at all. In the programs where the computer generated the input, mostly equal digits were not accepted.

The second, more general input example was 3456, which was correctly processed by 26 out of the 37 programs. These programs returned the correct amount of digits and positions. Eleven programs did not accept this input at all. One had a problem with the computer-generated number. Another one had only one user input at the beginning and continued without any further user input. A third one could perform only one guessing action. The other projects did not calculate the correct number of positions or digits.

Half of the programs took the number to be guessed from user input, while the other 19 generated the number by random. For this purpose, the students had to find out how to apply the random function of the Math package in Java.

In summary, regarding compilation and functionality, about 50% of the tested projects worked absolutely correct, both with the input “1111” and “3456”.

6.3 Implementation of programming concepts

Another very interesting question was which of the concepts that were explained on the three worksheets would be applied by the students in their programs. We extracted the concepts from the worksheets, ordered them in a list by their appearance on the sheets and investigated, which of them were applied by the students. The list looked as follows:

C1) Order of attributes, constructors and methods in class definitions: On the second sheet we presented a code sample that showed the correct order of attributes, constructors and methods in a class definition.

C2) Initialization of attributes with default values in the constructor: All attributes are set to a default value; arrays are initialized.

C3) Constructors with parameters: In the section about constructors on the worksheet the possibility of overloading a constructor was shown.

C4) Initialization of attributes with default values: We stressed the initialization of all attributes before they are used.

C5) Return values of methods: Did the students use the “return” statement to pass values from methods or did they only operate on global variables?

C6) Parameters of methods: Did the students use input parameters or operate on global variables or attributes?

C7) Access modifiers of methods: Not all methods have to be public. Did the students mark some of them as private?

C8) Arrays: Did the students use arrays?

C9) Self reference by “this”: Using the same name for attributes and parameters in methods can improve the readability of the code. Did the students use “this” in constructors or methods?

C10) Main-method: At the beginning of the pre-projects, the students used BlueJ. This IDE allows the direct interactive call of methods without implementing a “main” method. At the end of their work, the students were asked to execute their programs without the usage of BlueJ, so they had to implement a “main”-method.

C11) Conditional statement The students should use the “if”-construct with “else” if necessary (instead of two separated “if”-constructs).

C12) “While-Do” repetition: “While-Do” was the first form of repetition that was explained on the worksheets.

C13) “Do-While” repetition: The “Do-While” repetition was presented on the worksheets after “While-Do”.

C14) “For” repetition: The “For”-repetition was the last form of repetition on the worksheet. The students should use this type if the total amount of repeats is known already before the first one starts.

Figure 2 shows how many of the 37 inspected projects used the concepts C1 – C14, ordered by the frequency of usage. Obviously, some of the concepts are used very rarely, e.g. C3 (constructor method with parameters), which was implemented by only one student. Several other concepts were also not applied very often: C6 (parameters of methods), C13 (do-while), C9 (this), C7 (access modifiers of methods), and C5 (return values).

Figure 2: Frequency of the Implementation

On the contrary, the concepts C8 (arrays), C10 (main method) and C12 (while do) have been applied in most of the programs. Thus, these seem to be relatively easy to understand.

The interpretation of the remaining concepts is left to the next section, where conceptions of object orientation will be discussed.

In order to identify different types of students concerning their attitude towards object orientation, we performed a cluster-analysis on the matrix of dichotomous data that is formed by the criteria C1–C14 as columns and the programs as rows (1 in the cell with index ki means that criterion k is applied in project i). The cluster analysis was carried out by the Gnu R software [7]. To make the interpretation of the results more convenient, we rearrange our criteria C1–C14 in three different groups (see Table 1): (A) Algorithmic concepts and their implementation in Java, (F) Functional concepts, (OO) object oriented concepts.

Table 1: Renaming and Reordering for the Clusteranalysis

<table>
<thead>
<tr>
<th>Name</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>F3</td>
<td>OO1</td>
<td>OO2</td>
<td>OO3</td>
<td>OO4</td>
<td>OO5</td>
</tr>
<tr>
<td>Concept</td>
<td>C6</td>
<td>C1</td>
<td>C2</td>
<td>C7</td>
<td>C10</td>
<td>C9</td>
</tr>
</tbody>
</table>

C3
The concept C3 was omitted for the cluster analysis, because there was only one student who applied it. We also omitted C13, because the “do-while” repetition was not suitable to the regarded “Mastermind” programs.

As can be seen in Figure 3, the cluster analysis resulted in two groups. The y-axis represents the probability that the concepts on the x-axis are applied in a project that belongs to cluster 1 or cluster 2 respectively. Cluster 1 contains eleven projects (30%), cluster 2 26 projects (70%). It is evident that concept OO4 (C10: main-method) is differently applied than the other OO-concepts. This might result from its ambivalent character between object-oriented (as a “starter” for the creation of the objects) and procedural programming (C programs need main methods, too). Thus, it is applied similarly often by both clusters.

**Figure 3: Cluster analysis over the applied concepts**

Concerning the other concepts we received a quite interesting result: The students, whose projects belong to cluster 1 seem to have more problems in applying the object-oriented concepts OO1, OO2, OO3 and OO5, but are relatively strong in some of the functional and in all the algorithmic principles. The students of cluster 2 applied most of the concepts quite frequently, except for the functional F2 and F3. The distinction between the two clusters is mainly caused by the concepts OO1 (order in class definitions) and OO2 (initializing attributes in constructors).

Both clusters have nearly the same gender proportion as the total population of students (62% male, 38% female).

### 6.4 Characteristics of “Object Orientation”

The main purpose of our pre-projects is to investigate the learning process of object-oriented programming (OOP), see [6]. But what is OOP? Unfortunately, many different definitions of OO can be found in literature. Because there is no common definition, it is very hard to state which features a program should have in order to be regarded as “object oriented”. Nevertheless, we looked for a set of criteria that allows us to compare programs regarding their “degree of object orientation”. At first, we give a short definition of an “object”:

“An object is, in simple terms, composed of two types of information, the data and logic information components. An object does not have to have both types of information simultaneously, and can contain data and logic or just logic. The object must contain logic, because, as we shall see, the data component cannot be accessed except via the logic component. The data component is optional.” [4, p.42].

Unfortunately, for a comprehensive view on our programs this simple definition does not suffice. Thomas and Sanders gave a summary of concepts and misconceptions regarding OOP in [3]. Most of the misconceptions discussed by them are irrelevant for small projects like ours. Nevertheless, we picked five of their concepts that are relevant even for programs of the scale of our pre-projects:

- **(O1)** Constructors are defined and used
- **(O2)** Methods other than constructors are defined and used
- **(O3)** Methods’ return value are used
- **(O4)** The attributes are encapsulated
- **(O5)** At least one instance of a class is created

In figure 4, we give an overview how far these five concepts were implemented in our programs. As you can see, nearly all students applied O1 and O2. In most programs, a constructor was defined, but as we stated in section 6.3 only one student wrote a constructor with parameters. The concept that was used the least was O3. Only a few students used return values for their methods. Most of them worked with global variables instead. At this point, there seem to be difficulties in the understanding of the concept of methods. O4 and O5 deal with information hiding and instantiating. Therefore, they are one of the central concepts of OOP. More than 60% of our students implemented these concepts.

**Figure 4: Implementation of the OO concepts**

According to the five OO-concepts above, we tried to perform a grading of the degree of “object orientation” of the programs. We simply counted how many of these 5 concepts where implemented in each program and graded the program from “very poor” (0 concepts applied) to “perfect” (all 5 concepts applied). As shown in figure 5, three programs have applied all five concepts. In summary, we found 1 very poor, 6 poor, 18 average, 9 good and 3 perfect programs, concerning their degree of object orientation.

The large number of average programs correlates to the self-assessment of the students concerning the knowledge in Java programming they gained during the pre-projects
6.5 Self-assessment of the students

To get an idea how the students assessed their programming abilities after the courses, we asked them three questions which are listed in 5.2.

Each question could be answered on a scale from one to five. We calculated the mean values for the whole sample as well as for each of the two clusters. The results are shown in Table 2.

The question SA1 could be answered from 1 “learned a lot” to 5 “learned only a few things”. As can be seen in table 2, the results (average of 2.2) were the same for the whole sample as for both of the clusters.

<table>
<thead>
<tr>
<th></th>
<th>Sample</th>
<th>Cluster1</th>
<th>Cluster2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>2.2</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td>SA2</td>
<td>2.1</td>
<td>1.7</td>
<td>2.2</td>
</tr>
<tr>
<td>SA3</td>
<td>2.4</td>
<td>2.3</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 2: Mean-Value of the Self-Assessment

In question SA2 the difference between the clusters and the whole sample is much clearer. The possible answers started from 1 “can program on my own” to 5 “no Java knowledge present”. The self-estimation of the Java-knowledge in cluster 1 is significantly better than that of cluster 2. The members of cluster 1 think they have understood more of the programming concepts. It might be that the stronger algorithmic skills of the students of this cluster led to a better functionality of their programs, which resulted in a higher satisfaction of the programmers.

Question SA3 that could be answered from 1 “understood OO” to 5 “did not understand OO” also shows the result of the cluster-analysis. Cluster 1 rated their understanding of the OO-concepts a bit lower than the members of cluster 2. This is quite surprising regarding the different application of object-oriented concepts in the two clusters.

In the next step of our studies, we will have to prove that there is a real correlation between the self-assessment of the students and the analysis of the results of their work.

7. CONCLUSIONS AND FUTURE WORK

Although the tutors were advised quite strictly not to instruct the students, their contribution to the students’ products is still very difficult to assess. One of the most interesting questions which has to be answered in future work is how much help is to be given by them really. Therefore, we are going to match the tutors to the questionnaires. To avoid a lack of anonymity we will assign a color to each tutor. With this matching we will find out if there is a correlation between the cluster the student belongs to and the tutors that helped him/her. The questions the tutors are asked by the students will also be gathered in the next semester.

After evaluating more than 300 questionnaires of the participating students we can state that most of the students appreciated the pre-projects and suggested that they had learned something essential (about 90% in the first run). Most students were really surprised that they were able to learn programming in only two and a half days. Nearly all students developed a syntactical correct and more or less “working” and object-oriented program. One of the goals was to reduce the shyness from programming. This goal was apparently reached quite well.

Our next study will investigate how the pre-projects have influenced the scores of the students during their first year at the university. The results we gathered during our project will be validated in the context of a regular CS1-course. Therefore, we have to partly give up the anonymity in order to get a correlation with the results of the CS1-exams.

Another large study will cover the structure of the learning processes that are reflected by the object-orientated programs of the students.

8. REFERENCES


