The Gap Between Knowledge and Ability

Marc Berges  
TUM School of Education  
Technische Universität  
München  
Boltzmannstr. 3  
85748 Garching, Germany  
berges@tum.de

Andreas Mühling  
TUM School of Education  
Technische Universität  
München  
Boltzmannstr. 3  
85748 Garching, Germany  
muehling@in.tum.de

Peter Hubwieser  
TUM School of Education  
Technische Universität  
München  
Boltzmannstr. 3  
85748 Garching, Germany  
peter.hubwieser@tum.de

ABSTRACT

We present the results of an investigation on how well students are able to understand object-oriented programming (OOP) when learning with only very minimal guidance. We analyzed the source code that the students of a preparatory course produced during the course as well as concept maps that were asked to draw before and after the course. Our findings show that there are observable differences between what students know about some concepts and what they're able to do with it. Generally speaking, it seems that several OOP related concepts can be applied successfully without fully understanding the underlying concepts, while others are hard to understand and apply without a significant amount of prior knowledge. This gives rise to the suspicion that it might be possible to apply a concept without having understood it, at least with respect to some algorithmic concepts of CS.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education—Computer science education;  
K.3.2 [Computers and Education]: Computer and Information Science Education—Curriculum;  
K.3.2 [Computers and Education]: Computer and Information Science Education—Information systems education

General Terms

Experimentation, Measurement

Keywords

empirical study, computer science education, object-orientation, concept maps, CS1, code analysis, self learning, cluster analysis

1. INTRODUCTION

One of the main aspects that distinguish school education from education in universities is the fact that students are expected to learn on their own to a much greater degree, especially if learning content is presented in lectures only. Thus, students must learn early how to actively engage in the learning processes. In this setting, we were interested to find out how well students are able to pick up basics of object-oriented programming (OOP), if they have to learn on their own. Learning how to program is one of the most difficult tasks that new CS students face, if they have no prior experience with it. Especially, when starting with OOP, there are many learning objectives that have to be mastered before the students can complete their first program, as we have demonstrated [11]. We were interested in finding out, how this process works at the very beginning. Do the students understand all the things they are doing (or put otherwise: are they unable to put into practice things that they don’t understand)?

We developed a preparatory course for teaching basics of OOP to freshmen, before the start of their first semester. Following 2008 and 2009 the third generation of this course was held in September 2010. Similar to the first two years, all freshmen starting their studies of computer science were invited at the time of their registration. Now, that the course is established and running, we began to evaluate the learning processes of the students. Our main interest is how the students knowledge and abilities are affected by it.

The rest of the paper is organized as follows: The next section will present relevant theoretical background for our research. Section 3 will present details of the course and the data that we were collecting from it. Section 4 will present the results of our analysis which are discussed in Section 5. Finally, there will be some insights into further research.

2. THEORETICAL BACKGROUND

Our research mainly focuses on certain aspects of declarative knowledge as well as the students’ ability to write object-oriented(OO) programs. Concerning the representation of the specific subject domain knowledge of object-oriented programming, Pedroni and Meyer [23] proposed to organize it in Trucs, (testable, re-usable units of cognition) which are collections “of concepts, operational skills and assessment criteria”. Following this definition, we are mainly concerned with the first two parts of this definition. Note, that our work is based on research we were conducting last year, in which we were trying to find out more about the structure of CS content knowledge of students ([14], [13]).

We presented results of last years’ courses [12], however focusing on different aspects and only considering students without any previous knowledge in programming.
2.1 Learning Outcomes

In the end, we are investigating the results of learning processes. Following Anderson and Krathwohl \[3\], we regard such learning outcomes as a combination of a certain type of knowledge and an observable behavior specification (called cognitive process) concerning this type of knowledge, together forming the two dimensions of their taxonomy:

- The **knowledge dimension** is partitioned into A. factual, B. conceptual, C. procedural, and D. metacognitive knowledge,
- the **cognitive process dimension** contains the following levels of behavior: 1. remember, 2. understand, 3. apply, 4. analyze, 5. evaluate, and 6. create.

In order to categorize the knowledge that might be relevant to our research, we rely on the knowledge dimension of this taxonomy. The four categories are described there as follows:

1. **Factual Knowledge**: “basic elements that students must know to be acquainted with a discipline or solve a problem in it”.
2. **Conceptual knowledge**: “the interrelationships among the basic elements within a larger structure that enable them to function together”.
3. **Procedural knowledge**: “how to do something: methods of inquiry, and criteria for using skills, algorithms, techniques and methods”.
4. **Metacognitive knowledge**: “knowledge of cognition in general as well as awareness of one’s own cognition”.

A comparison of two different definitions by J.R. Anderson \[2\] respectively L.W. Anderson and Krathwohl \[3\] shows that factual knowledge can be represented by propositions, conceptual knowledge by propositional networks, semantic networks or schemata. These two categories describe both declarative knowledge, but we are interested mainly in the second category.

A recent ITiCSE working group \[7\] elaborated a specific taxonomy for computer science, which splits the cognitive process dimension of the taxonomy of Anderson and Krathwohl into two subdimensions **producing** and **interpreting**. The **producing** subdimension represents the more active part of the learning process and contains the steps **none**, **apply**, and **create**. The remaining activities of the cognitive process dimension are arranged on the interpreting subdimension: **remember**, **understand**, **analyze**, **evaluate**. This results (in combination with the knowledge dimension) in a three-dimensional taxonomy that combines two cognitive process categories of Anderson and Krathwohl to a single category, (e.g. **apply**/**remember**). This implies also that the new subdimensions **producing** and **interpreting** are independent, in particular that it should be possible to apply a concept without having understood it. This has to be investigated by empirical research, which might be extended by the results we present in this paper.

2.2 Concept Maps

Concept maps have long been established as a teaching and assessment tool in many subjects. Going back to Novak and Ausubel’s theory of meaningful learning \([15], [16]\), there have been many alterations to the original concept mapping task (e.g. a concept map usually doesn’t need to be hierarchically organized in contrast to the original description of a “good” concept map \[17\]). Concept maps are essentially a labeled, directed graph with nodes representing concepts and edges symbolizing associations between these concepts. The labels of two incident nodes together with the label of the connecting edge forms an association - the basic element of a concept map. For example “Object - is instance of - Class”. Although the main objective of the taxonomy of Anderson and Krathwohl \[3\] is to assess learning objectives, it is suitable as a taxonomy for assessment tasks. Therefore, the concept maps of students yield an externalization of their conceptual knowledge (with the cognitive process re-member or understand). Clearly, this externalization might be influenced by motivation, by the focus of attention or by many other external influences \[22\].

Sanders compared the knowledge of students (concerning object-orientation) in several nations using concept mapping techniques \[23\]. This serves as an example of a multitude of concept map assessment tasks that are published - especially outside of CS Education.

Usually, the students start drawing concept maps from a list of given concepts that they have to pick as nodes and then to connect by edges \[25\]. However, the assessment task itself can consist of various student activities: The student can be asked to draw a concept map or to complete a given basic map. When constructing a map, the list of concepts and/or the list of possible edge labels can be restricted or not. For a comparison over different methods, see the work of Ruiz-Primo \([19], [18], [30]\). Concerning the measures that used in those assessment tasks, there are also many existing measures to be found in literature, for example the ones of Shavelson \[27\] or Sanders \[26\], which is validated by McClure \[20\] by correlating it with several existing measures to be found in literature.

2.3 Code Analysis

The analysis of the students code (in other words, their “operational skills” concerning OOP) is another central aspect of our investigations. The analyzing and scoring of object-oriented code is a central topic in educational research ever since OOP has been taught in introductory courses in CS. Börstler \[6\] proposed three categories for the evaluation of OOP example programs following several criteria. Sanders \[26\] introduced a check-list for scoring OOP programs by investigating concepts and misconceptions in OOP. Truong et al. \[28\] built a framework for static code analysis of student’s programs. They summarized common poor programming practices and common logic errors from literature and a survey conducted on teaching staff and students. The framework is working on a XML basis and enables the students to get feedback on their programs and rate them automatically. In our courses the tutors should focus on these tasks. Another investigation of programming projects was done by Hansen \[10\] with the goal of analyzing the data in terms of engagement and frustration. Hanks \[9\] and Robins \[24\] investigated problems that encounter by
novice programmers. Hanks let the students work together while Robins let them program on their own. They list problems in three main categories containing 30 subcategories.

3. RESEARCH DESIGN

3.1 Setting

In autumn 2008 we developed a course at our university for the freshmen studying Computer Science (CS). It takes place just before the first semester. All the students starting their studies are invited during their enrollment process. The participation is voluntarily. The necessity of the course results from the German lecture system at universities. During the semester there are mainly lectures with only very little time for practical experiences. Nevertheless, it is officially communicated that it would be possible to study CS without any prior programming knowledge, which implies that students without such prior knowledge should be accommodated somehow, too. Therefore we have developed and installed specific programming courses for this purpose that take place before the first semester (“pre-courses”, see [12]), analogically to the basic pre-courses in Mathematics.

In this paper we refer to the pre-course of autumn 2010 which was attended by 167 students - about 42% of all the students that were invited. We formed 18 groups, combining 6 to 15 participants each.

All students were asked at the registration to self-assess their prior programming experience in one of three levels:

1. “I have no experience at all”,
2. “I have already written programs”,
3. “I have already written object-oriented programs”.

Based on this information we tried to compose the groups as homogeneously as possible. The demands of the programs the students should realize differed according to their respective level of programming experience. The students of the first level were asked to program a “Mastermind” game. The groups of level 2 should realize a tool for managing results from a sports tournament (e.g. a football league). The groups of the 3rd level should program a version of the dice game “Yahzee”.

The course took two and a half days. All participants worked on their own (instead of in teams), because we wanted to investigate the individual learning outcome. However, the students were actively encouraged to talk to each other. The material was presented in the form of worksheets that contained all the required information. Additionally, each group was coached by an experienced student as a peer tutor. The students were encouraged to approach the tasks in a self-guided learning process. With this, we try to counter the problems Ben-Ari mentions in his paper on constructivism in CS-education [4] concerning the absence of an effective mode of a computer with the individual help of the peer-tutors. These were responsible for helping the students to understand the worksheets and tools, but they were advised not to give any assistance (or instruction) on programming itself. The tutors were also asked to write down every question that the students asked together with the answer that they gave to them.

The four worksheets given to the students are described in detail in our paper on minimally invasive programming courses [12]. The first sheet describes the task itself and the programs that will be used. So the students received a first short overview over the course. The second sheet introduces the basic concepts of object-orientation: objects, classes, attributes and methods. It emphasizes the concept of data encapsulation or information hiding that the students should adhere to as early as possible. The third sheet presents the implementation of those concepts in Java. We chose Java as the programming language, as this is the language they will be mainly dealing with in the first semesters. The concepts are described by text, as well as by syntax diagrams, taken from a book of Müller and Weichert [21]. The last sheet presents the concept of algorithms and the imperative control structures (sequences, conditional statements, loops). We suggest, that the students use Bluej due to the reasons mentioned by Bergin [5] for their first steps in programming. Towards the end of the course, they had the choice to switch over to Eclipse 2.

3.2 Data collection

The students of the course were divided into homogeneous groups with regard to their prior programming experience. To this end, they had to self-assess their programming skills when they registered for the course, as described in the preceding section. In theory, our data for each participant should consist of four parts. At first we collected two concept maps, one drawn before and one after the course, following this a questionnaire dealing with basic personal data and the relevant prior knowledge. Finally, we have the source code of the programs that were written by the students during the course.

At the end of the course we had 167 datasets. We eliminated all datasets with no map in the post-test or no questionnaire. Afterwards we removed all sets without programming code and without complete information about their previous programming experience. By this way we ended up with 77 complete datasets that we could evaluate.

One of the biggest problems in the investigation of the first two runs was the fact, that we could not reliably determine the effect, that the peer tutor had on the results. Therefore, we made sure that each tutor participated in at least two groups: One composed of students with previous knowledge and one composed of students without. Since we are only analyzing the difference between those groups, we can ensure that the tutors have only a very minimal effect on the results.

3.3 Data processing

During our investigation we collected concept maps as well as programming code produced by the students. Each student had to draw two concept maps. One at the beginning and one at the end of the course. Due to the fact, that the students had no access to computers at the beginning of the course, the concept maps were drawn using pencil and paper as well as a given list of concepts. As described in our paper [12], the list of concepts was extracted systematically from the worksheets and encompasses all concepts that the course was dealing with. The list (from now on called CL) is: access modifier, arrays, assignment, association, attribute, class, conditional statement, constructor, data encapsulation, data type, inheritance, initialisation, instance,
loop statement, method, object, object orientation, operators, overloading, parameter, state.

The concept maps were then digitalized. We also assigned a score to each edge/association according to the following scheme:

- If the association is a correct statement it will be scored with 2.
- If the association forms a statement that is clearly wrong or the meaning of the statement could not be understood, it will be scored with 0.
- If none of those two conditions apply, the association will be scored with 1.

Edges with no label were excluded from the analysis altogether. We validated the grading scheme by having 3 experts grade a randomly chosen subset of the edges after having explained the grading scheme to them. We found a correlation between any 2 of the 3 persons to be not less than 0.8, with the most deviations occurring between the grades 2 and 1. Therefore, we skipped edges with a grade of 1 in this analysis.

Besides of concept maps, we analyzed the programming code of the 77 complete data sets that the freshmen had written. Of course, there are many ways in which source code can be analyzed. As we are interested in the correlation to the collected concept maps, we tried to conceive a systematic approach that helps us identify whether or not the concepts of CL were actually applied in the program.

For this purpose we first investigated the possible applications of the concepts of CL. For example, the concept constructor can be present in the form of using a constructor (i.e. by creating an object) or in the form of defining a constructor. By this way we identified for every concept properties of the code that were observable and showed that the respective concept was applied. Afterwards we categorized these properties and represented each category by a yes/no question that could be answered while analyzing the programming code.

The only concepts that we excluded right from the beginning were object-orientation, class, data type, constructor and instance. The first, because in the context of Java we will have object-orientation by design. The next two, because the use of Eclipse makes it impossible to distinguish between “implementation by the students” and “implementation by the IDE”. Finally, the last two were excluded since they cannot be separated clearly (calling a constructor will also create an instance).

We ended up with 35 questions. Some turned out to be trivial, as we could answer them with “yes” in all or nearly all data sets. For example, for the concept method we had, again, the categories of using a method and defining a method, however, every project used a method, because they were all printing something on the console. In the following list we show all these questions, which we will call analysis questions (AQ) from now on. The abbreviations indicate which concept the questions relate to as can be seen in the labels of Fig. 3.

IN1 Is there inheritance from existing classes?
IN2 Is the code using a manually created inheritance hierarchy?
ME1 Is there a method call in the code?
ME2 Is there a method declaration?
AG1 Is an assignment used in the code?
ST1 Is it possible to save the state of an object?
ST2 Is it possible to change the state of an object?
ST3 Is it possible to use the state of an object?
AC1 Is there an association between classes in the code?
AC2 Is there any use of associations between classes?
DE1 Is the visibility of the attributes other than public or default?
OP1 Is the assignment operator used?
OP2 Are there any logical operators used in the code?
OP3 Are there any other operators used, except the assignment or logical operators?
AR1 Are there arrays with pre-initialization declared in the code?
AR2 Are there arrays without pre-initialization declared in the code?
AR3 Is there any access of the elements of an array in the code?
AR4 Is there an initialization with new?
AR5 Are methods of the class Array used in the code?
IS1 Is there an explicit initialization of the attributes?
PA1 Is there a method call with parameters in the code?
PA2 Are there any method declarations with parameters used?
PA3 Are the parameters of a method used in the method?
AT1 Are there attributes declared in the code?
AT2 Are attributes of other classes accessed?
AT3 Are attributes of class accessed within this class?
CS1 Is there an IF-statement without ELSE?
CS2 Is there an IF-statement with ELSE?
CS3 Is there a SWITCH-statement?
OB1 Is there a declaration of any object attribute or variable?
OB2 Is a declared object used in the code?
OB3 Is there a use of the own object with this?
OV1 Is there a declaration of an overloaded method?
LO1 Is there a use of loops?
AM1 Are the access modifiers private and protected used with attributes or methods?

4. RESULTS

In this section we present the data of our investigation and the major findings, which are then discussed in the next section.

4.1 Conceptual knowledge

The maps give us an externalization of (parts of) the students’ knowledge concerning the topics of the course. We were mainly interested in finding a connection between this knowledge and the quality of the source code that the students produced, as well as investigating what influence any (relevant) prior knowledge might have. To this end, we analyzed how the knowledge of the students developed during the course. As we were interested in combining the results of this analysis with the analysis of the source code, we needed a measure indicating whether or not a student “knows” something about a concept or whether or not he (or she) has a “misconception” about a concept. We derived the following basic measure for this: If the concept map of a student has, for a given concept, an incident edge with a score of 0, we
assumed, that the student has a misconception regarding that concept. In this case, the concept was scored with 0. If the concept has an incident edge with a score of 2, we assume that the student “knows” something “correct” about the concept and the concept was scored with 1. For example, of the 501 edges in the pre-test, 150 edges were scored with 0, as well as 201 out of 1015 edges in the post-test.

Clearly, this is a very basic measure and one might expect that the percentage of maps fulfilling this criterion for a given concept is very large. However, as the concept maps were rather small and sparse, it turned out to work well in practice. Using this measure, we get two characteristic vectors for each concept map: for each concept we have a 1/0 score indicating whether or not the map fulfills our criterion for that concept. Thus, averaging the components of those vectors for each concept over all maps gives an estimator for the probability that the students know something about that concept or that they have a misconception about it. Taking those vectors for both pre- and post-test, we get an indicator of how the conceptual knowledge of the students developed on average. Figure 1 shows the final results for both groups in pre- and post-test.

4.2 Analysis of Source Code

Next, we focused on the source code, as ultimately that is what the students produced with the help of what they learned in the course. We analyzed the programming codes of the data sets by answering all the AQs for each project with “Yes” (score 1 for the question) or “No” (score 0 for the question). Since each of the questions is scored by 0 or 1, we were able to calculate an average score for each question over all student projects, which are shown in figure 2. The decreasing order of the questions has only been chosen to improve readability.

The projects differed a lot in their complexity. There were projects with only a few lines of code and some with more than one thousand lines of code containing a GUI and other features. This shows that clearly there were some students who had already extensive knowledge about (object-oriented) programming.

The list of our Analysis Questions (AQ) gives us a straightforward mapping of concepts to code questions. However, a concept may be typically related to more than one AQ, but any AQ belongs to exactly one concept. To get an average score for the application of a concept, which is represented by a set of AQS, we have to find a suitable formula. As we found that there are basically three ways how a concept is represented by several AQS, we could derive three different formulas for the combination of the AQ-scores.

1. The concept is applied, if at least one of these AQS is scored with 1 (e.g. CS1, CS2, CS3). In that case, we combined the scores of the AQS by a boolean OR-function.

2. The concept can be divided into several components and each component is represented by one AQ. In this case the score for the concept is calculated as the averages over the scores of the AQS (e.g. ME1, ME2).

3. There is a clear hierarchy between two questions (e.g. to use the parameter of a method, you have to declare
a method with parameters first). It turned out that in these cases the most suitable score for the concept is also the average over both questions (see case 2).

By using these three methods (in rare cases a sequential combination of them), we finally derived a mapping from the scores of the AQs belonging to a single concept to a value between 0 and 1 that can be regarded as a score that indicates the application of this concept in a Programming project. Applying this scoring scheme, we were able to compare the scores from of source code that indicate the application of the concepts to the scores of the concepts in the maps, indicating how a concept was understood by the students. The results of this comparison are shown in Figure 3. For the post-test we plot the mean-values of how the students understood a concept (as described above for Figure 1) in contrast to the score obtained from their implementation of this concept.

5. DISCUSSION

5.1 Development of the knowledge

Looking at Fig. 1 there are several developments worth mentioning. First, students without previous knowledge have a bigger difference in most concepts, between the pre-and post-test, i.e. they learned more. This is to be expected, of course. It is noteworthy, however, that the other students also show an increase in most concepts and that the two post-tests are actually looking very similar. So, from the very heterogeneous group at the beginning there is not much left after the course. Since that was the main goal (preparing students such that they all start into their first semester with equal opportunities), we see this as a indicator for success. The students with previous knowledge also show a better understanding of all concepts after the course, except for conditional statement, inheritance, instance, operators and parameter, where is not much difference. So even though those students haven’t been the main focus of the course design, they were able to take something from it. Interestingly, they seem to “lose” some knowledge regarding instance, operators and parameters. However, since our scoring method not very specific, this might for example happen when the students had a very “simple” proposition in the pre-test and then tried a more complex one in the post-test which was incorrect. Additionally, students without previous knowledge seemingly learned the most about “core” concepts of object-orientation (the four concepts with the biggest increase are attribute, class, method and object). As the course material put much emphasis on those concepts, this is to be expected.

Looking at the difference between the pre- and the post-test, there are some concepts (inheritance, initialisation, instance) where students without prior knowledge learned almost nothing. So it seems that those concepts need a certain level of understanding, before students can correctly incorporate them into their mental model. Since the students without previous knowledge were acquiring this basic knowledge in the course, they were unable to focus on the more advanced concepts, too. It may also indicate, that the course material is not suited for explaining advanced concepts to the students without any prior knowledge, though.

Next, we take a closer look on the misconceptions (i.e. edges rated with 0). Generally, the distribution of misconceptions follows the distribution of the knowledge. So, whenever students know more about certain concepts, they also make mistakes in that area. That may be partially due to prominent concepts in the course material that students feel obligated to include in their concept map, but the trend is rather obvious. The most misleading concept for students new to programming seems to be parameter followed by attribute and method. Out of those, parameter is special, because the rate of right and wrong edges are nearly identical.

Taking a close look at the two groups of students, those who had not programmed before started out with fewer misconceptions, which indicates that students were not using “everyday” meaning of the concepts in the pre-tests (e.g. “Attribute describes object”).

The concepts object and assignment are special because they have a decrease of misconceptions in the group without any previous knowledge whereas the group with previous knowledge doesn’t follow this trend. For assignment they even show an increase in misconceptions.

5.2 Difference between knowing and doing

The results of Figure 3(a) for students without prior experience fall into three categories:

(1) The two values are high and close together. This indicates concepts that were understood well and implemented well, on average. The core concepts of OOP, attribute, method and object fall into this category.

(2) The two values are low and close together. This is the opposite of category 1. Association, overloading and
Figure 3: Comparison of the code-questions and the concept-maps in the post-test. Line 1 shows the mean-values of the concepts in our code analysis. Line 2 shows the mean-values of the same concepts in the concept maps of the post-test.

inheritance belong to this category, which are the more advanced OOP related concepts.

(3) There is a (somewhat large) gap between the two values - this includes all the concepts that are related to imperative programming as well as the more “technical” concepts like arrays and access modifiers.

The students with prior knowledge (Figure 3(b)) showed almost exactly the same distribution along the concepts, however, the gap is not as pronounced in all of the cases.

It is interesting to note, that there are concepts of object orientation in both of the first groups. This clearly shows, that several OO concepts (like inheritance) are seemingly harder to grasp than the rest, while students actually were able to understand and apply the basic concepts of object-orientation after the (very short!) course. However, this must be seen in relation to the actual programming project that the students’ were asked to do: While an experienced programmer would certainly use inheritance for the task at hand, it can also be solved using very little or no inheritance (at least explicitly as covered by our code questions). This is reflected in the fact, that the students with prior knowledge were also not good at implementing inheritance but knew “something” about it.

Next, we take a closer look at the huge difference between the representation of knowledge and the usage in the code of the 3rd group of concepts like conditional statement and loop. These concepts do not fit directly into the paradigm of object-orientation. Both concepts were used quite often in the code but not often integrated correctly in the concept-maps. As mentioned above, the majority of the concepts have their origin in the object-oriented paradigm. So it is quite difficult to find a correct association for them - which may explain the low value in the concept maps. But it may just as well show, that “understanding” those concepts is not a trivial task that takes considerably more time then learning to apply them. This is supported by the fact, that there are relatively more misconceptions for those concepts in the post-test.

This indicates the difficulty inherent in learning OOP: There are several groups of concepts that are all needed to create a “real” OOP program, but those groups of concepts are showing radically different results even for those students that had prior programming experience. There is a group that is understood and applied well, a group that is only applied well and a group that is neither understood nor applied well.

Finally, there is an interesting observation on two very similar concepts. State on the one hand and attribute on the other. As the state of an object is defined by its attributes, we would expect nearly identical values. For the implementation, this basically holds true. But, if you look on the results of the knowledge post-test you can see a big difference in the values. So here we clearly have the case
of a concept that is being used by the students but not understood. Obviously, it is not surprising, that the students don’t fully grasp the concept of state transitions of objects after this particular course, though.

6. FUTURE RESEARCH

Future studies will have to try to validate our findings about the different groups of concepts. Especially, it will be interesting to find out how and when the concepts that have low knowledge and application scores at the beginning will eventually be mastered by the students. Additionally, it seems that our results are in support of the taxonomy introduced by Fuller et al. (2) - as for some concepts the ability to apply them and the ability to interpret them were very different. Further research may be able to focus more on the taxonomy in order to get further evidence for its validity in OOP. As mentioned above, if we try to correlate “knowledge” and “abilities” on a larger scale, we must identify a systematic approach on how to measure both. We are going to focus on how to analyze source code in a more systematic way than currently possible. Most probably, the creation of the list of concepts will have to take into account how well those concepts can be represented in source code. Ideally, we would devise a method that allows us to create an objective list of concepts (from given course material) and a list of questions that can be used to test the code. Concerning the influence that the tutors have on the groups (which will always be present to some extent), we are going to analyze the protocols of questions/answers. This will give us a better idea of how much the tutors affect the learning of the students. While concept maps are a proven tool, there are still some open questions about how to use them in order to get the best possible results from students. Especially the list of concepts has a major impact on the results. Also, when asking students to draw concept maps, the introduction given to them has a major impact on the results. We provided the students with an exemplary concept map (not related to CS) that was a little different from the usual concept maps because it had rather complex edge labels (almost complete sentences). To a great extent, the students tried to replicate this kind of concept map in their own map. So it would probably be best to provide the students with several examples that show a wide array of possible concept map applications, so that they are free to choose which way best expresses their knowledge. Additionally, it seems worthwhile to think about limiting the possible edge labels to a predefined set. Doing so would greatly simplify using the edge labels as part of the analysis. As we described (14), it seems like a set of 10-15 edge labels would suffice when taking into account what the most typical labels of students have been.

7. CONCLUSION

Our results show, that we can identify different kinds of OOP-related concepts. Some are basic enough, that students can learn them on their own in a rather short time frame, others need a certain level of prior knowledge in order for students to incorporate them into their mental model. However, it is possible for students to learn about application of certain concepts without fully understanding the underlying concepts. Also, our results show that it is possible to have a rather heterogeneous group of students working on the same material and still have a learning effect on both groups. The students that already had programmed focused on certain concepts in their learning and simply created more complex programs. And the students without previous knowledge picked up basic elements of OOP while ignoring (or not fully understanding) the more advanced concepts. We found a strong difference between the concepts corresponding to object-orientation and those who deal with imperative control structures. Both aspects are important in order to fully understand OOP, however. While we observed a learning impact in both aspects, the one for non-OO concepts was lower, which may be due to the course material that focused more on the OO concepts. However, this also shows the difficulty that students and teacher face concerning teaching OOP: The concepts involved are very diverse but all are important for programming.

8. REFERENCES

Chattanooga and TN and USA, 2009.


