On the understanding of Object and Class

Anna Eckerdal

December 21, 2004

Abstract

This paper presents the results from a phenomenographic study of first year university students' understandings of the concepts *object* and *class*. The students had just finished their first programming course using Java as the programming language. The analyses of the study display qualitatively different ways to understand the concepts. These aspects of the understandings are formulated as *categories of description*. Each aspect is important and relevant when solving a programming task, and a good understanding includes all aspects of the concepts. The categories show understandings of different complexity. There are students who can only express an understanding of the concepts as code and syntax rules. Other students can also express the role the objects play for the programmer and the performance of the task given, and classes as abstract data types. The students who express the richest understanding includes the understandings already mentioned, but they can also express that classes and objects depicts the real world.

Learning to understand a phenomenon means, in a phenomenographic perspective, to *discern* new aspects of that phenomenon. This discernment is only possible if there is a *variation* in a dimension that corresponds to the aspect, critical for the specific understanding. The question of variation gives interesting implications for teaching discussed in the paper. The discernment requires the students to have a mindful kind of learning, *reflective learning*. This implies for the teachers that *explicitness* in the teaching is of great importance. Explicitness in the teaching is defined at several levels, not only that the different aspects of the understandings found in the study should explicitly be mentioned, but also explicitness in the explanation and variation of these aspects.

The importance of offering a broad context for object oriented programming is also emphasized. This can include explaining not only the object oriented paradigm, but also to discuss some other programming paradigms, and the historical background that lead to these paradigms. It can also include giving the students the opportunity to follow a whole programming task with analysis of the problem, and to design, implement and test the program even at an early stage of their education.

The paper also includes some examples how the results from the study can be implemented in the teaching to help the students in their learning process.

Contents

1 Introduction

Object oriented programming languages are used at university courses at all levels throughout the world. Much has been reported on the experiences on teaching the object oriented paradigm. Object oriented programming is experienced as complex and significant detailed by many institutions (Roberts, 2004), and difficult to teach (Kölling, 1999). One important aspect of learning the object oriented paradigm is that it is built on some fundamental abstract concepts. Many studies illuminate the necessity of good understanding of those, where some of the concepts are necessary for students to learn even at an early stage of the programming education. Holland, Griffiths and Woodman claim that misconceptions of object concepts can be hard to shift later. Such misconceptions can act as barriers through which later all teaching on the subject may be inadvertently filtered and distorted (Holland et al., 1997). Fleury found in a study (Fleury, 2000) that students constructed their own understanding of concepts when they worked with programming assignments, and those constructions were not always complete and correct. "Because students construct their own meanings during instruction, it is not surprising that students possess only partial conceptions even when provided with complete and accurate information." says Fleury. Holmboe discusses how to reach good understanding in programming: "To reach understanding based on theoretical definitions, will mean trying to understand the formal aspects without a frame of reference due to lack of personal experience." And later in the article: "Both practical skills and conceptual understanding are necessary, and interconnection between these two preferable." (Holmboe, 1999). Box and Whitelaw argue from a constructivist learning theory, that more abstract types of learning are required by the student for object-oriented technology than for structured technology (Box and Whitelaw, 2000).

The references mentioned point to the importance of good understanding of abstract concepts when learning object oriented programming. This is the background to the research question posed in this paper: *How do students understand abstract concepts in object-oriented programming?* In particular the different understandings of the concepts *object* and *class* are in focus. The objective is to identify qualitatively different ways of understanding the concepts within a group of students. The group selected consists of undergraduate students aiming to a Master degree. They were interviewed in connection to their first programming course in their first year of study. The results show qualitatively different ways in which the concepts *object* and *class* have been understood within the group. Because of the acceptance of the object-oriented paradigm in university educations, the results from this study can have a positive impact of education in the object oriented programming.

The study has a phenomenographic approach, which gives the the framework for the study and the analyses of the results. This gives a sound base for analyses and discussions evolving from the study. The study focuses on the students' perspectives and conceptions, not on misconceptions. It does not take the researcher's perspective as the point of departure, but endeavors to

adopt the student's perspective on learning. Marton and Svensson (Marton and Svensson, 1979, p. 472) claim that in this perspective, the world as the student experiences it, becomes visible.

> [The student] experience of the world is a relation between him and his world. Instead of two independent descriptions (of the student on one hand and of his world on the other) and an assumed relationship between the two, we have one description which is of a relational character.

In the following, research results are described and presented as different categories of understanding found in the group of interest. The categories are taken as a starting point for a discussion of what can be the educationally critical difficulties (Marton and Booth, 1997) when learning these central concepts. Finally implications for teaching are discussed.

2 The Object-Oriented Paradigm

2.1 Background

Programming is a central subject in computer science education. The programming paradigm taught has however changed over time. Currently the object oriented programming paradigm is common. Programming languages within the object oriented paradigm are e.g. C++ and Java. This section is an attempt to briefly explain the object-oriented paradigm and the concepts *object* and *class* to readers not previously familiar with them.

The principal aim of software engineering is to produce programs with high quality. With high quality we mean programs that are correct, efficient, reusable, extendible, easy to use and so on. Some of these quality factors have been important reasons for the development of the object-oriented paradigm (Meyer, 1988), (Hamilton and Pooch, 1995).

Reusability is the ability of software products to be reused, in whole or in part, for new applications. When programs that have, in whole or in part, been thoroughly tested, further development will be faster and cheaper since these parts can be safely reused.

Extendibility is the ease with which software products may be adapted to changes of specifications. An important way to achieving this is to create independent parts of code, modules, with as little communication with the rest of the program as possible, and minimal interface. The more independent modules, the more likely that a simple change will just affect one or few modules, rather than start a chain reaction of changes over the whole system.

Program cost involves the cost for developing the programs, but also the cost for maintaining programs. This includes correcting errors in the code and changes in the programs when the circumstances change, e.g. change of specification. The using of well-tested modules (reuse of code) and code that can easily be changed (extendibility) reduces maintenance cost.

Describing the thoughts behind the object oriented paradigm Meyer writes: "A software system is a set of mechanisms for performing certain actions on

certain data. When laying out the architecture of a system, the software designer is confronted with a fundamental choice: should the structure be based on the actions or on the data?" (Meyer, 1988, p. 41). The latter choice is one of the main principles behind the object oriented paradigm. Meyer has the following definition of object-oriented design: "Object-oriented design is the method which leads to software architectures based on the objects every system or subsystem manipulates (rather than "the" function it is meant to ensure)." (Meyer, 1988, p 50). Arguments for choosing data instead of functions as a base for the program are:

- Data structures are more stable over time compared to the function of a program.
- It is easier to adapt new demands to a software system built on its data types than on a system built on its actions.
- A software system is easier to reuse when it is built on its data types compared to a system built on its actions.

2.2 Central concepts: class and object

There exists an established understanding among professional programmers and teachers how to understand the concepts of *object* and *class*. There might be some differences, but the main ideas are probably shared among most professionals (Meyer, 1988), (Bar-David, 1993). When describing the concepts *object* and *class* a variety of aspects should be mentioned:

- An object is the computer representation of some concrete phenomenon in reality.
- A class is the description of the general characteristics of the phenomenon. It is used as a template when objects are created and includes algorithms describing how the objects can be manipulated.
- A class represents a particular abstract data type implementation. An abstract data type is a set of values, a data structure, together with a set of operations, member functions, which access and modify those values.
- Objects are instances of the abstract data type. An object is a container of values of this data type. The state of an object is its current value. An object is manipulated by the member functions defined in the class.
- A class is a text file permanently saved in the memory. It is a static description of a set of possible objects - the instances of the class.
- Objects are created when the program is executed. They only exist during runtime, an object is a dynamic concept.
- A program or system is a collection of objects that get the work done by sending each other messages.

- A class can be understood as a module of the program. When designing a program, classes are used as the bricks, the modules the program is built of.

The two aspects of a class as an abstract data type and as a module are important when understanding the concepts class and object. They are useful in different situations, describing the concepts from different perspectives. When designing a program, the identification of classes is a support for the modularisation. As mentioned earlier, using objects as the key to system modularisation is based on quality aims. This is one of the major factors that contributed to the success of the object-oriented paradigm.

3 The study

3.1 Scope

The subjects chosen for the present study are students from a study programme where programming knowledge is not a major goal. Programming courses are compulsory in most technical and science university study programmes in Sweden, not only in programmes within the computer science area. The group selected is thus representative for a large number of students studying programming.

The present study focuses on students' understanding of some central concepts within the object-oriented paradigm, *object* and *class*. When studying Java, students are often confronted with these concepts at an early stage of their study. The understanding of the concepts is thus important even in a beginners' programming course. There are other concepts central within the object oriented programming like *encapsulation* or data hiding, *inheritance* and *polymorphism*. The reasons for the present study to focus on the concepts *class* and *object* are mainly the course content. In the present first programming course, other concepts are mentioned, but not thoroughly reviewed. These concepts mostly belong to a second programming course, offered to the students later in their education.

Aiming at understanding more about how students understand, or experience the concepts *object* and *class*, a phenomenographic research approach has been chosen. Phenomenography aims at describing the variation of understandings of a certain phenomenon found in a group of people. Marton and Booth discuss the idea of phenomenography:

> The unit of phenomenographic research is *a way of experiencing something,* (. . .), and the object of the research is the *variation* in ways of experiencing phenomena. At the root of phenomenography lies an interest in describing the phenomena in the world as other see them, and in revealing and describing the variation therein, especially in an educational context (. . .). This implies an interest in the variation and change in capabilities for experiencing the world, or rather in capabilities for experiencing particular phenomena in the world in certain ways. These capabil-

> ities can, as a rule, be hierarchically ordered. Some capabilities can, from a point of view adopted in each case, be seen as more advanced, more complex, or more powerful than other capabilities. Differences between them are educationally critical differences, and changes between them we consider to be the most important kind of learning. (Marton and Booth, 1997, p. 111)

And later:

> (. . .) the *variation* in ways people experience phenomena in their world is a prime interest for phenomenographic studies, and phenomenographers aim to describe that variation. They seek the totality of ways in which people experience, or are capable of experiencing, the object of interest and interpret it in terms of distinctly different categories that capture the essence of the variation, a set of categories of description (. . .) (Marton and Booth, 1997, p. 121-122)

The object of interest in a phenomenographic study is thus how a certain phenomenon is experienced by a certain group of people, and the *variation* in the way the phenomenon is experienced (Marton and Booth, 1997, p. 110). A fundamental assumption in phenomenography is that there exist only a limited number of ways in which a certain phenomenon can be understood. The understandings of a certain phenomenon can be described in hierarchically ordered qualitatively different *categories of description* which form *the outcome space* of the phenomenographic analysis.

Phenomenography is an empirical research approach. Data are often, like in the present study, gathered in the form of interviews. The data are analysed and the results, the different understandings found in the data, presented in the outcome space. The understandings are found when the data are read and reread and patterns of distinctly different understandings are looked for. Individual, decontexualised quotes illustrating certain understandings are compared with each other to ensure that all qualitatively different understandings found in the data are expressed in the outcome space. The quotes are also read and reread in their own context to avoid misunderstandings. The researcher formulates the essence of the understandings found with his or her own words in the categories of description. Suggested categories are often discussed within a group of researchers, comparing the data again, both single quotes and quotes in their context. In this retentive analysis, by again and again going back to the data, and in the discussion with other researchers reading the same data, the categories of description are finally agreed upon among the researchers.

The analysis is done at a group level, not aiming at putting individuals in certain categories. An individual can hold several of the understandings expressed in the categories of description, but mapping between individuals and categories is not the aim of the analysis. It is unlikely that the collected data can reveal all the different ways in which each individual student understands the concepts of interest. However, when statements from different students are brought together, that collective "pool of meaning" reveals a rich variety in understandings. When quotes are taken out of their contexts and compared

with each other, the individuals are put in the background, and the collective understandings of the group are in the foreground. The focus on the group level is important in a phenomenographic study.

3.2 Data collection

The interviews in the present study took place at Uppsala University, Sweden in May 2002. A questionnaire was given to a group of 45 undergraduate *engineer* students. 22 of the students answering the questionnaire were willing to participate in a one-hour tape-recorded interview. 14 students were selected with the intention to get as many male as female students, and to select students with different programming knowledge background. The students participated voluntarily in the study, but got one movie ticket each as a sign of recognition.

3.3 The course

The students selected for the study had just finished their first computer-programming course, a compulsory course giving 4 credit points. (At Swedish universities one credit point represents one week's full-time study and 40 credit points one full academic year.) The programming language used in the course was Java. The study programme the students attend is called Aquatic and Environmental Engineering. It is a 4.5 years graduate engineer education, demanding good previous knowledge in mathematics, physics, chemistry and biology. The study programme has an emphasis on environmental issues, and the students are likely to get high-qualified jobs after the education.

When choosing programming course for the present study, one criterion was to select a course where the authors were not teachers themselves. In this decision, we followed the recommended rules of ethics, established by the Swedish Research Council and used at universities throughout Sweden (http://www.vr.se, 2003).

3.4 The interviews

The interviews were semi-structured (Kvale, 1997, p. 117). Kvale describes a semi-structured interview as a human interplay. This interplay is not as anonymous and neutral as when a person answers a questionnaire, nor is it as personal and emotional as in a therapeutic interview. If necessary regarding the answers from the students, it is possible in a semi-structured interview to dynamically change the form and order of the questions, in response to the answers given by the students. The present interview had two themes: students' use of resources when learning Java, which we will return to in a later report, and their understanding of the concepts *object* and *class*. The interviewer had prepared a small number of questions on each theme, intended to approach the themes from different perspectives. Follow-up questions during the interviews, in addition to the prepared questions, were also given to clarify students' statements.

The aim was to encourage the students to demonstrate as much as possible of their understandings and experiences within the themes.

Some examples from the interviews are presented below to illustrate the questions asked and to show how different understandings can be expressed when new questions are asked. In the quotes, the interviewer is labeled I, and the students A,B,C etc. All students were asked a few questions on each concept. Depending on their answers, many or few questions followed. Sometimes a statement from the student needed to be clarified. Sometimes the student had problems to verbalise his or her understanding, and more questions were needed.

The following interview excerpt illustrates that student G can express many different ways to understand the concept *object* with only a few questions from the interviewer.

> I: I would like you to tell me how you think about what an object is.
> G: An object I see as a thing in a way that has different characteristics. Eh... it is something that you can touch, it feels so to speak, that is something that contains different information of how it behaves, that thing. That is the simplest explanation I think. That I see as an object.
> I: Yes.
> G: That is to say objects in programming of course. (laughter)
> I: That's right (laughter). That is the question.
> G: But it is really something that... it's an object you can touch, this is how it works and in the object there are characteristics of how it behaves.
> I: Yes. Characteristics, yes. Okay. You can draw or write or something that you think associates with object.
> G: (draws)... so... it's an object...
> I: Mm.
> G: Somewhere in the memory so to speak Java saves, or like a space, that's what an object looks like. Then in there you can so to speak put in things where you want to put in it in some way.

Student H needs many questions before he/she can find a way to formulate an understanding of the concept *object*. Although the interviewer asks many different questions to find what aspects of the concepts the student has grasped, the student can only express a few aspects of the concept.

> I: [...] then I want you to tell me, write an example, draw, talk about how you think about what an object is.
> H: Well, it is that which is a little like that.
> I: It may willingly be like that, it doesn't matter.
> H: Because I don't know really what it means with an object. I haven't really any idea of it. An object for me is very fuzzy.
> I: That's okay. (laughter)
> H: (laughter) So I don't know really, it's so to speak difficult and (laughter)... well...
> I: So you might not have an image so to speak...
> H: No I don't really have that.
> I: Any example.
> H: (giggle) Any example, well... an object, I really don't know what to...

I: What would you say then to a friend who doesn't know anything about programming and who asks, what does it mean with object oriented programming.
H: ... well (giggle) I would be rather, yes, would..., no, actually I don't know really. Well, it is programming that... well the difficulty is that I don't really know it either what's the difference of object orientation and so to speak what it is otherwise.
I: Well. That you couldn't know since you haven't programmed before.
H: No I have never programmed non-object... that much I know about programming.
I: Precisely. It exists but that I can't ask you about since...
H: No, I don't know, unfortunately.
I: You don't have an example from the exam of something you would do or from some assignment or lecture that got stuck, a special case...
H: No, yes... no nothing that I can...
...
I: Yes. How do you think of so to speak what an object and a class are.
H: Well... yes... it's a difficult question.
I: What do they have to do with each other so to speak.
H: Yes the class is so to speak, in some way it is, if you now will write some program then it is practically so to speak built up of classes, it is kind of that you... so an object is, well in that case a part of a class, or something like that. If you can say it like that, perhaps you can't. It is like a sublevel maybe. A class it feels like it is the top if you think in levels. First it is so to speak classes and then methods and they also lie underneath the classes.

The example below shows that some students, like student J, has several understandings of the concept *object*, but the interviewer needs to draw attention to the various aspects of the concept to make the student express his/her different understandings:

I: [..] what do you think about what an object is.
J: What an object is. When I have discussed and so on then I have said that an object is a class or can be a class, or can be a method in a class. Eh, when I have been studying I have tried to think of if you have a programme, I don't remember but... which consists of a couple of objects and the objects can be different classes which contain methods. In that way I tried to get a picture of what an object is, because it is easier for me to think that if I have a class and in the class we have a lot of objects.
I: Although now you drew it the other way around...
J: Yes because it was like that I thought about it when I read the book, so I don't know really which is right. It's different when you ask different persons, yeah but, class is that an object. Yes it is an object but a method was also an object. Is it the class which contains different objects or is it an object that contains different classes. So that I don't know.

I: Precisely.
J: But you get a little bit more grips that it is so to speak... (long pause)... if you think of the Java programme, that it will be built up with different

objects and that it is the object which we modify in order to get what we want out of it, something like that I try to think of instead of trying to divide it into different classes and so on.
I: Okay, okay. You think of it a little from the user side like that.
J: Mm.
...
I: Okay. Then you talked about methods too.
J: Mm. Eh... in the methods so... uses to write it the way we want them to do, like will happen so to speak. What will happen in the programme, what you will do or like that.
I: How is it connected with the object then.
J: (Sigh, giggle) Well, that is the question, it depends on in case the method itself is an object or... if object is something else but, eh, method, well, you can use different methods, in a method you can call another method and get them to cooperate in that way. So then, or call another class or.
I: And the objects.
J: I don't know if, because I feel so to speak that I don't really understand what object is then it feels more like that methods are parts in the object.
...
I: Eh, what do you think is the point of having objects and classes?
J: Well, that you can ask. (laughter) I think the point of having objects and classes is that it will be easier to think of what it is you shall do and what the programme shall contain so that you will try to get some reality picture of it but I don't manage with that so...

The three examples from the interviews above show that in a semi-structured interview the aim is to help the student to talk as freely and as much as possible, avoiding leading questions, but to stick to the subject of the interview, the understandings of the the concepts *object* and *class*. A variation in understandings of the concepts *object* and *class* is found in the group. An individual student can express a certain way to understand the concepts in the beginning of the interview, but when attention is lead towards another aspect of the concept, when new questions are asked, the student might express other types of understandings too, see the excerpt from the interview with student J above. There might also be different needs to encourage the students to talk and express his/her understandings, compare the excerpts from the interviews with student H and student G above. The researcher's goal is to help the student to articulate his or her different ways to understand the phenomena of interest. However, there is no claim that the interview reveals *all* the different ways in which the individual student understands these phenomena, which is also not possible to certainly know. Consequently, the analysis has its focus on the variations of the understanding within the *group*, not for individuals.

4 Phenomenographic analysis

The fourteen interviews in the study were tape-recorded and transcribed verbatim to text files. The files were read and analysed. All statements concerning

the concept *object* were copied to a separate file to form a pool of statements on the phenomenon *object*. The same procedure was made for the concept *class*. The decontexualised statements were read and compared with each other. By reading the statements several times, looking for variation in the understandings of the concept, a limited number of categories describing the different understandings found in the group appeared. In this way the interviews were analysed at a group level. The individual understandings were not in focus, the ambition was to describe the variation of the understandings found in the group. The statements were also read in their context to avoid misunderstandings of any statements.

When looking for categories of description found in the group, two researchers independently read the interviews and formed their opinion of the categories of description appearing in the interviews. The results were very similar. One person had found three categories in the interviews, the other had found four, including the first person's categories. Going back to the interviews we agreed upon the number of categories and how to describe the categories found.

4.1 The concept of "object"

The different comprehensions of the concept *object* found in this study, can be formulated in three categories of description presented in Table 1. The three categories are illustrated by quotes below.

Object is experienced as a piece of code.
Object is experienced as something that is active in the program.
Object is experienced as a model of some real world phenomenon.

Table 1: Categories describing the different ways to understand the phenomenon *object* found in the group.

In the first category, the comprehension of the concept is limited to an analysis of the structure of the code. Student C says:

> I imagine that it is a piece of the code with all the variables piled under.

When the interviewer asks the student how he/she would explain to a friend who does not know anything about programming what an object is, student N answers:

> I'd just say that it is a part of the program.

In the second category the comprehension is extended to include the results of the program execution, and the task of the object. It can be illustrated by the following answers.

Student B explains what an object is:

> ... what you create and want to use in the program [...] an object that you want to work with.

Student H says:

> the object is a kind of, what is doing something [..] because it is all about that something is going to happen.

Student J says:

> If you think of the Java program, that it is built of different objects and it is the objects we modify so that we can get what we want from it.

The third category describes an understanding that an object is a model of some real world phenomenon. This is expressed in the following quotes:

> C: Yes an object, you can have a rather physical image of it....
> I: What did you say, physical?
> C: Kind of, you can think of a car and then it has one variable for how many wheels it has, one variable for the size of the engine like that.

The three categories express an increasing understanding and complexity. The first category shows an understanding that all students express in one way or the other, objects as they appear in the code. A few students express only this understanding. The second category expresses the importance of the objects for the results of the program execution, the active task the object has. The last category describes the relation between the objects and the real world. The first category expresses a poor understanding, while the last one shows a rich understanding including fundamental thoughts behind the object-oriented paradigm. In an unpublished pilot study, Eckerdal (2002) has found similar indications. Students with the least understanding of the phenomenon *encapsulation* only understand it at a code level, while a richer understanding requires that programming is seen in a context that goes beyond the code and the syntax of the programming language. Holmboe writes about understandings which includes the world outside the computer itself: "A person with holistic knowledge relates the implementation and design of a computer program to the real world being simulated." (Holmboe, 1999). It is thus of great importance that students reach an understanding of the concepts *object* and *class* that goes beyond the code level.

4.2 The concept of "class"

When looking for the different understandings of the concept *class* expressed in the study, a pattern similar to the understanding of the concept *object* is found. There are comprehensions focusing on the code and the task of the programmer, but there are also comprehensions where the reality the program is supposed to model is present. The categories of description are presented in Table 2 and illustrated by quotes below.

Many of the students express an understanding belonging to the first category, *"Class is understood as an entity in the program, contributing to the structure of the code"*. Student H says:

Class is experienced as an entity in the program, contributing to the structure of the code.
Class is experienced as a description of properties and behaviour of the object.
Class is experienced as a description of properties and behaviour of the object, as a model of some real world phenomenon.

Table 2: Categories describing the different ways to understand the phenomenon *class* found in the group.

> A class is well, yes, like I think a class is like a little programme, that's how I think of it, a small programme inside the whole big programme being kind of the main programme. Then a class is like a small programme which does certain things.

The understanding has its focus on the program structure and the programmers task and describes the class-concept as a help for the programmer when structuring the code. It deals with the code and the programming task, and the description of the class reminds of a description of modules, even if no student explicitly uses this formulation. Some students emphasize this module aspect:

> C: Then the class should really be something "clever" which contains that you shield, this is a class and in a class you could put everything under the same class although it's not very clever to do so if you want to use some things in other programmes. Then it is good to have them kind of shielded from each other. But the class is really just some blurred collection of, this I think belongs together, in some way.

> E: Class. Mm, it took a while before you came to grips with classes, what it really was actually, that I don't know if I still have. But classes which only contain a lot of methods for instance that you later use or, like how a vector works, it's a class for instance and, a bunch of computing vectors which you then will be able to call and use so that you don't have to write everything at the same place, a sort of classification of chapters or something similar.
> I: Classification of... ?
> E: But well, you divide the programme simply and then... theoretically you can always write everything in the same programme, or? Although it would be so incredibly... I don't know, it wouldn't work.

The second category, *"Class is understood as a description of properties and behaviour of the object"* is the most common understanding expressed in the group. Even if none of the students explicitly uses the expression "abstract data type", the descriptions point in this direction. All students mention the methods, that is the behaviour of the objects, when talking about the classes. A few students do not mention that the object's properties are defined in the class. Most of the students focus on the behaviour of the objects in the program during execution, described in the class. Some examples from this category are given in the following:

Student L emphasises the behaviour of the object:

> It was then when we were making our own classes. At that point I came up with that it was just kind of a storage space for methods that belong to certain objects.

Student O and M articulate that a class contains a description of both properties and behaviour of the object. Student M also points to the variation of the objects properties when objects of the same type are created, and variation of properties using the class functions:

> O: [...] Eh, when you write a class, for instance class vector which we have had as a class particle, then you write well, yes, to be able to create an object of that class later you write how you want it to look like and that is how I see a class, that you will be able to create an object and some of what you will be able to do with this object in the different methods [...]

> M: How I think about a class... well, like a ginger cookie cutter maybe. [...] It is more like a cast form then cause how you make new ones from the beginning identical one maybe, if you now will think about that. But which can be different very quickly. It can be... well, it is...
> I: They can be different, what did you say?
> M: They have the same origin in some way but they don't need to be identical because they come out from the same.
> ...
> I: What is it that you think so to speak is different?
> M: Different contents then, more like instance variables. They... well, you can do the same operations with them, you can do, with the same class you can in principle do the same elements and so on with all objects. If they are not too...
> I: But element that is to say.
> M: You can, if you have your cat object you can let the cat run even though it has three legs defined or two legs.
> I: Yes, that's right.
> M: You can still let it do certain stuffs anyway.
> I: What do you think is then... the difference so to speak of class and object?
> M: Class and object. Yes, that the class is the pattern over how the objects of the class look like. That's how I think of it.

When describing a class as an abstract data type, many interesting metaphors are used. In the quotes above student M uses "ginger cookie cutter", "cast form" and "pattern". Student A uses "pattern" and "mathematical formula":

> A: Oh, the point is that you have a pattern from the beginning, then you can make them green and blue if you want that and if you don't want it then you take them away and make them orange instead for instance. It is, it is about like a mathematical formula before you have put in the numbers. With this formula you can do plenty of different things and you can perhaps change an m and take it away and, well, you can do very much without affecting itself... if I want something to be 18, then I put in the numbers that makes it 18. It's the same here. If I so to speak want something to look exactly so.

Student L uses the expressions "box" and "storage space":

> L: (Hm) I would probably almost describe it as a box where you put different characteristics, different things that this object will do.
> [..]
> L: It was then when we were supposed to do our own classes. Then I realize that it was so to speak just a storage space for methods that belong to certain objects.

In the third category in Table 2, *"Class is experienced as a description of properties and behaviour of the object, as a model of some real world phenomenon"*, the close relationship between the class definition and the reality the class depicts is pronounced. This category explicitly includes the understanding expressed in category two. Only a few students express category three. Student C says:

> I: But this about class, I mentioned, how do you think about class?
> C: That is a bit more diffuse actually. Class, it is that I would probably think of that a class contains, can contain a couple of objects or just one object and different operations that you can do in an object or between objects. So that you can also think of what it would mean in reality.
> I: Okay.
> C: Well, you can have a working space and a human that works there, then you have two objects and then they can so to speak interact with one another through different operations so to speak, what do I know, the human gets some coffee and then the coffee variable goes down at the working place and so on.
> I: Okay (laughter). And what do you think the class now, now I want to...
> C: Then the class would really be something smart containing what is to be shielded, so this is a class and a class you could put everything under the same class although it is not very clever to do if you want to use some things in other programmes. Then it is good to have them so to speak shielded from each other. But the class is just some blurred collection of, this I think belongs together, in some way.

Notice that in this excerpt student C expresses understandings belonging to all three categories in Table 2.

4.3 The purpose of using objects and classes

One of the questions to the students was "What do you think is the point of using classes and objects?" Most of the students in the study had never programmed before. A few of them had tried other programming languages like C++, Pascal or Basic. Although most of them had never tried a non-object oriented programming language, still all had an idea of what is the point of using classes and objects.

The students' different understandings of the point of using classes and objects, are presented in three categories in Table 3.

Most of the students expressed an understanding of the purpose belonging to the first category. Most of them also expressed an understanding belonging to

The purpose is understood from a code perspective: the syntax requires it, and it gives the program a good structure.
The purpose is understood from a user and result perspective: simpler for the programmer to make the program solve the task given.
The purpose is understood from a reality perspective: support to connect reality and programming.

Table 3: Categories describing the different understandings of the purpose of using the concepts *object* and *class*, found in the group.

the second category. Only a few students expressed an understanding belonging to the third category.

The first category expresses the understanding of the purpose at a code level. The reason for using objects and classes is because the programming language requires it and that objects and classes give a good structure to the code because of the modularisation the classes achieve. The purpose is built in the construction of the language, in the syntax rules.

> I: ...what do you think is the point with having classes and objects? Why do you create classes and objects?
> N: It is because the programmes require it. That's the way it goes to create a programme. I don't know how to do it otherwise.

Student E says:

> E: But well, you divide the programme simply and then... theoretically you can always write everything in the same programme, or? Although it would be so incredibly... I don't know, it wouldn't work. (giggle)
> I: Why do you think it wouldn't work?
> E: You have to have everything in order, the structure, in the beginning you don't think so much about the structure but when you start programming some more then you realize how important it is [...]

Student G, having previous experience in the non-object oriented programming language Basic, says:

> G: [...] I remember so to speak that it was... if you will compare Basic with Java then it was so that large programmes in Basic became very unstructured, it's difficult to find how you will put it forward and structure it... run so to speak with classes, divide it up in different files so to speak and this whole part makes it so it can be built upon each other in some way, build so to speak programmes on other.

In the second category of Table 3, *"The purpose is understood from a user and result perspective: simpler for the programmer to make the program solve*

the task given", the understanding is not focused on the program code and compiler but on the programmer and the task the of the program. The most usual way to express this is to say that objects and classes simplify the work for the programmer e.g. when debugging and in reuse of code, and by using objects and classes it is more easy to get what you want from the program. Student G above, and also student H express an understanding that includes both the first and the second category:

> I: Mm. What do you think the point is having objects and classes?
> H: [...] one point is this that you can use, if you write it in some place so to speak...right this that you can call them and use them even in other places so that it will be a kind of...yes... [...] Right this that you were not allowed to describe, that otherwise perhaps would have been the alternative, that you only could have been writing what you need and then you had to write it so many times but now you can only write one class and then you can use it anywhere you want and so on. It is very time saving. That's what it's all about to write...
> [...]
> I: [...] Why do you create classes?
> H: Yes why... well it is... yes it's the same reason as the latter question, a little bit right this that it is like this that it's simply built up. It's like this you, well the whole programme is done a little like this so you will do so but... yes why, otherwise you wouldn't be able to do anything if you didn't have any classes. That's how I feel about it. I don't know really what it would consist of otherwise really.

Student K expresses the users perspective:

> I: What do you think is the point of having objects and classes then?
> K: The point is that... (pause)... well it is that you can write programmes easy for that purpose which you are looking for kind of.

And later K says:

> K: But simply that it's easier with classes to get what you want from the programme or to write...
> I: To write the programme?
> K: Yes or to solve the assignment perhaps you can say. To write the programme can be too hard even with classes but, yes. To solve the assignment. If you take for instance assignment no. four, if you didn't have classes in it then it would have been really hard for sure.

The third category of Table 3, *"The purpose is understood from a reality perspective: support to connect reality and programming"*, shows the most abstract understanding of the purpose of using objects and classes. Classes and objects reflect the reality that is going to be matched in the work of the computer program. Student C explains this clearly:

> I: What do you think is the point with having classes and objects?
> C: Well, the point is really that you can have this clear image, this is how it looks in the real world. Yes but then it is something similar on the

computer then. So that it can be described so to speak, therefore a rather clear image of it. Now I have a particle and I have a box and the particle has vectors. Yes it has it in the reality too in some way.
I: Precisely.
C: So you get a very concrete picture of also how you perhaps can put forward the programme if you will do something which in reality has a couple of objects, then they create one object at the time so to speak and then you link them together so they will behave as you want. So it is a very clear picture...

The categories are arranged hierarchically, from a concrete way to understand to the most abstract in the third category. The first category, *"The purpose is understood from a code perspective: the syntax requires it, and it gives the program a good structure"* is comparable with the first category describing how students understand the concept *object* in Table 1 *"Object is experienced as a piece of code"* and the first category in Table 2 *"Class is experienced as an entity in the program"*. The third categories in the Tables 1, 2 and 3 all discuss the reality aspect. Even the second category in Table 3: *"The purpose is understood from a user and result perspective: simpler for the programmer to make the program solve the task given"* is comparable with the second category in Table 1: *"Object is experienced as something that is active in the program"* and the second category in Table 2 where the behaviour of the object is described: *"Class is experienced as a description of properties and behaviour of the object."*. They all focus on the activity of the program , the task it solves.

The understanding expressed in the first part of the first category in Table 3, *"The purpose is understood from a code perspective: the syntax requires it"*, is hardly a professional way to understand the point of using *object* and *class*. The use of objects and classes has come from a need programmers experienced and thus object–oriented programs developed. Meyer writes: "The case for using data (objects) as the key to system modularisation is based on some of the quality aims [...]: compatibility, reusability, extendibility." (Meyer, 1988, p. 49). Object oriented programming has not developed because of a compiler, the compiler is developed because the users needed a tool to be able to program according to this paradigm.

4.4 Summary of the analysis

The concepts *object* and *class* in object oriented programming are closely related to each other, and can hardly be understood without each other. As described in Section 2 an object is an instance of a class, and a class is the description of a phenomenon in the reality. The class is used as a template when objects are created and includes algorithms describing how the objects can be manipulated. When describing the different understandings found in the group, we were not surprised to find similar patterns for the understandings of the concept *object* and the concept *class*. In the empirical data collected for the present study, most students express understandings of the concepts in corresponding categories. If a student for example expresses an understanding of *object* corresponding to the

second category in Table 1, he or she also expresses an understanding of *class* corresponding to the second category in Table 2. There are few, if any examples where students show an advanced understanding of one concept, and a poor understanding of the other concept. As a consequence of these observations, the categories of understanding described in Table 1 and Table 2, respectively, are merged in Table 4

The understandings are merged in the following categories of description in Table 4, including both the *object* and the *class* concepts. When comparing the outcome space in Table 1 and in Table 2 with the professional way to understand *objects* and *classes* described in Section 2, we also notice that the main ideas behind the concepts are covered in the outcome spaces.

Class is experienced as an entity of the program, contributing to the structure of the code and describing the object, where the object is understood as a piece of program text.
Class is experienced as a description of properties and behaviour of objects, where object is understood as something that is active during execution of the program.
Class is experienced as a description of properties and behaviour of objects, where object is understood as a model of some real world phenomenon.

Table 4: Categories describing the different ways to understand the phenomena *object* and *class* found in the group. The latter categories include the understandings in the former.

In Table 4, like in the previous tables, the categories are intended to be inclusive. This means that an understanding expressed in one of the latter categories includes the understandings expressed in the former categories. It is hardly possible to understand that an object is a model of something in the reality without understanding that this implies a description of its properties and functions, expressed in the code. This inclusive character of the categories is described by (Marton and Booth, 1997, p. 107) in terms of a logic relation between the categories which is often hierarchic. The authors explain:

> [...]the limited number of qualitatively different ways in which something is experienced can be understood in terms of which constituent parts or aspects are discerned and appear simultaneously in people's awareness. A particular way of experiencing something reflects a simultaneous awareness of particular aspects of the phenomenon. Another way of experiencing it reflects a simultaneous awareness of other aspects or more aspects or fewer aspects of the same phenomenon. More advanced ways of experiencing something are, according to this line of reasoning, more complex and more inclusive (or more specific) than less advanced ways of experiencing the same thing, 'more inclusive' and 'more specific' both implying more simultaneously experienced aspects constituting constraints on how the phenomenon is seen.

Since the tables above are inclusive, an understanding corresponding to one

of the latter categories expresses a richer understanding of the concepts *object* and *class* compared to an understanding corresponding to one of the former categories. In this context it is of interest to mention Pong's article "The Dynamics of Awareness" (1999), where he discusses learning in terms of "multi conceptions". He gives a short survey on how the understanding of learning in the 1960's was defined as conceptual change, whereas today some researchers advocate what is known as the "multiple conceptions" perspective, which can be interpreted as increasing the number of ideas about the physical and cultural world. Another reference of relevance for the discussion about inclusiveness is Berglund (2002). Students do, depending on the context, express different understandings of a phenomenon. Berglund claims that in Computer Science a good understanding of a phenomenon is reflected in the capacity to choose in a situationally relevant way between different ways of experiencing the phenomenon. A person with an understanding belonging to one of the less advanced categories is not able to do so, because he or she can not see the aspects of the phenomenon expressed in the more advanced categories. All categories in Table 4 are thus important and valid in different situations when working with a programming task. By the programming task we mean the whole problem solution process, that is the analysis of the problem, the design, implementation and testing of the program. An understanding of *objects* and *classes* as a model of the reality is important when working with the analysis of the problem, and the overall design of the program, when finding the different roles of the objects and how they interact with each other. This corresponds to the third category in Table 4. When focusing on the task of the program, transferred to the objects' behaviour and interaction with each other, the understanding that the classes are descriptions of properties and behaviour of the objects is important. This is described in category two. To know the syntax-rules and how to write code is a fundamental skill in all programming work, and when implementing and testing a program the understanding that the classes are entities that give a structure to the program is valid. This corresponds to the first category described above. This is all in line with Berglund's arguments stressing the importance of the ability to shift between the different categories (Berglund, 2002), and Pong's discussion about the "multi conceptions" perspective, that different aspects of a certain phenomenon can be focused on depending on the context.

5 Enhancing the learning process

5.1 A framework to find implications for education

A teacher's reflection when studying the outcome space is that according to my experience many students seem to follow the pattern showed in the outcome space in Table 4 in their learning process. This means that students begin to experience objects and classes as text, that have a structure, before they understand that objects are the active entities when the program performs a task, and before they become aware of the mapping from a real problem to a

computer and understand what might be relevant objects to be described in classes with attributes and methods. My experience is however from programming courses where programming is approached via the language and syntax, which is the case in the present study. Other approaches may show other patterns of development in the understanding. In this discussion it is important to stress that the analysis of the understandings is at a group level, see Section 3.4, which implies that the development of understanding described above does not necessarily hold for every single individual.

What can the educators do to facilitate for the students to develop their understanding? According to the phenomenographic tradition, the learning process is a question of *discerning* new aspects of phenomena. A specific aspect cannot however be discerned without experiencing *variation* in a "dimension" corresponding to that aspect. These dimensions are characteristic for the specific aspects, and the variations make central features of these aspects visible.

In order to identify the variation necessary for learning, we use a theoretical framework to describe understanding as a whole. This framework, provided by Marton and Booth (Marton and Booth, 1997), states that understanding comprises two aspects: the meaning the particular understanding captures, and the focal awareness on certain aspects of the phenomenon, here called the focus of the understanding. The meaning aspect of an understanding is called the referential aspect, the focus of a specific understanding is called the structural aspect.

From the empirical data we have identified three qualitatively different categories of understanding, see Table 4. We have expressed each category of understanding in terms of its referential aspect. Intending to discuss implications for teaching, we will now make explicit also the structural aspects of *class* and *object* corresponding to each of the identified categories of understanding. An example of this is the first category in Table 4, where the students have experienced classes as entities of the program, contributing to the structure of the code. The focus of this understanding of a class, is the appearance of the program with classes as separate entities. The structural aspect of this category is hence found.

Learning requires discernment of a new structural aspect of the phenomenon, and discernment requires variation in a dimension corresponding to the structural aspect.

We choose an example, not from the present study to illustrate this. How is a circle defined?

Figure 1: How is a circle defined?

To understand what a circle is, the *size* is one relevant aspect. To be able to discern *size* as an aspect, that is to have the focal awareness on this aspect,

circles of different sizes are needed. If a person only observes one circle, he or she might not discern that *size* is one aspect when describing circles.

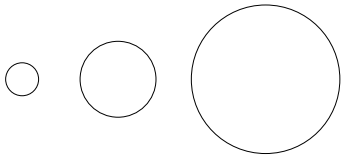

Figure 2: The aspect that a circle has a size is possible to discern when showing many circles with different sizes.

A variation in a dimension corresponding to the aspect that circles have sizes brings the person to focus on this aspect, and to discern it.

The reason for choosing the theoretical framework mentioned above to analyse understanding is thus twofold. The framework gives us a tool to get an overall picture of an understanding, with referential and structural aspects. The structural aspects then provide a basis for the analysis to find the dimensions of variation, necessary for the learning process. The structural aspects of the understandings expressed in the empirical data are discussed in the next paragraphs, while the necessary variation, found when analysing the structural aspect, is discussed in the paragraphs that follows.

In the first category in Table 4, the students have experienced classes as entities of the program, contributing to the structure of the code, and objects as a piece of program text. The focus of this understanding of a class, the structural aspects, is, as mentioned above, the appearance of the structure of the program text. The focus of the understanding of objects, is on the program text.

In the second category, classes are experienced as descriptions of properties and behaviour of objects, where objects are understood as something active in the program. The focus in this category is on what happens during execution of the program, in particular on the objects created and how they contribute to different events at run-time[1]. The objects are the active parts of the program, accomplishing the task given.

In the last category, class is experienced as a description of properties and behaviour of objects, where object is understood as a model of some real world phenomenon. The focus is still on the class' description of the active objects, but now with an emphasis on the reality aspect of the class description.

The structural aspects, the students foci, are hence identified. Variation in a dimension corresponding to the structural aspect is, as discussed above, a prerequisite for learning to take place . Having expressed the structural aspects of the concepts *object* and *class*, as captured by the categories of description in Table 4, it is now possible to discuss what dimensions of variation correspond to each category. In the first category in Table 4 the structural aspect is the

[1]For readers not familiar with programming: by "run-time" we mean the period of time when a program is running.

program text and the appearance of the text. To be able to focus on this aspect, students need to discern that in different programs objects and classes appear in different ways. In that sense, the textual representation of programs constitutes a dimension of relevance for the understanding of *object* and *class*. Different, specific program texts constitutes values along this dimension.

To be able to discern the understanding expressed in the second category, the students need to focus on the objects the program creates and events happening at execution of the program. Here, the relation between class description, object action, and resulting events during the execution of the program constitutes a dimension. Different specific cases of such relations provide values along this dimension. The variation between these values can enhance an awareness of object and class corresponding to the second category of understanding, according to Table 4.

In the last category in Table 4, the structural aspect is the active objects in the program, described by the class, with an emphasis the reality aspect of the class. In this case, the relation between class, object and real-life phenomena constitute a dimension. Different specific cases of such relation, constitute values along this dimension.

The line of reasoning above is summarized in Table 5. It includes both the referential aspects of the concepts *object* and *class*, see the left hand column in Table 5 and the structural aspects, see the mid column. In the right hand column we have included what we conclude to be the corresponding dimensions of variation.

5.2 Implications for education

Table 5 has implications for teaching. Teaching is here defined in a wide sense. By teaching we mean everything that supplies resources for learning. Examples of such resources could be programming assignments, laborations, software tools, lectures, internet and fellow students, anything the students choose to use in their learning. The whole organisation of the learning environment is in this sense teaching.

A general implication for teaching is to make resources in the learning environment available that help students to discern the aspects mentioned in Table 5. The teacher can create the conditions for such discernment with the judicious use of variation. By varying the teaching and holding the critical aspect of the phenomenon invariant, that critical aspect is lifted out of the surrounding "noise". We speak of opening a dimension of variation, in which taken-for-granted ways of understanding are now brought into focus.

The results in Table 5, developed and clarified in the previous section, can be implemented in the teaching and learning environment offered to the students, in a number of ways. There is a great freedom and possibility to adapt the results to the need and desire of each teacher, study group and learning resources. The following paragraph discusses possible ways to achieve this, showing a few examples.

The class and object concept, the referential aspect	The class and object concept, the structural aspect	Dimensions of variation
Object is understood as a piece of program text. Class is experienced as an entity of the program, contributing to the structure of the code and describing the object.	Focus on the program text and the syntax rules of the programming language. Focus on the structure of the program.	*Variation of the text in the program.* *Variation of the appearance of the structure of different program texts.*
Class is experienced as a description of properties and behaviour of objects, where object is understood as something that is active in the program.	Focus on the objects that the program creates and events happening at execution of the program.	*Variation of the relation between objects and events when the program is executed.*
Class is experienced as a description of properties and behaviour of the object, where object is understood as a model of some real world phenomenon.	Focus on the objects that the program create, the reality the class depicts.	*Variation of the relation between objects and concepts within the problem domain of the program.*

Table 5: Categories describing the different understandings and the corresponding aspects of variation of the phenomenon *object* and *class* found in the group.

An example on how the students can discern the aspect that classes and objects have something to do with the program text and its structure, is that students need to become aware of that different programs represent classes and objects differently, on a textual level. This corresponds to the first part of the first category. In the second part of category one, the focus is on the structure of the program text. There are however several aspects of a program structure. A single class has a structure in terms of its attributes and methods. Students also encounter problems including several classes where each class is an entity of the program. Both these aspects of the structure of the code is possible to expose in the teaching and the students discern it. A way to achieve this is to use a variety of simple UML class diagrams (Rumbaugh et al., 1999). To transfer the structure from the diagram to the code where the methods are separated from the attributes is possible even if there is only one single class. This is often the case in the examples considered in the beginning of a programming course. The

aspect that the class is a help when structuring the program is made even more apparent when more than one class is used to solve a problem. Each class is represented in a UML diagram and forms its own entity of the program.

Another example is in the second category, which has a focus on the relation between class, objects and events during program execution. Let one object call different methods and check the result after each method call. Also let the same method be called by several different instances of one and the same class and check the results. The aspect that objects are used to make things happen can thus be discerned. An example of a resource that can be used for the examples mentioned is BlueJ (Barnes and Kölling, 2003), where class diagrams are used to illustrate classes and relations, and with a debugger showing the values variables get during execution.

The last category in Table 5 includes the aspect that objects and classes model the real world. The study points to the importance of letting students follow the whole process of a programming task, even including the analysis of a problem in real life. This can be implemented in teaching by using an assignment where several classes are needed. The first part of the assignment would be to do an object oriented analysis of a real world problem, deciding which classes are needed, which methods each class should include, and which information the classes need to exchange. If the students are in their first programming course, they after this may be given a suggestion on suitable classes with attributes and methods before starting to code. After implementing and testing the code, the students are supposed to discussed in groups their different solutions and how their final solution differ from their first analysis. This might help the students to discern the real world aspect of objects and classes, and also to discern the differences between the real world problem and the implementation of the problem.

Our results can shed new light upon and give explanation to other research and discussions in the field. The following paragraphs show some examples of this.

The importance of letting the students follow a whole programming task, not only the coding is commented in Computer Curricula 2001 section 7.2 (Roberts and Engel, 2001):

> Introductory programming courses often oversimplify the programming process to make it accessible to beginning students, giving too little weight to design, analysis, and testing relative to the conceptually simpler process of coding. Thus, the superficial impression students take from their mastery of programming skills masks fundamental shortcomings that will limit their ability to adapt to different kinds of problem-solving contexts in the future.

This is in line with the discussion above on the need for the students to follow a whole programming task, including the analysis to find suitable objects in a real world problem, to get a good understanding of object-oriented programming. Our analysis above provides empirical and theoretical support for this statement.

Holland, Griffiths and Woodman list some misconceptions noticed at distance courses where Smalltalk was taught, in one introductory undergraduate

course, and one postgraduate course (Holland et al., 1997). One misconception mentioned is "object as a kind of variable". Students with previous experience of procedural programming may, if the examples they first come across have only one instance variable, develop the misconception that objects are in some sense mere wrappers for variables. It is trivially easy to avoid this misconception by ensuring that all the classes showed as an introduction, have more than one instance variable and that they are of different type. Another misconception that can appear is if the data aspect of objects is overemphasized at the expense of the behavioural aspect. This misconception can be avoided by using introductory object examples where the response to a message is substantially altered depending on the state of the object. Both the misconception "object as a kind of variable" and the overemphasizing of the object's data aspect is an indication of the importance to attain an conception according to the second categories in Table 1 and Table 2. The second category in Table 1 emphasizes that objects are active during execution of the program. This points to the behavioral aspect of objects. The second category in Table 2 explains classes as a description of both data about the object, and methods explaining the behaviour of the object. As explained in section 5.1, the relation between class description, object action, and resulting events during program execution constitutes a dimension where variation is needed. This implies variation in values of several instance variables, caused by several method calls. This is according to the recommendations from Holland et al.

A common problem among novice programmers, also mentioned by Holland et al, is to understand the difference between *class* and *object*. This is obviously a problem if several examples are presented in which only a single instance of each class is used. To avoid this, good practice is always to work with several instances of each class, see category one Table 5. As explained in section 5.1, the textual representation of programs constitutes a dimension of variation. This implies variation in the sense of presenting more than one instance of the class in the code, as recommended by Holland et al.

In the light of the present study, the recommendations from Holland et al can be summarized as *variation of dimensions corresponding to critical aspects of the understanding is of great importance*. These dimensions of variation are not only pinpointed in the present study, but also explained in the theory of phenomenography and in the analysis of the data by applying variation theory on the results of the study.

Holmboe (Holmboe, 1999) performed a study where a few people of different background were asked to describe in their own words what object-oriented programming is. He asked students who had just finished an introductory course on object-oriented programming, senior students tutoring the same course and professors of Computer Science or System Engineering. He made a qualitative analysis of the answers, and comment that some types of knowledge are more suitable as basis for further knowledge construction than others. When analysing the results from the study he writes about understandings which includes the world outside the computer itself: "A person with holistic knowledge relates the implementation and design of a computer program to the real world

being simulated." Holmboe emphasizes the importance that "[...] more students will experience the connection between reality, model and implemented program, and thus reach holistic knowledge of object-orientation sooner in their learning process." The third category in Table 1 and Table 2 capture an understanding of classes and objects that includes the world outside the computer itself. The dimensions of variation found and discussed in section 5.1 are valuable knowledge for teachers to facilitate for the students to reach this understanding.

In Fleury's study on student's constructed rules (Fleury, 2000), she stresses, with a reference to Holland, Griffiths and Woodman (Holland et al., 1997), the importance of carefully constructed sample programs to avoid misconceptions of concepts. Our study stresses the importance of designing the education so that the students can discern the critical aspects of the understanding. Carefully constructed sample programs in this sense means variation of dimensions corresponding to these critical aspects. This is applicable not only on sample programs, but in all different aspects of the learning environment.

For the Java educator, one challenge is to construct an educational environment which facilitates for students to reach a rich understanding of the concepts *object* and *class*. To this end it is important to know the different ways in which students (as opposed to experts) typically experience these concepts. Our phenomenographic study has given such insight. Next the educator needs to identify what variation the students have to discern in order to become aware of aspects belonging to a rich understanding of these concepts. Here, variation theory can be a useful tool, as demonstrated in the previous discussion.

5.3 Explicitness and Variation Theory

Variation in dimensions corresponding to critical understandings of a phenomenon can make understandings explicit to the students. Explicitness in the teaching, where teaching is defined in a wide sense, see Section 5.2, is of great importance. Explicitness in teaching has however several dimensions. The teaching, including not only the educator but also all other resources the students use, see Section 5.2, should be explicit in terms of *what* the students are suppose to learn. In addition, the learning process must be explicit in terms of using dimensions of variation so that the students can discern the object of learning. The students on the other hand, need to be offered opportunities to give explicit feedback to the teaching. The educator need to organise forms of feedback in order to find what the students have learned.

Learning is sometimes in constructivistic contexts expected to take place in the way that the student discovers on his/her own what he/she is supposed to learn, without anyone telling explicitly what he/she is supposed to discover. This is criticized by some authors. In a study on school children's' understanding of some physics phenomenon, Säljö showed that the intended learning did not necessarily take place when the concepts to be learned were not explicitly expressed but only illuminated by experiments (Säljö and Bergqvist, 1997).

Our results support this critique. Examples from our empirical data emphasize the importance of explicitness in the teaching, and not to presuppose

knowledge. Student D says:

> D: [...]But, no, in some way it feels like there are a lot you're expected to know and then a lot you're expected just to understand immediately and to put in the right context. Yes.
> I: Is it the conception, what's on your mind then, what do you mean with that it's a lot?
> D: It's so difficult, it feels like everything is just floating like this (laughter), ehm. No it is, what is it about. Therefore, oh...

About learning of concepts student F says:

> F: (sigh) Object, it's hardly that I even come to any real grips with it. [...] I think you got rather bad conceptions of it anyway, this object oriented. You heard the word all the time but I find it very hard to put my finger on what it is and what's the difference compared to other programming languages. [...] No but they say, it says if you read and everybody says that Java is an object oriented programming language. But it doesn't tell me very much actually.

And later:

> I: [...] Eh, do you think it has been difficult to understand this?
> F: Yes I do. It easily becomes very abstract. So it is very difficult to get a grip on it. Which is which and how it works, yes, for instance classes and objects and what it actually is and what it actually comes out to be everything. It turns out to be a lot, difficult to get a real understanding of it.

In many teaching situations the concepts the students are supposed to learn are not explicitly taught, the students are supposed to realise themselves what is quite obvious to the teacher, but presupposed.

In this way variation in dimensions corresponding to critical understandings of phenomena presuppose explicitness in the teaching. When the educator is aware of the aspects of an phenomenon that are critical for the understanding, he or she needs to provide explicitness in the teaching and learning environment, so that students can discern these aspects.

Similar observations were made by Fleury (Fleury, 2000). She made a study on student-constructed rules in a beginning programming course where Java was taught, showing misconceptions in the understandings. If concepts are not correctly learned, students might construct their own incorrect understandings. To change an understanding can prove to be difficult.

An example from the previous study is student J, who is in the end of the study course and has almost completed all the compulsory Java assignments. Student J still has vague ideas about the concepts *object* and *class*:

> I: [...] what do you think about what an object is?
> J: What an object is. When I have discussed and so on then I have said that an object is a class or can be a class, or can be a method in a class. Eh, when I have been studying I have tried to think of if you have a

> programme, I don't remember but? which consists of a couple of objects and the objects can be different classes which contain methods. In that way I tried to get a picture of what an object is, because it is easier for me to think that if I have a class and in the class we have a lot of objects.
> I: Although now you drew it the other way around ...
> J: Yes because it was like that I thought about it when I read the book, so I don't know really which is right. It's different when you ask different persons, yeah but, class is that an object. Yes it is an object but a method was also an object. Is it the class which contains different objects or is it an object that contains different classes. So that I don't know.

Our study also illuminate that laboratory assignments might not always be sufficient for the students to create a good understanding, depending on how they are designed.

Student L and F discuss learning of concepts in relation to lectures and programming assignments:

> I: [...]. Do you think it has been difficult to understand objects and classes?
> L: The class is no problem but it's the way I mix all the conceptions together, so I don't know if I have had that clear image of objects and there were no one who really used the word object .
> I: Not on the lectures?
> L:No I don't think so. [...] once in a while sometimes but not referred to them so to speak as they are called, particles and vectors, so to speak, never said really that it's an object. Possibly he said it in the beginning, yes this is what an object is, then I didn't really think he used the word.
> I: So when you have written some code and so, it has not emerged that right here ...
> L: No.
> I: ... creates an object or something?
> L: No, I don't think so.

> F: [...] I think you use Java the way you learned it and programme but still it's difficult to return to basic and what the actual object is, that is the central idea and so on.

Curricula 2001, Section 7.2 has similar discussions on problems to learn if "essential character of programming" is overshadowed by working with code only: "Moreover, concentrating on the mechanistic details of programming constructs often leaves students to figure out the essential character of programming through an *ad hoc* process of trial and error." (Roberts and Engel, 2001). In that respect, this study contributes to pinpointing what might be problematic with learning by programming assignments only. If there is no explicitness in what the assignments are supposed to illustrate, and how to understand central concepts, learning might not take place and misconceptions occur.

We notice an interesting connection between explicitness as discussed above, and the concept *reflective learning* introduced by Linder and Marshall (Linder and Marshall, 2003). They write that the phenomenographic perspective

> ... is a dynamic relationship between *focus* and *meaning*, and that in this dynamic relationship it is the *shifting of focus* that facilitates the kinds of discernment that generate the necessary variation for learning. From this point of view, making sense of something requires a mindful kind of learning that is both purposeful and deliberate–a fundamental aspect of what we propose calling *reflective learning*.

Later they write about the learners:

> ... they need to confront those aspects of the phenomena which are taken for granted to become invariant, and vary them. As such *reflective learning* is the exploration of *the object (the content)* of learning through a mindfulness of the *act* of learning.

From the teacher's point of view, learning that is "purposeful and deliberate" demands *explicitness* in the teaching of the phenomenon with all its different aspects, implying that no aspect must be taken for granted. This may help the students to achieve *reflective learning*. In this sense again, explicitness corresponds to the teacher's perspective of the learning situation, which enhance reflective learning from the students perspective.

5.4 Learning in a context

In previous sections we have discussed the different meanings of the concepts *object* and *class* and how this could influence *the teaching*. Table 3 however describes the *purpose* of using objects and classes. It turns out that this also points to factors that might be important for the teaching. The students understanding expressed in the first part of the first category in Table 3, *"the purpose is understood from a code perspective: the syntax requires it"*, is notable. The understanding that the reason for using objects and classes is because the compiler requires it shows a total lack of understanding of what the object-oriented paradigm is about, why it has appeared and its advantages and disadvantages compared to other programming paradigms. Several students also express that they do not know any other way to program, that they even question if there are other ways to think.

> I: What do you think is the point with having objects and classes?
> B: That I don't know, what else would it be so to speak, if you didn't have it. It's necessary, isn't it, so to speak.

Another student says about the point of using objects and classes:

> N: It is because the programmes require it. That's the way it goes to create a program. I don't know how to do it otherwise.

Compare their answers above with their answers on the question if it has been hard to understand objects and classes:

> I: Do you find it difficult to understand classes and objects?
> B: Yes I think so. It is very difficult to know if you kind of have understood it correctly because you could as well have understood quite wrong things. I don't know if I have understood it correctly.

And student N says:

> N: I have a very vague image so I find it very difficult.

The conspicuous connection between want of seeing the point of using objects and classes and the experience that it is difficult and unintelligible is notable. On the other hand, the rich understanding of the point of using objects and classes formulated in category three in Table 3, *"The purpose is understood from a reality perspective: support to connect reality and programming"*, expresses one of the main thoughts behind the object oriented paradigm. This understanding is elucidated if the object oriented paradigm is explicitly evolved to the students. This also implies that there are more programming paradigms. To understand why the question of different paradigm exists at all, the students need a context that involves something from the historical background and the problems that enforced the development of the different paradigm, and the contexts where they are used.

The phenomenographic analysis in the study revealed that the understanding expressed in the last category in Table 3 corresponds, with few or any exceptions, to a rich understanding of the concepts *object* and *class* among the students. The connection between fundamental programming concepts in Java, and the understanding of the programming paradigm itself is stressed by Hadjerrouit (Hadjurrouit, 1998). She writes:

> It is critical to understand that Java is not only a programming language, but that it is also an emerging paradigm with a set of fundamental concepts that can be used to explore a wide range of problems that was previously beyond the reach of computing

Here she mentions "The concept of object for designing software as a set of interacting objects." Later she writes about "viewing Java as a computing paradigm organised around a set of fundamental concepts." Understanding central concepts within object-oriented programming is fundamental, and is closely related to understanding the object-oriented paradigm itself.

A rich understanding of the concepts *object* and *class* includes an understanding that classes and objects are models of real world phenomena. In the present study only a few students expressed these understandings. To be able to discern these understandings of the concepts, the study points to the importance of letting students follow the whole process of a programming task, even including the analysis and design of a real world problem. Holmboe writes about understanding object-oriented programming which includes the world outside the computer itself: "A person with holistic knowledge relates the implementation and design of a computer program to the real world being simulated."

(Holmboe, 1999). He emphasizes the importance that "[...] more students will experience the connection between reality, model and implemented program, and thus reach holistic knowledge of object-orientation sooner in their learning process." He advocates a focus on object-oriented analysis and design early in the education, even prior to introducing of a programming language.

Curricula 2001 comments the importance of design and analysis in the programming education: "Introductory programming courses often oversimplify the programming process to make it accessible to beginning students, giving too little weight to design, analysis, and testing relative to the conceptually simpler process of coding."

The research and comments referred to above point in the same direction as the present study. To follow a whole programming task, including analysis and design, puts the programming in a context and can help the students to get a richer understanding of the object-oriented paradigm and fundamental concepts within the paradigm.

The discussion of the importance of putting programming in a context, here discussed in terms of knowledge of different programming paradigms with the historical development that lead to object oriented programming, and the importance for the students to follow a whole programming task, are interesting issues that need more investigations in later studies.

6 Conclusions and future work

In the study presented, fourteen university students taking a first programming course in Java have been interviewed on their understanding of the concepts *object* and *class*. The question of interest was how students understand abstract concepts in object oriented programming, and the study aimed at identifying qualitatively different understandings found in the group.

Via a phenomenographic analysis of the interviews, we have identified different understandings of the concepts among the students. The understandings are summarised in three *categories of description*, see Table 4. Each understanding is important and relevant when solving a programming task, and a good understanding includes all aspects. The categories are inclusive, meaning that the understandings expressed in the latter categories include the understandings expressed in the former. The categories describe the understandings at a group level, see Section 3.4, which implies that this pattern does not necessary hold for every single individual. The first category of understanding in Table 4, does however hardly correspond to a professional way of understanding the concepts. It explains the concepts at a code–level. In the process of learning object oriented programming, a crucial step is to exceed an understanding at this level. To help the students to discern all the different aspects of the understandings, the present study points out the importance of students getting the opportunity to follow a whole programming task including the analysis of the problem, the design, implementation and testing of the program. To help the students understand the point of using objects and classes, there might be

a need of explaining something about other programming paradigms and about the historical development that lead to object oriented programming.

How does a student exceed his or her understanding of a phenomenon? The phenomenographic tradition states that there is always a limited number of qualitatively different categories, or aspects of a certain phenomenon, and learning means to *discern* new aspects of that phenomenon. A specific aspect cannot however be discerned without experiencing *variation* in a "dimension" corresponding to that aspect. These dimensions are characteristic for the specific aspects, and the variations make central features of these aspects visible. By using variation in the learning situation and holding the critical aspect of the phenomenon invariant, that critical aspect is lifted out of the surrounding "noise". We speak of opening a dimension of variation, in which taken-for-granted ways of understanding are now brought into focus. The question of variation gives interesting implications for teaching, discussed in Section 5.

The discernment requires the students to have a mindful kind of learning, *reflective learning*. This implies for the teachers that *explicitness* in the teaching is of great importance. There is a risk when central background knowledges and some aspects of the understanding are presupposed, that students do not discern these aspects. All aspects are important to get a good understanding of the concepts, and should be explicitly taught and not presupposed. Teachers thus need to be aware of different understandings of the concepts *class* and *object* that can occur among the students, see Table 4.

The conclusions point to interesting issues for future work. By combining phenomenography and activity theory (Berglund, 2004) in coming studies and analyses, we plan to identify resources that promote learning of object oriented programming. This might aid both students in their learning process and institutions that offers programming education.

This study shows qualitatively different understandings of some central concepts in object oriented programming found in a group of university students. The results are interesting per se for teachers and educators, but they also point to interesting implications for planning of educations in programming. The learning environment offered to the students should support variation and explicitness of all different aspects that together form a good understanding of central concepts, and an awareness of these aspects of the concepts and the importance of explicitness in the teaching among teachers and educators are of great importance.

References

Bar-David, T. (1993). *Object-Oreinted Design for C++*. P T R Orentice Hall.

Barnes, D. and Kölling, M. (2003). *Objects First with Java - A Practical Introduction using BlueJ*. Prentice Hall / Pearson Education.

Berglund, A. (2002). *On the Understanding of Computer Network Protocols*. PhD thesis, Uppsala University, Department of Information Technology.

Berglund, A. (2004). A framework to study learning in a comples learning environment. *ALT-J*, 12:65–79.

Box, R. and Whitelaw, M. (2000). Experiences when migrating from structured analysis to object-oriented modelling. In *Proceedings of the Australasian conference on Computing education*, pages 12–18. ACM.

Fleury, A. E. (2000). Programming in java: Student-constructed rules. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*.

Hadjurrouit, S. (1998). A constructivist framework for integrating the java paradigm into the undergraduate curriculum. In *ACM SIGCSE Bulletin , Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education, Volume 30 Issue 3*.

Hamilton, J. A. and Pooch, U. W. (1995). A survey of object-oriented methodologies. In *Proceedings of the conference on TRI-Ada '95: Ada's role in global markets: solutions for a changing complex world*.

Holland, S., Griffiths, R., and Woodman, M. (1997). Avoiding object misconceptions. In *ACM SIGCSE Bulletin , Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education, Volume 29 Issue 1*.

Holmboe, C. A. (1999). Cognitive framework for knowledge in informatics: The case of object-orientation. In *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*.

http://www.vr.se (2003). Retrieved November 28, 2003.

Kölling, M. (1999). The problem of teaching object-oriented programming, part i: Languages. *JOURNAL OF OBJECT-ORIENTED PROGRAMMING*.

Kvale, S. (1997). *Den kvalitativa forskningsintervjun*. Studentlitteratur.

Linder, C. and Marshall, D. (2003). Reflection and phenomenography: towards theoretical and educational development possibilities. *Learning and Instruction*, pages 271–284.

Marton, F. and Booth, S. (1997). *Learning and Awareness*. Lawrence Erlbaum Ass., Mahwah, NJ.

Marton, F. and Svensson, L. (1979). Conceptions of research in student learning. *Higher Education*, pages 471–486.

Meyer, B. (1988). *Object-oriented Software Construction*. International series in Computer Science. Prentice Hall.

Roberts, E. (2004). The dream of a common language: The search for simplicity and stability in computer science education. In *Proceedings of the thirty-fifth SIGCSE technical symposium on Computer science education.*

Roberts, E. and Engel, G. (2001). Computing curricula 2001: Final report of the joint acm/ieee-cs task force on computer science education. IEEE Computer Society Press, December 2001, http://www.acm.org/sigcse/cc2001/.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Addison Wesley Longman, Reading, Massachusetts.

Säljö, R. and Bergqvist, K. (1997). *Seeing the light. Discussion and practice in the optic lab. In L. Resnick, R. Säljö, C. Pontecorvo & B. Burge (Eds.). discourse, tools and reasoning. Essays on situated cognition.* Springer-Verlag, New York.