IT Licentiate theses
2006-006

Novice Students' Learning of Object-Oriented Programming

ANNA ECKERDAL

UPPSALA UNIVERSITY
Department of Information Technology

UPPSALA
UNIVERSITET

Novice Students' Learning of Object-Oriented Programming

BY
ANNA ECKERDAL

October 2006

DIVISION OF SCIENTIFIC COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY
UPPSALA UNIVERSITY
UPPSALA
SWEDEN

Dissertation for the degree of Licentiate of Philosophy in Computer Science with specialization in Computer Science Education Research
at Uppsala University 2006

Novice Students' Learning of Object-Oriented Programming

Anna Eckerdal
Anna.Eckerdal@it.uu.se

Division of Scientific Computing
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

http://www.it.uu.se/

ISSN 1404-5117
Printed by the Department of Information Technology, Uppsala University, Sweden

Abstract

This thesis investigates students' experiences of learning to program. Learning to program is a complex activity. It involves elements of learning abstract concepts as well as both learning and using advanced resources like computers and compilers. The learning experience is affected by factors like students' motives to learn and their general understanding of what learning to program means. These issues form the basis for the four research themes addressed in this thesis, specifically: students' experiences of what learning to program means; how students understand central concepts in programming; how students use and experience help from resources; and students' motives to learn to program.

The thesis presents a qualitative study on novice students' experiences of learning object-oriented programming. Data was collected via semi-structured interviews. The interviews were analysed mainly using a phenomenographic research approach. The analysis resulted in the formulation of categories of description of students' qualitatively different ways to understand what learning to program means. In addition, categories describing different ways to understand the concepts *object* and *class* in object-oriented programming were formulated. From an educational point of view, these results can be used to identify aspects of learning to program that are critical from the students' perspective.

The analysis of students' use of resources revealed that some resources were mainly used in a search-for-meaning way that promotes good learning, while another group of resources were mainly used in a superficial way. The two groups of resources seem however to interact with each other when students take responsibility for their own learning, which in particular characterizes their work with the larger computer assignments. When working with those, the students describe that both groups of resources were important for the learning.

The analysis of students' descriptions of their motives to learn pinpoints motives that can enhance learning.

In the study there were students who expressed that they had problems to know how to go about to study computer programming. This might indicate problems about knowing how to use available resources efficiently. Students who do not know how to use resources like the compiler in an efficient way, will have difficulties to perform assignments, which is expressed by the students as very important for the learning of concepts. The results also indicate the importance for educators to provide a learning environment with a variety of resources which can connect to students' different motives to learn, pointed to in the study. In this way all four aspects of the learning experience examined in the present study are important for students' learning of object-oriented programming.

Parts of the work presented in this thesis have appeared in publications after peer review:

- The results in Chapter 4 have been published in a shorter version: (Eckerdal and Berglund, 2005)
- The results in Chapter 5 have been published in a shorter version: (Eckerdal and Thune', 2005)

Acknowledgements

I want to thank my supervisors, Michael Thuné, Uppsala University and Shirley Booth, Lund University for their support and advice during the process that lead to this thesis. I am particularly happy for their encouragement to try new ideas and methods, still scaffolded to stay within the academic domain, characterized by a rigorous way to think and work.

I also want to thank my colleagues in the research group at the department, particularly Anders Berglund, who has given valuable feedback and encouragement.

The work behind the thesis would never have been fulfilled without support from my family, Per, Nils and Olof. In this thank I also include my mother Anne-Mari Sundin who has encouraged me through the whole work, and to start it by saying:

Bättre lyss till den sträng som brast än aldrig spänna sin båge.

The work with the thesis has been financed by The Swedish Research Council, and Faculty of Educational Sciences, Uppsala University.

Contents

1 Introduction

Computer programming is one of the core areas in computer science education. Even many non-major computer science students in technical and natural science education at Swedish universities take at least one compulsory computing course where they gaining an introduction to programming. This can involve basic knowledge of what programming means in general, a conceptual understanding in the subject area and a knowledge of complex resources like compilers and how computers work.

There is an ongoing debate among educators on how to introduce programming to novice students (Joint Task Force on Computing Curricula, 2001) where several different approaches have been suggested. My interest is to investigate how students go about learning to program, and what they learn, from the students' perspectives. Students' own experiences of learning fundamental programming is interesting to study and can inform the dealing with programming education.

The focus of this thesis is on novice students' learning of object-oriented programming. The research presented aims to give a broad picture of students' experiences of their learning including both learning outcomes and the way in which the students go about learning.

1.1 The research questions

This thesis builds on empirical data concerning novice students' experiences of learning to program. Learning to program differs in some aspects from many other subjects students met at university level (Daniels et al., 1999). Many students have little or no previous knowledge of the complex resources like compilers and how computers work. These resources play a significant role in learning the subject. Furthermore many students have not encountered the subject before their first university course.

The main focus of the research presented in the thesis is students' experiences of their learning in a programming course. This involves the students' experience of what learning to program means in a specific course. Other aspects of this experience taken into consideration are students' conceptual understanding, students' experience of their learning environment, and students' motive to learn.

Aspects of students' experience of the learning environment focused on in the thesis are students' use of resources in the learning process. The reason for this is twofold. Some resources used in the course are part of learning the subject itself, and are thus an important aspect of the learning experience. The learning environment, as defined by Entwistle (2003, p. 7) spans too broad a research area to be covered in this work and motivates a limitation.

The research questions posed are thus:

- *How do students understand what learning to program means?*
- *How do students understand abstract concepts in object-oriented programming?*
- *How do students use resources when learning computer programming and what are their experiences of the support they provide?*
- *What motives to learn computer programming can be found among the students?*

The four research questions mentioned are studied, analysed and discussed separately in the thesis, but the way in which they relate to each other is also considered. In this way a broad picture of novice students' experiences of learning object-oriented programming is painted.

1.2 Descriptions of terms used in the study

This section gives a list of terms used frequently in the thesis and describes the way in which they have been used. Other terms are defined when they are introduced in the text.

Code refers to the instructions which tell the computer what to do, written by a programmer. These instructions follow rules from the particular programming language used.

Software includes the computer programs, associated documentation and configuration data that is needed to make the programs work correctly. The purpose of producing software systems is to make computers solve problems.

Computer science is defined in a wide sense including "theories and methods that underlie computers and software systems" (Sommerville, 2004)

Software engineering "is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use. [...] Some knowledge of computer science is essential for software engineers" (Sommerville, 2004, p. 7)

Programming paradigm. There exists several fundamentally different ways to tackle a problem for a program developer. Consequently there are different programming paradigms available. This thesis will discuss the object-oriented paradigm which is the dominate paradigm currently used in industry and university education. Examples of programming languages within the object-oriented paradigm are Java and C++.

1.3 Overview of the thesis

The thesis has the following outline. Chapter 1 discusses the research questions and related work. The research approach is described in Chapter 2, and the study performed is presented in Chapter 3. Chapter 4 describes students' understanding of what learning to program means in a first course in object-oriented programming. Chapter 5 presents the results from the analysis on students' understanding of the concepts *object* and *class* in object-oriented programming. In Chapter 6 the analysis of the students' use of resources is evolved, it describes how the resources were used and how the students experienced that the resources supported them in learning to program. Students' motives for learning to program are discussed in Chapter 7 by presenting a few, from an educational perspective interesting students' motives to learn. All chapters include discussions and implications for education, concerning the specific topic in the chapter. The last chapter in the thesis presents conclusions drawn from the whole study and discusses future work.

1.4 Related work

This section reviews previous research on students' learning of programming, and investigates research related to the research questions on students' understanding of concepts, students' understanding of what it means to learn to program and students' use of resources.

1.4.1 Students' learning to program

Examples of studies that give nice overviews of the research within programming education are Booth (1992) and Robins et al. (2003), where Booth covers a somewhat older spectra of the literature than Robins et al.

Many papers have been written on students' difficulties to learn to program (Ben-Ari, 1998; Fleury, 1999; Fleury, 2000; Fleury, 2001; Kölling, 1999a). A study much referred to is McCracken et al (2001), a multi-national, multi-institutional study showing that first year students do not know how to program after their first programming course. Other multi-national, multi-institutional studies are Lister et al. (2004) who showed that novice students have problems to predict what a short piece of code would do, and also to put in the right piece of missing code when asked to select from a small set of codes, and Eckerdal et al. (2006) who investigated senior students' ability to design computer programs and found that only few students have a satisfactory design ability at the end of their computer science education.

These studies give a solid foundation for the statement that students, both at novice and higher levels, have difficulties to learn to program.

1.4.2 Students' learning of central concepts

Many studies point at the necessity of good understanding of central concepts within object-oriented programming. Some of these concepts are necessary for students to learn at an early stage of the programming education. Holland, Griffiths and Woodman (1997) claim that misconceptions of basic 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."

Fleury (2000) found that students constructed their own understanding of concepts when they worked with programming assignments, and that 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." writes Fleury.

Holmboe (1999) 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." Box and Whitelaw (2000) argue from a constructivist learning theory, that more abstract types of learning are required by the student for object-oriented software technology than for structured software technology.

1.4.3 Students' understanding of what it means to learn to program

The question how students understand what it means to learn to program has been investigated in previous studies. Booth (1992) studied undergraduate engineering students' experience of what it means and what it takes to learn to program. Bruce et al (2004) investigated first year university students' early experiences of computing, with a focus on revealing differences in how they go about learning to program. Both indicate that it is important for students to get an overall understanding of what learning to program means.

1.4.4 Students' use of resources

Resources for learning to program are frequently discussed topics in conference papers and journal articles. This section first discusses related work on resources that are not mentioned, or only slightly touched upon by the students in the present study, but frequently discussed in the computer science education community. After that follows a discussion on related work on the use of the resources presented in this study.

Examples of resources that are not mentioned in the study presented in this thesis are *technology supported resources* like visual programming tools, and *collaboration methodologies* like extreme programming/pair programming collaboration and collaboration used in Problem Based Learning. The students in the present study do not discuss technology supported resources other than the compiler. A few students also mention internet as a resource for finding information, and one student discusses that he or she would prefer a more advanced software development environment in the course. Many students discuss collaboration as an important resource in the learning, but they do not put labels on it, like 'peer programming'.

A reason why students in my study do not mention technology supported resources or collaboration methodologies much, is probably because many of them have not programmed before, and thus are not aware of other resources and terminology than what is offered by the teacher. Below I will mention some examples with references from each area as they are well researched and much used in programming education at higher level.

Technology supported resources

In 2004 the ACM Education Board appointed the Java Task Force. The mission was to develop a stable collection of pedagogical resources that would support the use of Java in first-year computer science courses. The problems focused on the increasing complexity and instability students encounter in new programming languages like Java, which give negative effect on pedagogy. The Task Force has designed new Java packages[1] that for example eliminate the need for a static main method and simplify the development of graphical applications in an object-oriented way (Roberts, 2006). The reports from the Java Task Force with associated material are available from **http://www.acm.org/education/jtf/**.

According to Powers et al. (2006), software resources developed to help novices to learn to program can be divided into *narrative tools*, *visual programming tools*, *flow-model tools*, *specialized output realizations* and *tiered languages tools*. *Narrative tools* "support programming to tell a story ". An example of such software is Alice (Moskal et al., 2004). *Visual programming tools* "support the construction of programs through a drag-and drop interface" as examplefied by JPie (Goldman, 2004). *Flow-model tools* "construct programs through connecting program elements to represent order of computation", with the example Iconic Programmer (Chen and Morris, 2005). *Specialized output realizations* "provide execution feedback in non-textual ways". Lego Mindstorms is a well-known tool (Kay, 2003). Finally *tiered languages tools* "in which novices can use more sophisticated versions of a

[1] Java packages are sets of classes in the programming language, designed for specific tasks, and ready to use.

language as their expertise develops" where ProfessorJ is an example (Gray and Flatt, 2003).

Ellis et al. (1998) report on technology supported resources for Problem Based Learning. For example, the authors discuss resources to provide subject guidance and information access, and resources to assist scaffolding. In the former group reference material like CD-ROM and the web is mentioned. In the latter group visualization and experimenting systems, with references are mentioned. The authors further discuss that "Communication and collaboration tools are often classified according to the pattern of communication that they support." The classification techniques are one-alone, one-to-one, one-to-many and many-to-many. Examples of the techniques are databases, electronic mail, bulletin boards and computer conferences respectively.

Collaboration methodologies as resources

Pair programming has been greatly discussed in the computer science community during recent years. Studies on the results of pair programming, and how pairs best are selected have been performed. Examples of this are VanDeGrift (2004) and Katira (2004). The fundamental thoughts behind pair programming are described as "students sit side-by-side at one computer to complete a task together, taking turns 'driving' and 'navigating.' " (VanDeGrift, 2004).

Extreme programming (XP) has been discussed and used in industry, and to some extent in higher education. In XP planning, analyzing, and designing is done a little at a time, throughout software development. The XP practices also include other factors like pair programming and programmers' collective ownership of the code in the system (Beck and Andres, 2004).

Resources mentioned in the present study

Research on students' use of resources when learning to program has an emphasis on technology supported resources. Research on the resources students mention in the study presented in this thesis is found, but mostly in discussions on single resources in the programming education. Some research on how individual resources are used in programming education is discussed below.

Jenkins (2001) discusses the role of the teacher in programming courses. He discusses teachers' reflections on their teaching in terms of qualitatively different levels. Teachers' roles in computer science education are discussed by Lister et al. (2004) from a phenomenographic perspective. There is plenty of research and literature found on teaching in higher education in general, including discussions on the teachers' role (Ramsden, 1992; Marton et al., 1984).

The role of projects and programming assignments are discussed for example by Daly (2004) and Newman (2003) .

The roles of the programming language and programming environment are discussed in Kölling (1999a) and Kölling (1999b) where the author discusses where different programming languages and different programming environments are suitable.

2 The phenomenographic research approach

Phenomenography was first developed in the 70's in Gothenburg, Sweden by a group of researchers. Ference Marton, Lars Owe Dahlgren, Lennart Svensson and Roger Säljö performed a study on students reading a text. Aimed at understanding differences in outcome of understanding the text, they found clear qualitative variation in *what* the students understood, as well as *how* they went about studying the text. These findings have been used as a point of departure for research in various subject areas in higher education, and have led to insights, such as the distinction between deep and surface approach to learning (Marton et al., 1984). From this empirical basis the phenomenographic research approach emerged.

Numerous phenomenographic studies have since been carried out in different parts of the world, and in different subject areas, and the theoretical separation of learning experiences in *what* students learn and *how* they learn, has shown to be a useful tool to get a better understanding of students' learning experiences. Phenomenography has developed and is now described as a research approach into learning.

2.1 The experience of phenomenon

Phenomenography aims at describing the variation of understandings of a certain phenomenon found in a group of people. *Phenomena* is described by Marton and Booth (1997) as the units that exceed a situation, bind it together with other situations and gives it a meaning. In this thesis I discuss phenomena in terms of central concepts that are critical to understand in order for the learner to progess further with the subject area. It is not limited to single words like 'object', 'class' and 'encapsulation'. It includes aspects of the learning like 'what does learning to program mean?'. In this sense 'phenomena' are something that can bear meaning, relevant for the subject studied.

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 others 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 capabilities 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 I 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, pp. 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). It focuses on the students' perspectives and conceptions, not on misconceptions. It does not take the researcher's perspective as the point of departure, but endeavours to adopt the student's perspective on learning. Marton and Svensson 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. (Marton and Svensson, 1979, p. 472)

A fundamental assumption in phenomenography is that there exist only a limited number of qualitatively different 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.

The analysis is done at a collective 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 to each other, the individuals are put in the background, and the collective understandings of the group are in the foreground.

Marton and Booth (1997) have developed a model for analysing and describing the experience of learning, see Figure 1. The model can be used

as a tool in the analysis to cover central aspects of the learning experience, and to unfold the complex pattern of the experience.

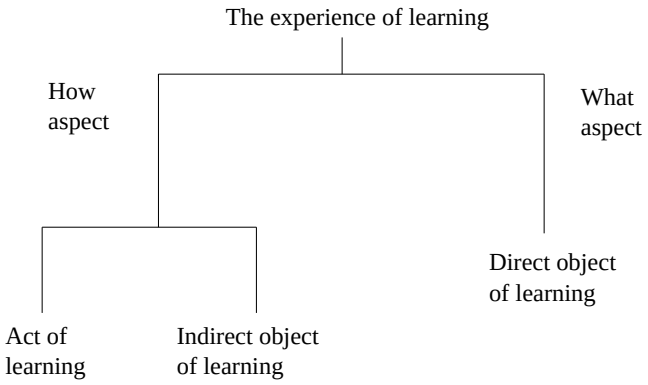

Figure 1: The experience of learning (Marton and Booth, 1997)

According to Marton and Booth, a learning experience can be analytically divided into a *what*-aspect and a *how*-aspect. The *what*-aspect relates to the content of what is being learnt, the phenomenon studied. In phenomenographic research this is often referred to as the *direct object*. The *how*-aspect refers to the learners' approach to his or her task, or *how* the learning is accomplished.

Marton and Tsui write about the analytical separation:

> The learners' focus is normally on what they are trying to learn (the direct object of learning), whereas the teacher's focus should be on both; not only on that which the learners are trying to learn, but also on the way in which the learners are trying to master what they are trying to learn. (Marton and Tsui, 2004, p. 4)

The *how*-aspect can be further analysed into an *act of learning* and an *indirect object*. The latter is, according to Berglund (2005), often referred to as the learners' motives to learn. Berglund describes the former aspect:

> The term "act" should here be interpreted in a broad sense, beyond the physical acts that a student performs in order to learn, such as reading a book, solving a problem and asking a friend. The term "act of learning" also includes abstract aspects, such as how students go about achieving their aims. (Berglund, 2005, p. 42)

The different aspects of the experience the model-based analysis gives, bear useful and educational critical information to the researcher. It is still important to hold in mind what Berglund writes about the two aspects, the "what" and "how":

> [...] it must be remembered that the students experience the learning as a whole. The distinction is entirely analytical – the two aspects can

> only be thought apart – and aims to be a tool for the researcher in his efforts to understand, analyse and describe the students' learning. (Berglund, 2005, p. 40)

2.2 Phenomenography and learning

According to the phenomenographic tradition, the learning process is about experiencing, or seeing something in a new or different way, to open up aspects previously taken for granted or invisible for the learner. Marton and Tsui (2004) write:

> [...] the way that something is seen or experienced is a fundamental feature of learning. If we want learners to develop certain capabilities, we must make it possible for them to develop a certain way of seeing or experiencing. (Marton and Tsui, 2004, p. 8)

Marton and Tsui continue to discuss what it takes to "develop learner's eyes". Human beings have limited capacity to process information. We can only discern certain aspects of a phenomenon simultaneously. Different ways to see something means to discern partly or wholly different aspects of that thing, or phenomenon. "A particular way of seeing something can be defined by the aspects discerned, that is, the critical features of what is seen." (Marton and Tsui, p. 9). It is important for a learner to be able to discern new critical features in the subject of learning. The authors continue: "we not only discern features, but also discern different qualities (i.e., values) in the relevant dimensions such as "blue", "ray of light", "very short", and so on." (Marton and Tsui, 2004, p. 11). Discerning a phenomenon thus includes discerning features, or aspects[2], with different values, together with the ability to discern the relation of parts within the experience of the whole phenomenon, and the whole from the context and how the whole relates to the context.

A specific aspect of a phenomenon 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 the aspects visible. When aspects of something are discerned, values in the corresponding dimensions of variation thus are experienced. Marton and Tsui give an example:

> In order to experience the object as a blue, cylindrical, ceramic mug, all these aspects must be discerned and related to potential dimensions of variation. And because these aspects are necessary for defining the object in question, they are also called its *critical features*. (Marton and Tsui, 2004, p. 15)

An example is presented below to illustrate this. How is a circle defined? One aspect of the experience of a circle is that is has a size.

[2]In this context, *feature* and *aspect* are used in the same way as *part*, that is, part of an experience of a certain phenomenon.

Figure 2: How is a circle defined?

To be able to discern *size* as an 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.

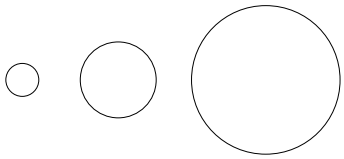

Figure 3: The aspect that a circle has a size is possible to discern when showing many circles with different sizes.

A variation in a dimension corresponding to the aspect that circles have sizes creates the opportunity for the person to focus on this aspect, and to discern it. This opens for the possibility to learn - a new way of seeing is opened.

Marton and Tsui have identified patterns of variation in learning situations:

1. *Contrast.* In order to experience something, then something to compare with must be experienced.

2. *Generalization.* Variation of values of the aspect is necessary to discern the aspect.

3. *Separation.* To be able to experience an aspect and to be able to separate the aspect from other aspects, it must vary while other aspects remain invariant.

4. *Fusion.* Several critical aspects need often to be experienced at the same time in everyday life. Separating the aspects first and then fusing them together is efficient for the learning. "[T]his fusion will unavoidably take place through the simultaneous variation in the dimensions of variation corresponding to the critical aspects." (Marton and Tsui, 2004, p. 17)

For a further analysis of the experience of learning, and in order to identify the variation necessary for learning, an extension of the theoretical model presented in Figure 1 is presented by Marton and Booth (2005), see Figure 4. The *what*-aspect in the model can be further analysed into

two aspects. The focus of the awareness, its parts and their relationships, and its surroundings, is called *the structural aspect*. Since the focus has had such direction, a certain meaning is discerned. This meaning is called *the referential aspect*. A further distinction of the structural aspect of an experience is made into the *internal horizon*, the aspects that are in focus of the awareness, and the *external horizon*, the aspects "which surrounds the phenomenon and to which it is related and of which it is a part" (Berglund, 2005, p. 41). The model is illustrated in Figure 4.

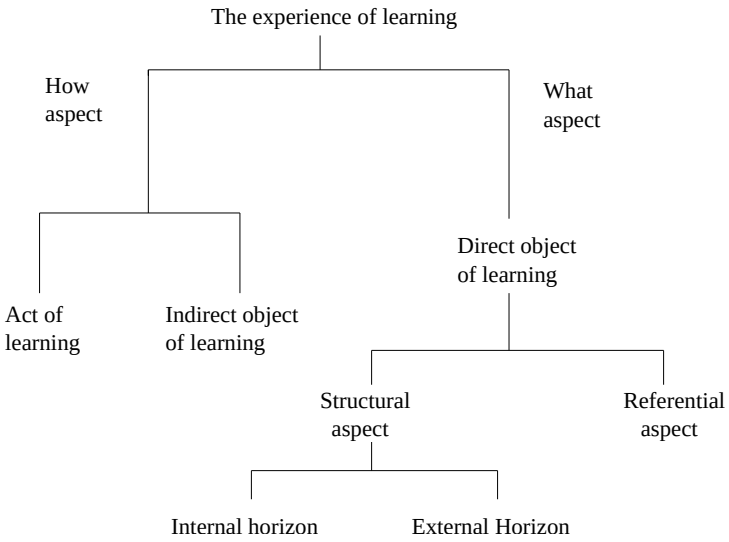

Figure 4: The experience of learning, including the referential and structural aspects with its' internal and external horizons (Marton and Booth, 1997)

The structural aspect describes the focal awareness of the learner. Variation in critical aspects thus refers to the structural aspect.

The reason for choosing the theoretical framework to analyse understanding is thus twofold. The framework gives a tool to get an overall picture of an experience, with the what- and how-aspect. The what-aspect can be further analysed into referential and structural aspects. The structural aspect then provides a basis for the analysis to find the dimensions of variation, necessary for the learning process.

2.3 Data collection, analysis and trustworthiness in phenomenographic studies

Phenomenography builds on an empirical, qualitative research tradition. Data gathering, analysis, the questions of validity, reliability and generalizability are inspired from this tradition, even if "these notions need to be reframed within the context of [...] the research approach" (Åkerlind, 2005, pp. 329-220).

In phenomenographic studies, data are often gathered in the form of interviews. Aiming at selecting a theoretical sample for the interviews, subjects are chosen with the aim to cover as broad range of relevant characteristics of the subjects as possible. Relevant characteristics could mean e.g. background knowledge in the subject area, sex and age. This strive for a broad representation, instead of finding what characterises an average subject, is fundamental for phenomenographic research. The pool of meaning gathered from the data aims at representing the whole group studied, through the selected subjects.

The interviews are transcribed verbatim. In this way data, in the form of text are analysed. The results, the different understandings found in the data, are presented in an outcome space. In phenomenographic analysis, the understandings are found when the data are read and reread and patterns of distinctly different understandings are looked for. Individual, decontextulised quotes illustrating certain understandings are compared with each other, grouped and regrouped, and eventually different categories of understanding emerge. The quotes are also read and reread in their own context to make subtle distinctions to the researcher's understanding of the data. The researcher formulates the essence of the understandings found with his or her own words in the categories of description. In this iterative analysis, by again and again going back to the data, the categories of description finally emerge.

Validity in qualitative research is, according to Åkerlind (2005) a question of "the extent to which a study is seen as investigating what it aimed to investigate, or the degree to which the research findings actually reflect the phenomenon being studied." In the phenomenographic tradition on the other hand, this is not so much the question . Åkerlind writes

> However, a phenomenographic researcher asks not how well their research outcomes correspond to the phenomenon as it exists in 'reality', but how well they correspond to human experience of the phenomenon. [...] the focus of research quality shifts to ensuring that the research aims are appropriately reflected in the research methods used (Åkerlind, 2005, p. 330)

According to Åkerlind, two types of validity checks are commonly used within phenomenographic research. *Communicative validity checks* includes the researcher's ability to argue for his or her interpretation of the data. It also includes "ensuring that the research methods and final interpretation are regarded as appropriate by the relevant research community." (Åkerlind, 2005, p. 330) The *pragmatic validity checks* on the other hand, have to do with whether the research outcomes are seen as useful and meaningful for the intended audience.

Åkerlind discusses, with reference to Kvale (1996) and Guba (1981) reliability in qualitative research in terms of "reflecting the use of appropriate methodological procedures for ensuring quality and consistency in data interpretation". With reference to Kvale (1996) Åkerlind describes two forms of reliability check[3] commonly used with qualitative, interview-based research such as in phenomenography:

1. *Coder reliability check*, where two researchers independently code all or a sample of interview transcripts and compare categorizations

2. *Dialogic reliability check*, where agreement between researchers is reached through discussion and mutual critique of the data and of each researcher's interpretive hypotheses.

Alternatively, Åkerlind discusses reliability in terms of researchers who "make their interpretive steps clear to readers by fully detailing the steps, and presenting examples that illustrate them."

There is however a discussion within the phenomenographic research community on how reliability of the results should be established, see e.g. Sandberg (1997). Sandberg discusses that coder reliability check can draw the attention from more fundamental checks of the research reliability, including a description of how the researchers "have adopted a critical attitude towards their own interpretations". (Åkerlind, 2005, p. 332).

The question of generalizability in qualitative studies has been discussed by e.g. Kvale (1996). Kvale points out three different ways of generalizability:

1. *Naturalistic generalization* rests on personal experience: It develops for the person as a function of experience; it derives from tacit knowledge of how things are and leads to expectations rather than formal predictions; it may become verbalized, thus passing from tacit knowledge to explicit propositional knowledge.

2. *Statistical generalization* is formal and explicit: It is based on subjects selected at random from a population [...]

3. *Analytical generalization* involves a reasoned judgment about the extent to which the findings from one study can be used as a guide to what might occur in another situation. Is is based on an analysis of the similarities and differences of the two situations. (Kvale, 1996, pp. 232-233)

[3]The word *check* is in this thesis interpreted not as proving something, but rather as one of several concerns of the issue of trustworthiness.

Of these methods, the naturalistic and analytical generalization seem to be most useful in phenomenographic studies, where a small sample of subjects are selected from a larger group with the intention to get a theoretical sample.

3 The empirical study

3.1 The course

The informants chosen for the present study are students from a degree course where programming knowledge is not a major goal. Programming courses are compulsory in most technical and natural 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 students had just finished their first programming course in Java, a compulsory course giving 4 Swedish credit points. (At Swedish universities one credit point represents one week's full-time study and 40 credit points one full academic year.) 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 highly qualified jobs after the education. One of the students in the study attended a degree course in Chemical Engineering.

A programming course for the present study was chosen where the author was not the teacher. In this decision, I followed the recommended rules of ethics, established by the Swedish Research Council and used at universities throughout Sweden (http://www.vr.se, 2003).

3.2 Data collection

The study took place at Uppsala University, Sweden in May 2002. A questionnaire was given to the student group. 22 of the 45 students answering the questionnaire were willing to participate in a one-hour tape-recorded interview. 14 students were selected with the intention to get a theoretical sample, see Section 2.3. The students participated voluntarily in the study, but were each given a movie ticket as symbolic remuneration.

3.3 The interviews

The interviews (see Appendix A) 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. If necessary, 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. On the other hand, the interview is neither as personal and emotional as in a therapeutic interview.

The present interview had four themes following the four research questions, see Section 1.1. The interviewer had prepared a small number of

questions on each theme, intended to approach the themes from different perspectives. In addition to the prepared questions, follow-up questions were given. These questions served as starting points for discussions to clarify students' statements, and for helping students to verbalise their experiences. The aim was to encourage the students to demonstrate as much as possible of their understandings and experiences within the themes.

The interviews were tape-recorded and transcribed verbatim to text files. In the quotes cited in the thesis, a pronounced pause has been denoted by three dots with no brackets round, ..., while three dots in square brackets denote that text has been cut away [...]. The latter applies also for quotations from books and other written material refered to. In some quotes expressions from the students are explicitly marked since the author found them relevant for the context. These expressions are put in parentheses, like (giggle) and (laughter). In the transcriptions of the interviews each student was given his or her letter of identity, A,B,C etc. with no connection to his or her real name, and the interviewer was labelled I.

3.4 The analysis

The research questions in the present study attempt to cover a broad spectrum of the students' experiences of learning object-oriented programming. Since the aim of my research is to understand more about these experiences, a phenomenographic research approach has been chosen. Learning experiences are complex and can be difficult to grasp and describe as a whole. The researcher can benefit from doing an analytical separation of aspects of the experience, as described by the students. The analytical separation of the experienced learning in *What*- and *How*-aspects has proved to be useful in this study.

The phenomenographic model for describing and analysing experiences of learning, see Figure 1 in Chapter 2, shows the theoretical framework for the analysis. This section aims at showing how the research questions posed in the thesis are in line with this model.

The main focus of the thesis, the learning outcome of a programming course, can be discussed in different ways. One way to discuss is to say that the learning outcome is correlated to

1. the understanding of what programming means
2. the understanding of concepts in the programming paradigm
3. the programming capability, or level of programming skill achieved

The first two items in the list correspond to the first two research questions investigated, *How do students understand what learning to program means?* and *How do students understand abstract concepts in object-oriented programming?*. The last item is not within the scope of this thesis. With

the focus to investigate learning outcomes of the course, two phenomena are studied, the first two research questions. These two questions are analysed with a phenomenographic approach. The results of the analysis of these questions constitute the *What*-aspect of the phenomenon investigated, see Figure 1. The two direct objects investigated are thus *understanding of what programming means*, and *understanding of central concepts* in the course.

Students' understanding of the computer and other central resources is another aspect of the learning outcome. I have, however, decided to look at the role of the resources from another angle, how the students' have approached the learning by means of the resources. The questions of students' use of resources thus belong to the *How* aspect of the phenomenographic model, see Figure 1. The question *How do students use resources and experience support of such in the learning?* is a part of the Act of Learning in the model. I do not claim that students' experience of the resources cover all aspects of the Act of Learning. This would demand a much larger study of the students' whole learning environment (Entwistle, 2003), which is beyond the scope of this study. Still the questions of students' use of resources is an interesting part of the Act of Learning, and few studies have been found that investigate more than one or a few resources students use.

Finally, the Indirect Object of Learning as the second part of the *how* aspect, corresponds to the question *What motives to learn computer programming can be found?*. Berglund (2005) writes "[t]he *motive* is frequently referred to as the *indirect object of learning* in phenomenographic research." This thesis presents a limited analyses of the Indirect Object in the sense that only positive motives to learn to program are presented. The reason for this is a decision to make the chapter of students' motive to learn mainly a discussion on implications for teaching. I plan to investigate students' motive to learn further in later studies.

As discussed above, the phenomenographic model is a theoretical tool to analyse the complex picture of students' experience of learning. This thesis aims at using this tool to make this picture more accessible for educators and contribute to better understanding of how students experience their learning situation.

3.5 Reliability, validity, and generalizability

In the present study reliability checks have been performed, following Kvale's suggestions (Kvale, 1996). Two of the research questions were analysed in the phenomenographic tradition. In the first phenomenographic analysis two researchers independently read all data and made preliminary categories before meeting and discussing. The results from the two researchers were very similar, and the final categories were easily agreed upon.

In the second analysis one researcher read the data and made preliminary categories. A second researcher checked the categories by matching chosen

quotes from the students to the categories. In a discussion between the researchers on the quotes and categories, the categories were adjusted and agreed upon.

For the validity check, communicative validity checks (Åkerlind, 2005) have been used and the content of the thesis has been discussed with education researchers. These discussions on some of the content have included phenomenographic and computer science education research conferences, some of which has been published in conference proceedings after peer review. In this way the relevance of the study has been scrutinised both concerning methods used and the relevance of the results in the research community. Furthermore, the question discussed by Åkerlind in terms of researchers "make their interpretive steps clear to readers by fully detailing the steps, and presenting examples that illustrate them" is considered in the thesis. I have interpreted this as giving a detailed description of data gathering and analyses, and a detailed description of data in order to make it possible for the reader to judge the outcome space.

As discussed in Section 2.3 naturalistic and analytical generalization are close to how generalizability is discussed in this study. My intention is to present as carefully as possible the group of students interviewed and the course the students have taken. There are advantages if the reader knows the subject area, object-oriented programming, and has some experience in teaching. Since phenomenographic research in general requires subject knowledge of the researcher this affects the results, and thus the reader's ability to judge to which extent the results are generalizable. I have nevertheless added a section describing the specific knowledge of the subject, required for judging the results and this is presented in Chapter 5.

3.6 Interview technique, some examples

Some examples from the interviews are presented below to illustrate how a semi-structured interview technique can elicit students' understanding by coming back to the same questions over and over again, but from slightly different angles.

An example is student G, who 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, have expressed 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. 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; this is something which is not possible to know. Consequently, the analysis has its focus on the variation of the understanding within the *collective*, not on individuals.

4 What does it mean to learn to program?[4]

4.1 Introduction

The main focus of the study presented in this thesis is on the learning and the learning outcomes of the course studied. The research question in this chapter

- How do students understand what it means to learn to program?

corresponds to one aspect of this focus. The interview questions on this theme serve as a background for a discussion on students' understanding of concepts and their use of resources. My data show that a general understanding of what learning to program means is important for the learning of the subject.

A phenomenographic analysis was performed. The students' answers in the interviews reveal five qualitatively different ways to understand what it means to learn to program, where some are more valuable for learning than others. In this chapter the phenomenographic analysis is described, and the results presented as an outcome space. The analysis is taken further by comparing with research in Mathematics education and this is followed by a discussion on implications for education.

4.2 Phenomenographic analysis

This chapter aims at shedding light upon the students' understanding of what it means to learn to program. The primary interview question on this theme is:

- What do you think learning means (involves) in this course?

Other questions in the study that appeared to shed additional light on the subject of interest were:

- What do you experience this course to be about?
- What has been most important to you in this course/Why has this course been good for you?
- What do you think was the aim for you when learning to program?
- What has been difficult in the course?

[4]This work has earlier been presented in a shorter version: What Does It Take to Learn 'Programming Thinking'?, Proceedings of the 2005 international workshop on Computing education research, (October 01 - 02, 2005) © ACM, 2005. http://doi.acm.org/10.1145/1089786.1089799

Students' answers to these questions were therefore added as a source of relevant information.

One researcher read and analysed the transcribed interviews, looking for qualitatively different ways to understand the phenomenon *what does it mean to learn to program* expressed in the data. For a reliability check, see Section 2.3, a second researcher studied quotes from the students and the categories identified by the first researcher. Five different ways to understand the phenomenon found in the data were agreed upon.

The different understandings, expressed as categories of description, are presented in Table 1. The categories are inclusive. This means that an understanding expressed in one of the later categories includes the understanding expressed in the former categories. The categories are furthermore hierarchical in the sense that the new understandings expressed in the later categories are more advanced.

Five qualitatively different ways to understand what it means to learn to program were discerned. Three of these are directed towards the computer, the programming language, and programming in general, while two are directed outwards, towards society with its programmed artefacts and the world of the programmer. The five categories are described in Table 1. Each category is described and illustrated with excerpts from interviews.

1. Learning to program is experienced as to understand some programming language, and to use it for writing program texts.
2. As above, and in addition learning to program is experienced as learning a way of thinking, which is experienced to be difficult to capture, and which is understood to be aligned with the programming language.
3. As above, and in addition learning to program is experienced as to gain understanding of computer programs as they appear in everyday life.
4. As above, with the difference that learning to program is experienced as learning a way of thinking which enables problem solving, and which is experienced as a "method" of thinking.
5. As above, and in addition learning to program is experienced as learning a skill that can be used outside the programming course.

Table 1: Categories describing the different ways to understand the phenomenon *What does it mean to learn to program?*.

All students in the study expressed an understanding that can be described as in category 1 in Table 1. Many students also expressed the understanding described in category 2. Fewer students expressed the understandings described in the last three categories.

4.2.1 Learning is to understand some programming language, and to use it for writing program texts

The first category summarizes an understanding that is directed towards the programming language itself, to understand it and to be able to use it. It is commonly expressed in such a way that students describe that learning of syntax details gives the feeling of knowing how to program. Other students focus on the ability to write short pieces of programs, 'program chunks', as characterising what learning means. To sit by yourself and code is desirable and appreciated. The skill to code in Java and to remember details in the language summarize this understanding and is presupposed in the other understandings the students express.

Student N emphasizes the importance of detailed knowledge of the syntax, and learning by heart. Answering the question what it means to learn in this course, student N answers:

> N: (giggle) Yes, but to learn must mean to understand and... But it doesn't mean that, because we have done the mandatory assignments in pairs, so it doesn't mean to sit beside and look when the other person does it, of course. Yeh, but to pick up what it's about, to understand what it's about and hopefully remember something.
> I: What is it about then?
> N: Well (giggle), don't know... difficult to say.
> I: [...] what is your opinion of what it is all about? [...]
> N: What it is all about, I think it is all about learning, partly the commands, fundamental commands I use, I have to remember them [...]

Student D expresses an understanding that is directed toward the language. He/she is talking about being able to read and find errors in the code. Student D answers the question what it means to learn in this course:

> D: Yes, it's probably to understand the language of the program. That is for example to see a program and see that, okay, this will happen and this is what the computer will do, this will be performed. And then also to see what's wrong in the language, to discover errors when you program and to see that this will not work because this can't be written like that.

4.2.2 Learning a way of thinking, which is experienced as difficult to capture, and which is understood to be aligned with the programming language

A common way to express what it means to learn to program, or what is missing in their understanding of programming, can be described as 'programming thinking'. Half of the students in the study talk about the actual thinking behind programming as something specific, an ability one has to acquire to be able to program. Many of these students seem to have problems

identifying what 'programming thinking' involves. Some express themselves as if it is something magic, difficult to catch.

Category two in Table 1, *Learning to program is experienced as learning a way of thinking, which is experienced as difficult to capture, and which is understood to be aligned with the programming language*, summarizes this understanding. This second category includes the first one, *Learning to program is experienced as to understand some programming language, and to use it*, but is more developed. An example that illustrates that the first conception is included in the second is when student D says: "you're supposed to get an understanding of the actual thinking when you program." The student discusses programming, but the focus is on the special thinking that is required. The understanding is, like in the first category directed toward the programming language but also toward the logic and thoughts behind the language. In this spirit some students discuss the differences between human beings and computers as something crucial to grasp in the learning of programming.

Student C talks about what is most important in the course:

> C: ...it's probably the way of thinking, that is when you program, how you are supposed to think and computer code and how it is interpreted, that's the difference to how human beings think.

Some students use the word 'logic' when they discuss how to think when they learn to program. This 'logic' is discussed by student A when he/she is asked what is most important in the course:

> A: It is the understanding of how the programming language is built rather than the specific command, if you want to do this, it's more the thinking itself, the logical thinking. Everything you need to know you must think of when it comes to programming. It's kind of, yes, it's very exclusive, everything is simply very detailed and you've kind of got a small insight into what it's like to program and how the computer works like that, or the software.

Student A articulates that the problems with the logic are the precise demands of the syntax of the programming language. Student A also connects this special thinking to how the computer itself works, not only the features of the programming language.

Student E expresses 'programming thinking' as different choices to reach a specific goal. He/she answers the question what it means to learn in this course:

> E: Well it's like thinking programming I think. Understanding things, putting together and how you make things work sort of and accomplishing what you want yourself. There are also many roads to take yourself to the goal.

Student D describes that this special way of thinking makes it difficult to know how to construct a program, to understand concepts, and that it also causes problems in knowing how to go about studying. Student D talks about programming thinking in a way reminiscent of magic. On the question what has been difficult in the course student D says:

> D: Yes, I think it has been difficult with concepts and stuff, as to understand how to use different, how one should use different things in a program. And I actually think that most of it has been difficult, but this very thought behind, it feels as some people just understand programming, it's something they... but I also think that some people who have been programming before have probably learned to think like that. But I still think the course, it's difficult for a novice to sort of get a grip of how to study when you implement the programs and like that. (Giggle)...

Ben-Ari's (1998) discusses problems computer science students may have if they lack an effective model of a computer. They may believe that there is some 'hidden mind' within the programming language that possesses intelligence. The problems in the process to discern the differences between human logic and capabilities, and the computer with its compiler's way of 'thinking' may be one reason for the students' strong emphasis on and struggling with this special 'programming thinking'.

4.2.3 Learning is to gain understanding of computer programs as they appear in everyday life

Some students talk about the programming they come across in the everyday life. Category three in Table 1, *Learning to program is experienced as to gain understanding of computer programs as they appear in everyday life*, summarizes this understanding. The understanding in category three is close to the understanding in category one in the respect that students mention it in connection to the ability to understand computer programs in general. This is expressed by student B when answering the question what the course has been about:

> B: It has mostly been about learning, for me it has been about how a lot of things are built up. That is to say that I've understood a lot of things, how banc programs works and such. I knew absolutely nothing before, so to speak.
> I: Exactly programming, is that what you mean?
> B: Yes exactly, yes. How, only such simple things that a question comes up on the screen and then I'm supposed to answer or something like that. I knew on a rather low level so to speak. But, I don't know what the course has been about exactly. I guess it's been about learning to understand and learning programming.

Student D answers the question what was most important in the course: (Förkortat i eng. översättningen)

> D: [...] You just think of things like when you withdraw money from a cash point, kind of, then you start to think, okay, it's these steps, figures and the sum and kind of... if there is money in the account and so on. No but those things that one starts to think a little about how certain things are built and exactly, yes, such things as when you're going to withdraw money or different games or such.

Student C answers the same question:

> C: Yes, no I don't know. It probably will be useful perhaps now and then or the understanding of how devices work in general. And machines. [...] No but there are many things that are run by computers today undeniably so that, it's some kind of understanding how things work. It's in cars, computers, lifts and everything. So that, yes, no, a good overview.

Student N discusses the goal and motive for learning to program:

> I: What did you think was your goal when learning to program?
> N: For doing it at all? (laughter) It's probably because it's in the program. But also, of course, it's rather ... I can't imagine that I will be programming in the future really but surly it's good to have seen it and tried it a little bit actually.
> I: Why do you think so?
> N: I don't know, but there are computers everywhere, aren't there and everything that's in the computers is programmed, isn't it, so just to have looked at it a bit I think can be rather important anyway.

The quotes from the students above express a vague and shallow understanding of programming as something they meet in everyday life that *might* be of some use, because of the wide spread of computer programs. The third category includes the first two categories in Table 1 because it discusses computer programs and the thinking when building programs. Despite the superficialness of the understanding expressed in category 3, it bridges to the last two categories found in the data when reaching out beyond the programming language and the course itself. The last two categories in Table 1 express understandings that are richer than the understandings expressed in the first three categories.

4.2.4 Learning a way of thinking, which enables problem solving, and which is experienced as a "method" of thinking

The understanding expressed in category four in Table 1, *Learning a way of thinking, which enables problem solving, and which is experienced as a "method" of thinking*, talks about learning to program in terms of problem

solving. It is closely related to the understandings expressed in category one and two. Programming knowledge is more or less presupposed, and the discussions on the 'programming thinking' are connected either to the course and course context, or to a need not limited by the course itself with its specific language learned. By taking the discussion outside the course context the understanding expressed in category four reaches beyond the first two categories and includes and builds upon the third category which discusses programming as it is met in everyday life. 'Problem solving' is discussed as ability useful within the course.

Student G discusses what it means to learn in the present course:

> G: To get to try, like, you learn to think in a special way, you learn problem solving. [...] It's problem solving. With the mandatory assignments, that is the difficult part, this you can say at least I think so.

Notice that student G mentions problem solving at the same time as he/she talks about learning a certain way to think. Problem solving is seen as part of 'programming thinking'.

Student K discusses problem solving as an ability separated from the programming language learned in the course. When answering the question what it means to learn in the present course student K says:

> K: [...] You know, it's good to have this kind of courses because you get to kind of exercise problem solving. That's actually really good. You have a problem that you solve in different ways and then you perhaps find the best way. That's one of the central parts I think. Then that you must write in some programming language, that you can perhaps do in any language. But exactly the problem solving, the way to handle problem solving, that's what I important think is important.

Student C answers the same question. He/she focuses on problem solving as meaning certain types of problems appearing in the course. Student C also discusses problem solving as an ability which might be useful after the present course:

> C: I don't know... I guess it's actually to solve a certain type of problem, it's rather like the math courses. Then learning different methods to solve them in different ways. Much like that, if you look back at the course it's not much actually but very, very fundamental. So to... get an overview and a basic idea of what it's about and that you can read on your own whenever you need.

4.2.5 Learning is a skill that can be used outside the programming course

The last category in Table 1 is *Learning to program is experienced as a skill to use outside the programming course.* This understanding presupposes

the understandings in the previous categories. The ability to know and use a programming language, as expressed in category one, two and four, are clearly expressed, but no longer the focus. The focus is moved outside the course and course context. The purpose of learning to program is not vaguely expressed as in category three. The students can clearly discuss why they want to learn to program and how they will use this knowledge after the course. Whilst this understanding is expressed in different ways by different students, what is common in the students' expressions is that the knowledge acquired is seen as something the student believe will prove useful later on, in further studies or in working life. Programming is experienced as a tool that will be beneficial for the student even after the study course.

Student C focuses on the use of Java knowledge when learning other programming languages. Student C answers the question what he/she thinks the course is about:

> C: [...] But it feels as if you get a better grip on most languages, if you want to study C it will easier after this course.

When answering the question what it means to learn in this course, student C discusses this in terms of reaching a level of knowledge in programming where you know enough to manage on your own. Student C obviously strives to come to an independent level of knowledge so that he/she can master situations involving programming in the future.

> C: [...] get a flair for and have some idea of what it's all about and that you then can read on your own when you need to. [...] even if I don't know how to do I can look it up, some examples, study the method and than presumably be able to write the code. That's probably what the course has laid the foundation for so one can reach that stage.

Student E also emphasizes the importance of independence. Knowledge is clearly described as a tool for his/her own success, to be used to manage the working life better. Student E answers the question what he/she thinks the course is about:

> E: [...] I guess, it's ... learning to think like a programmer

Later in the interview:

> I: What's the point of learning to program [...]?
> E: Yes but it's that the more you know about computers the less dependent on others you'll be, sort of.
> I: I see.
> E: I don't know, if you work somewhere later and have some insight into things, then I think it'll open a window so that you know what it's about at least even if you don't, I mean, it's the pros that will deal with the real things.

Student F and student H both discuss programming knowledge as something useful outside the course itself. Student F and H discuss the usefulness of programming knowledge when working with computers in general, or as a general knowledge of what programming is. Student F answers the question what is the point to learn to program:

> F: It's a fairly good aid when one shall do things, other things. Complement with computer programs, build programs, add information or tasks or calculations, than it can be good to know.

Student H answers the question what has been most important with the course:

> H: But I don't know, never programmed. Are you a bit into it you kind of know what one does. You hear of programming all the time, but what are they really doing? So I don't think it's the actual knowledge to know, ok, it's this that is programming. So I think it's been, because I don't know if I'll ever use it at all in the future. It might happen that I'll do that but even if I'll wouldn't I still know. Yes but when they talk about it on TV or friends doing it talk about it, then you at least know, okay, that's what they are doing. That's been a great advantages I think. Then, of course, I guess you can program some smaller easy program. Sometimes they demand that when you apply for a job maybe.

Students who express an understanding belonging to category five, have managed to place the course and the course context in their own world and thinking about their future. Learning to program is experienced as meaningful for themselves, even though the reasons for this vary.

4.3 Discussion on students' understanding of what it means to learn to program

Table 1 presents the results of the study as qualitatively different understandings of what it means to learn to program. The most crucial step seems to be from the second and third categories, *Learning is a way of thinking, which is experienced as difficult to capture, and which is understood to be aligned with the programming language* and *Learning is to gain understanding of computer programs as they appear in everyday life*, to the fourth category, *Learning is a way of thinking, which enables problem solving, and which is experienced as a "method" of thinking.* In the understanding described in category two, the students have noticed that a special way of thinking is required, but not necessarily what that is. As discussed in Section 4.2.3, the understanding described in the third category is a shallow understanding, but it bridges to the understanding in the last two categories. In contrast, in the understanding described in category 4 the students have

realized that the special way of thinking has to do with problem solving and a systematic way of thinking. The interview excerpts indicate clearly that students who express an understanding corresponding to category two feel confused about programming.

Before continuing the discussing of results I want to point at the relation between the categories of understanding identified by us, and the discussion by Hazzan (2003) concerning 'process-object duality'. This duality goes back to work by Piaget, and was developed in mathematics education to discuss the idea of reducing abstraction. Hazzan discusses this, referring to Sfard (1991) in terms of a passage from the 'process conception' to the 'object conception' [5]:

> *Process* conception implies that one regards a mathematical concept "as a potential rather than an actual entity, which comes into existence upon request in a sequence of actions." (Sfard, 1991, p. 4). When one conceives of a mathematical notation as an *object*, this notation is captured as one "solid" entity. Thus, it is possible to examine it from various points of view, to analyze its properties and its relationships to other mathematical notations and to apply operations on it. (Hazzan, 2003, pp. 107 - 108)

She concludes that according to these theories, "when a mathematical concept is learned, its conception as a process precedes - and is less abstract than - its conception as an object". It is thus a natural process when learning abstract concepts to start at the 'process conception'. The learning, in terms of process-object duality assumes however a passage from 'process conception' to 'object conception'. This is the desirable development also when learning computer science, including object-oriented programming.

Hazzan speaks of 'canonical procedures'. These are ways for the students to reduce abstraction level when dealing with concepts in different subjects. She writes:

> A *canonical* procedure is a procedure that is more or less automatically triggered by a given problem. This can happen either because the procedure is naturally suggested by the nature of the problem, or because prior training has firmly linked this kind of problem with this procedure. The availability of a canonical procedure enables students to obtain a solution without worrying too much about the mathematical properties of the concepts involved. It seems that this technical work gives students the assurance of following a well-known, step-by-step procedure, where each step has a clear outcome. In contrast, relying on abstract reasoning, for example by exploring properties of concepts or by relying on theorems, may be shaky mental approach

[5] When referring to my results, I will use the term 'category', while when referring to Hazzan's research, I use the term 'conception' to be consistent with her original terminology.

> for the students. Using the process-object duality terminology we may say that solving a problem by relying on a canonical procedure is an expression of process conception of the concepts under discussion; solving a problem by analyzing the essence and properties of concepts is an expression of object conception of the concepts under discussion. (Hazzan, 2003, p. 108)

The present chapter has its focus on students' understanding of what it means to learn to program, and more precisely to learn object-oriented programming. The process-object duality is of immediate interest in a course where abstract concepts like *object* and *class* are introduced early in the teaching, and where the understanding of these and other object-oriented concepts are fundamental for the rest of the course and for the ability to learn to program. Programming is a skill, but requires also a deep understanding of abstract concepts. "[A]nalysing the essence and properties of central concepts" in object-oriented programming is very much in line with the analysis and design phase in a programming problem. Analysis and design are abstract skills that belong to an 'object conception' that requires a good understanding of central concepts and has proved to be difficult, even for students who are in the end of their computer science education (Eckerdal et al., 2006).

In object-oriented programming as in mathematics there are standard solutions to certain types of problem, 'canonical procedures' to learn and discover. They are used by experienced programmers, and necessary for the simplification and speed up of the work. Following Hazzans arguments, it is desirable to help the students to discern such procedures. In this discussion I want to compare the students' discussion on 'programming thinking' when learning object-oriented programming, with a discussion on 'canonical procedures'. Many students mentioned 'programming thinking' as something specific, different from other subjects they had studied. Student D is an example of this. He/she compares programming with other subjects studied.

> D: [...] I guess, it's just a rather different way of thinking.
> I: Now that you have taken this programming course could you put your finger on something you think is different than chemistry... or you must have taken math too I suppose.
> D: Sure, I've taken many math courses but math is kind of logical and you understand it but this is... no I don't know (laughter). No but I kind of think it's easier to study math. Then you often have something creative to base it on, or you don't, but you learn more methods and kind of, there is some theory behind. Here you feel as if you only learn a lot of examples. You know, we've gotten so many examples of everything, in some way it feels as if you don't understand the base from the beginning [...]

Compare this when student C discusses what it means to learn in the present course:

> C: I don't know... I guess it's actually to solve a certain type of problem, it's rather like the math courses. Then learning different methods to solve them in different ways. Much like that, if you look back at the course it's not much actually but very, very fundamental.

Student C, who expresses an understanding belonging to category four has, in contrast to student D obviously discerned 'canonical procedures', and has less problems in his/her learning. Student C seems to have reached the level of 'process conception', and thus reached further in his/her understanding.

Student D on the other hand has not even reached the level of 'process conception'. He/she explicitly finds it simpler to study mathematics because there they learn methods to use, which he/she has obviously not been given, or discerned in programming. Student D furthermore discusses how troublesome it is to know how to study programming. "But I still think the course, it's difficult too for a novice to get a good grip on how to study". This points to that student D is looking for 'canonical procedure' as a study technique, but has not found such to the extent he/she asks for. Student D also explicitly expresses that he/she finds it problematic to understand concepts within the subject and connects this to the ability to program "Yes, I think it has been difficult with concepts like that, as to understand how to use different, how one should use different things in a program. And I actually think that most of it has been difficult, but this very thought behind, it feels as some people just understand programming".

In the understanding described in category four the students have realized that it has to do with problem solving and a systematic way of thinking. Using Hazzan's terminology, category four corresponds to 'canonical procedures', important in the learning process to reach the desired 'object conception'. It is not until category four that the students express an understanding of programming in terms of methods to use. This is therefore an important stage to reach. From the educator's perspective, it is important to support students who have problem to reach 'object conception', to first discern this understanding that corresponds to a 'process conception'. According to Sfard, the 'process conception' is necessary for the more abstract 'object conception'.

Results reported in Chapter 5, Table 2 and 3 indicate that there are students in the study who have reached an object conception of the concepts *object* and *class*. Table 1 indicates that there are students who have reached a process conception in the learning of object oriented programming. What is more alarming is however, that the results show that some students do not even discern 'canonical procedures', and thus have not even reached an process conception. Although with different starting points Hazzan's

research and the results from my study point to the same problem, but my study indicates that in learning object-oriented programming there are students who do not even reach a level of 'process conception'. My main question is:

- *How can we help students to reach a level of 'object conception' in object-oriented programming?*

The present study indicates however that an earlier question educators need to ask is:

- *How can we help beginning programming students to discern 'canonical procedure' in the process of learning object-oriented programming?*

Both my results and Hazzan's emphasize that students might need 'canonical procedures' as a heave to reach the higher level of abstraction, the 'object conception'.

4.4 Related work

The question on how students understand programming has been investigated and reported in other studies. To broaden the base for the conclusions drawn from the present study, I compare my results to those of two other studies.

In her thesis Booth (1992) addressed the question *What does it mean and what does it take to learn to program?* Undergraduate Computer Engineering and Computer Science students from first and second year at Chalmers University of Technology were interviewed several times over one term. Recurrent themes of the interviews were, among others, the nature of programming, the nature of programming languages, the nature of learning to program and the nature of studying programming. In the present study, the prime interest is the results from the theme 'the nature of learning to program'. Booth identified four conceptions of learning to program in her study.

Bruce et al. (2004) investigated first year university students' early experiences of learning to program. The question addressed was: *What are the different ways in which foundation year students go about learning to program?* Five different conceptions of how students go about when learning to program were identified. Bruce's research question is slightly different from the research question in the present study. It focuses on revealing differences in how students go about when learning to program, while the question addressed here is how students understand what learning means in the context of the programming course.

Both Booth and Bruce et al. use a phenomenographic approach when analysing their data. The results are expressed in outcome spaces as categories of description. Booth's categories are similar to the categories in

Table 1, with the exception that Booth has no category comparable with my third category. Bruce's et al. categories are also comparable, expect the first one, *Following - where learning to program is experienced as 'getting through' the unit.* There are students in the present study who express these thoughts. The reason for not mentioning this is that the outcome space from the present study does not explore how the students went about learning, but how they understand what it means to learn to program. Student E is shown as a comparison how he or she expresses the aim when taking the course:

> I: What has been the most important goal for you when attending the course?
> E: Pass it.

Student H has similar aim:

> I: But is the goal, why should you learn the course?
> H: Well, the goal is I suppose, the goal I suppose is to pass the exam, I almost said. (Laughter)

Booth, like Bruce et al., has no category corresponding to my third category. The reason for this difference between the studies is not possible to say from the data given.

Results from the studies discussed give even more support to the importance for students to reach the understandings of what it means to learn to program expressed in the last two categories in Table 1.

Booth investigates what she calls 'Framework constituents of programming'. These are *Conceptions of the nature of programming*, *Conceptions of the nature of programming languages* and *Conceptions of learning to program*. In this comparison I have only looked at the *Conceptions of learning to program*. Booth asks whether these frameworks are relevant for learning technical constituents like *function, recursion* and *correctness*. Her answer is:

> the conceptions of these apparently peripheral phenomena act as a scaffold to support the meeting with new aspects of programming and lend meaning to the experience, thereby affording one conception or another of the technical phenomena involved. [...] Abstract conceptions of technical constructs demand a well-developed set of framework conceptions. [...] [T]he conceptions of the nature of programming itself, programming languages and learning to program were seen as providing some sort of framework for the whole enterprise of learning to program. (Booth, 1992, pp. 261-262)

A good understanding of the framework constituents of programming, where *Conceptions of learning to program* is one, scaffolds better understanding of programming concepts, which is important for a programmer.

Investigation of how the students in the present study understand central concepts in object-oriented programming will be presented in Chapter 5. This is comparable with the technical constituents Booth investigated in her study. The question of how the students understand what it means to learn to program is thus a relevant question to connect to the findings of the students understanding of the concepts. This is done in Chapter 8.1 in this thesis.

The conceptions identified in the present study are discussed in terms of a hierarchy. There seems to be a critical step between the first three categories and the last two. The last two categories include the former and are thus more advanced. Moreover, the data shows that the students who only express understandings corresponding to the first three categories are more confused than students who express the understandings in the last two categories. These latter categories also correspond to the latter categories in Booth's study, which Booth shows better scaffold the learning of technical constituents within programming.

Bruce's et al. study supports the same conclusions. The authors comment on the categories found:

> It would appear that problems are likely to occur when students don't move beyond the learning experiences of categories 1 or 2. Attempts to influence the act of learning, or how students go about learning, through, for instance, changes in curriculum or the development of teaching tools such as online tutorials, need to focus on encouraging students to experience the range of different ways of learning to program. (Bruce et al., 2004, p. 42)

The last categories in Bruce's et al. study correspond to the last two categories in Table 1. The conclusions from Bruce's et al. study thus support the importance of helping the students to reach an understanding of what it means to learn to program expressed in the last two categories in the present outcome space.

Booth also discusses students when they encounter a programming problem. This does not happen "in a vacuum but against a background of earlier experience. This experience is not only made up of the programming awareness, with its conceptual character, but also of the experience of how one writes programs and solves problems in general" (Booth, 1992, p. 264). In this context, I find arguments to investigate the environment the students meet in the study course in terms of resources students are offered and choose to use in the learning process.

In this sense I go further than the studies of Booth and Bruce et al. I do not only investigate the students' understanding of what it means to learn to program, but I also explore the same students' use of resources. I investigate students' experiences of the resources they use, how they use them and the ways they are used. This use can enhance the teaching and learning

environment such that it communicates and scaffolds an understanding of what it means to learn to program as expressed in the last two categories in Table 1.

5 On the understanding of Object and Class[6]

5.1 The object-oriented paradigm

Object oriented programming languages are used at university courses at all levels throughout the world (Roberts, 2004b). Much has been reported on the experiences of teaching the object oriented paradigm (Thomas et al., 2004; Mahmoud et al., 2004; Chen and Morris, 2005). Object oriented programming is experienced as difficult both to teach (Roberts, 2004a; Kölling, 1999a) and learn (Lahtinen et al., 2005). The object oriented paradigm is built on some fundamental abstract concepts, including *object* and *class*. This chapter focuses on students' understanding of these concepts, which, when they learn Java, are met at an early stage. The understanding of the concepts is thus important even in a beginning programming course. There are other concepts central within the object oriented programming like *encapsulation* or data hiding, *inheritance* and *polymorphism*. The reasons 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.

The studies already referred to in Section 1.4 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 chapter:

- 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 the group of students in the study presented. 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 object oriented programming.

5.1.1 Background

Programming is a central subject in all forms of computer science education and often also in other forms of engineering education. The programming

[6]This work has earlier been presented in a shorter version: Novice Java Programmers' Conceptions of "Object" and "Class", and Variation Theory, in inroads - SIGCSE Bulletin , VOL 37, ISSN 0097-8418, (Jun 27-29, 2005) © ACM, 2005. http://doi.acm.org/10.1145/1067445.1067473

paradigm taught has however changed over time. Currently the object oriented programming paradigm is common. Programming languages within the object oriented paradigm are for example 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, which is to say programs that are correct, efficient, reusable, extendible, easy to use, which are exactly features that underpinned 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 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 achieve 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 affect only 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, for example 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.

5.1.2 Central concepts: class and object

There exists an established understanding, widely shared among professional programmers and teachers, as to the meaning of the concepts of *object* and *class* (Meyer, 1988; Bar-David, 1993). When describing the concepts *object* and *class* a variety of aspects should be mentioned according to these authors:

- An object is the computer representation of some 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 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.

5.2 Phenomenographic analysis

The interviews were read and 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 followed for the concept *class*. The decontextualised 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 aim was to describe the variation of the understandings found in the group.

Using the terminology of the theoretical framework developed by Marton and Booth (1997), presented in Figure 4, Section 2.2, the understandings captured in the described analysis, represent the referential aspect of the What-aspect. The referential aspect represents the meaning experienced in the direct object, that is the concept studied. In the phenomenographic tradition this part of the analyses is commonly described as creating for "a pool of meaning" from the data.

When looking for categories of description found in the group, for a reliability check (see Section 2), 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.

5.2.1 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 2. 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 2: 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:

> C: 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:

> N: 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:

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

Student H says:

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

Student J says:

> J: 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, similar indications were found (Eckerdal, 2002). 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 (1999) writes about understandings which include 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.". It is thus of great importance that students reach an understanding of the concepts *object* and *class* that goes beyond the code level.

5.2.2 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 3 and illustrated by quotes below.

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

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:

> H: 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. Student E emphasizes the module aspect:

> 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.

Student C discusses the module aspect in terms of encapsulation:

> 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.

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 are 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:

> L: 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 O says:

> 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 [...]

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:

> 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: 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 3, *"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 3.

5.2.3 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 4.

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

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 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 4, *"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 of the program. The most usual way to express this is to say that objects and classes simplify the work for the programmer for example 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 user's 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 4, *"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 2 *"Object is experienced as a piece of code"* and the first category in Table 3 *"Class is experienced as an entity in the program"*. The third categories in the Tables

2, 3 and 4 all discuss the reality aspect. Even the second category in Table 4: *"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 2: *"Object is experienced as something that is active in the program"* and the second category in Table 3 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 4, *"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.

5.2.4 Discussion on 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 5.1 an object is an instance of a class, and a class is the description of a phenomenon in 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, it is not surprising 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 4, he or she also expresses an understanding of *class* corresponding to the second category in Table 3. 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 2 and Table 3, respectively, are merged in Table 5 Table 5 thus include both the students' understanding of *object* and *class*. When comparing the outcome space in Table 2 and in Table 3 with the professional way to understand *objects* and *classes* described in Section 5.1, it can also be noticed that the main ideas behind the concepts are covered in the outcome spaces.

In Table 5, as in the previous tables, the categories are intended to be inclusive. This means that an understanding expressed in one of the latter

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.
As above, and in addition 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.
As above, and in addition 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 5: 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.

categories includes the understandings expressed in the former categories. It is hardly possible to understand that an object is a model of something in 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 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. (Marton and Booth, 1997, p. 107)

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 (1999), who 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 there are today researchers who 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 Booth (1992) . Students do, depending on the context, express different

understandings of a phenomenon. Booth claims that in Computer Science a good understanding of a phenomenon is reflected in the capacity to choose in a situational 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 5 are thus important and valid in different situations when working with a programming task. By the programming task I 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 5. 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 Booth's (1992) arguments stressing the importance of the ability to shift between the different categories , and Pong's discussion about the "multi conceptions" perspective, that different aspects of a certain phenomenon can be focused on depending on the context.

5.3 Enhancing the learning process

What can the educators do to facilitate for the students to develop their conceptual understanding? The following sections discuss how the empirical results in this chapter can be further analysed and give implications for teaching.

5.3.1 Learning in a context

Previous sections have discussed the different meanings of the concepts *object* and *class*. Table 4 however describes the *purpose* of using objects and classes. It turns out that these points to factors that might be important for teaching. The students understanding expressed in the first part of the first category in Table 4, *"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 pro-

gramming 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 problem to see 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 4, *"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 4 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. He 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 (Hadjurrouit, 1998)

Here he mentions "The concept of object for designing software as a set of interacting objects." Later he 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 (1999) 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.". 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.

Computing Curriculum 2001 (Joint Task Force on Computing 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 investigation in later studies.

5.3.2 Identification of critical aspects

The analytical framework developed by Marton and Booth (1997) presented in Figure 4, Section 2.2, has been applied to the results presented in Table 5. The reason for this is twofold. The framework gives a tool to get an overall picture of the students' experiences, with referential and structural aspects. The structural aspects then provides a basis for the analysis to

identify critical aspects of the concepts studied, and thus discuss dimensions of variation, necessary for the learning such critical aspects.

Three qualitatively different categories of understanding have been identified in Table 5, each category of understanding expressed in terms of its referential aspect. Intending to discuss implications for teaching, the structural aspects of *class* and *object*, corresponding to each of the identified categories of understanding, will now be made explicit. An example of this is the first category in Table 5, 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.

As discussed in Section 2.2, learning requires discernment of a new structural aspect of the phenomenon, and discernment requires variation in a dimension corresponding to the structural aspect.

In the first category in Table 5, 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[7]. 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 5, it is now possible to discuss what dimensions of variation correspond to each category. In the first category in Table 5 the structural aspect is the 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

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

textual representation of programs constitutes a dimension of relevance for the understanding of *object* and *class*. Different, specific program texts constitute 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 5.

In the last category in Table 5, 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 6. It includes both the referential aspects of the concepts *object* and *class*, see the left hand column in Table 6 and the structural aspects, see the mid column. In the right hand column I have included what I conclude to be the corresponding dimensions of variation.

5.3.3 Implications for education

Table 6 has implications for teaching. Teaching is here defined in a wide sense, to mean everything that supplies resources for learning. Examples of such resources could be programming assignments, 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 6. The teacher can create the conditions for such discernment with the judicious use of variation, see Section 2.2. By varying the teaching and holding the critical aspect of the phenomenon invariant, that critical aspect, in contrast to the surrounding, is lifted out of the surrounding "noise". Furthermore, generalisation, expressed as variation of values of the critical aspect, can open possiblities for discernment. I speak of opening dimensions of variation, in which taken-for-granted ways of understanding are now brought into focus.

The results in Table 6, 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

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.	Focus on the program text and the syntax rules of the programming language.	*Variation of the text in the program.*
Class is experienced as an entity of the program, contributing to the structure of the code and describing the object.	Focus on the structure of 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 6: Categories describing the different understandings and the corresponding aspects of variation of the phenomenon *object* and *class* found in the group.

and learning resources. The following paragraph discusses possible ways to achieve this, showing a few examples.

An example of 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 that different programs represent classes and objects differently, at 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 are 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[8] (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 6 includes the aspect that objects and classes model the real world. The present 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 may in a next step 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 discuss in groups their different solutions and how their final solutions 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.

[8] UML (Unified Modeling Language) is a visual language. It is a standard for modelling, developing and documenting object-oriented computer systems.

My results can shed new light upon and give explanation to other research and discussions in the field.

For example, the importance of letting the students follow a whole programming task, not only the coding is commented in Computer Curriculum 2001 Section 7.2 (Joint Task Force on Computing Curricula, 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. My analysis above provides empirical and theoretical support for this statement.

As an second example, Holland, Griffiths and Woodman (1997) list some misconceptions noticed at distance courses where Smalltalk was taught, in one introductory undergraduate course, and one postgraduate course. One misconception mentioned is "object as a kind of variable". If the examples the students first come across have only one instance variable, students with previous experience of procedural programming may 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 a conception according to the second categories in Table 2 and Table 3. The second category in Table 2 emphasizes that objects are active during execution of the program. This points to the behavioural aspect of objects. The second category in Table 3 explains classes as a description of both data about the object, and methods explaining the behaviour of the object. As explained in Section 5.3.2, 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 in line with 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 6. As explained in Section 5.3.2, 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 (see Chapter 2).

As a third example, Holmboe (1999) performed a study where people of different background, 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, were asked to describe in their own words what object-oriented programming is. He made a qualitative analysis of the answers, and his comments to his research can be illustrated by my results. When analysing the results from the study he concludes that some types of knowledge are more suitable as basis for further knowledge construction than others. He writes about understandings which include 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 2 and Table 3 capture an understanding of classes and objects that includes the world outside the computer itself. The dimensions of variation found and discussed in Section 5.3.2 are valuable knowledge for teachers to facilitate for the students to reach this understanding.

Lastly, in Fleury's (2000) study of student's constructed rules, she stresses the importance of carefully constructed sample programs to avoid misconceptions of concepts. My study stresses the importance of designing the education so that the students can discern the critical aspects of the concepts. 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. My 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.

6 Students' use of resources when learning to program

6.1 Background

Learning object-oriented programming involves learning central concepts in the programming paradigm, the thoughts that build the foundation of this particular problem solving strategy. Learning to program furthermore inevitably involves use of complex resources like computers, compilers and editors. These resources are not only tools in the learning process, but learning to master them is part of the subject itself. Following this reasoning one of the aims of this study is to give a coherent picture, from the students' perspective, of the roles of some frequently used resources[9] in an object-oriented programming course. This involves how the students use the resources, how they experience that the resources support them in the learning, and the interaction between resources. It was not decided in advance which resources to examine, but I let the answers from the students lead me.

The importance to learn how to use resources in a meaningful and efficient way is seen to be of decisive importance for learning to program, and problems related to how to use the resources are thus important. There were students in the study who expressed serious problems in learning the subject. They discussed this in terms of problems of understanding concepts, but also as an inability to know how to go about studying. When learning to program this is closely related to use of resources, since learning is mostly discussed as taking place in front of the computer, when the students write programs.

Thoughts in this direction have specially been developed in the sociocultural tradition. In this tradition, learning is viewed as taking place in a context where *tools* and complicated forms of cooperation between people play significant roles (Säljö, 2000, pp. 19 - 20). Learning does not occur in a vacuum, we observe and work on the world around us using tools, in cooperation with other human beings. Tools are the resources, both linguistic and intellectual as well as physical, which we have access to and use when we act in and understand the world around us. Discussing from this perspective, people act and think in the interplay between mental and physical resources, artefacts. We use resources to solve problem and master social practices. Thinking can thus not be studied as an isolated phenomenon. Säljö (2000, p. 76) argues, with reference to Dahlbom (1993), that the inability to integrate artefacts in our understanding of development and learning is the

[9] In this thesis the word *resources* is used in a wide sense. It includes everything the students mention they have used and experience as support in their learning. It corresponds well to Säljös discussion on artefacts as mental and physical resources (Säljö, 2000).

great weakness of the psychological and pedagogical sciences, which risk to make them abstract and out of touch with reality.

One of the goals of the present study is to identify resources used by the students in learning object-oriented programming. This chapter highlights the roles of resources mentioned by the students in the interviews, how these resources interacted with the students in their learning process. When analysing the students' answers on their use of resources, two different themes appeared, following the questions asked in the interviews. The two themes were:

- How were the different resources used by the students?
- How did the students experience that the different resources supported them in their learning?

The following sections will first describe resources mentioned in the present study. After that the research approach, the interviews and the analyses are described. Then, the two questions above will be discussed. Finally implications for education are formulated.

6.2 The resources

The answers did not differ much when the students discussed what they considered to be "resources used for the learning". Most of the resources mentioned are resources given by the educator, and explicitly pinpointed as recommended tools in the study course. All students mentioned the lectures and notes taken during the lectures but also the text book, even though the experience of how important the text book was and the use of it varied a lot among the students. All students, except one, mentioned the compulsory assignments and the computer itself as important resources. Friends and peer students were also mentioned by all students. A few students mentioned other resources, like 'patience', 'Internet' and 'development environments'. This thesis does not discuss the students' answers on their use of old exams and how they prepared themselves for the final written exam. These resources are planed to be discussed in forthcoming work.

6.3 Research approach: Content analysis

Students' use of resources in the learning process belongs to the *How*-aspect in the phenomenographic model presented in Section 2. In this model the use of resources constitutes the *act of learning*.

In this study, the data on students' use of resources have not been analysed phenomenographically, but by use of content analysis. The reason for not using phenomenographic analysis is that the research question does not concern students' experiences of a phenomenon, but rather their approaches

to the resources. The analysis performed is qualitative text analysis aiming at finding categories describing different approaches to the resources. Here, content analysis is used as a method for investigation. The phenomenographic research approach, presented in the model, represents a full view of how to understand students' experiences of learning. In this way content analysis can be used as method to explore some parts of that full view.

Content analysis is a wide term. Mayring (2000) discusses qualitative content analysis as an approach of systematic, qualitative text analysis. The material is analysed step by step, divided into analytical units. Categories created should be in the centre of the analysis. The research questions guide the text interpretation and lead to categories which are revised in feedback loops. The question of trustworthiness should be discussed in an adequate way.

The analyses of students' use of resources mainly followed the given description. The analyses were performed in a systematic way. All data was studied and categorised, according to the research questions. The systematic analysis were performed in "revised in feedback loops", by again and again going back to the data, and by refining the categories found. For reliability, validity and generalization, methods common in phenomenographic analysis have been used. One researcher analysed the data. The emerging categories were agreed upon together with another researcher, and in the study of parts of the data, see Section 2.3. Data from the study is richly presented in the thesis, the results are related to results from previous studies, and the reader is supposed to know the subject and thus to be able to recognise the results from his or her own experiences, see Kvale (1996).

6.4 The interviews

The questions the students were asked about their use of resources are first presented. They are important in the sense that they can be assumed to trigger the answers the students gave.

One initial question was asked. From the answer given by the student, the researcher listed the different resources mentioned on a paper in order to ask more questions on each of them later in the interview. The initial question was:

- There are many available resources and tools to use in this course. I'm interested in which resources and tools you have used and which you found helpful and meaningful, especially considering learning concepts like object and class?

A few more questions were asked to help the students to present as many as possible of the resources he or she has used.

- What do you do if you get stuck when studying or programming?

- Which resources were especially helpful when learning the concepts *object* and *class*?

For the resources listed, a few questions were asked to enlighten how the resources were used, the student's aim when using the resources, and how they supported the student in the learning process. Each individual question was not asked for each resource.

- How did this resource support you in your learning?
- What was your aim of using this resource? Why did you use this resource?
- How did you use the resource?

Following-up questions on the students' answers were asked to clarify the students' statements. The discussions following from the following-up questions were sometimes a considerable part of the whole interview.

6.5 The analysis

When analysing the students' answers to the questions above, two main themes appeared. The first theme was how the different resources were used by the students, and the second was how the students had experienced that the different resources supported them in their learning. The interviews were analysed in order to find qualitative and, if appropriate, quantitative differences with respect to the two themes.

In a first analysis all resources found in the interviews were put in separate columns in a spreadsheet, while the students' descriptions of how the resources were used were put in the rows. These descriptions were formulated in the researcher's own words. The student that had expressed a certain way to use a resource was reported with his or her letter of identity under the resource where the description was found. The second theme was analysed and reported in a similar way.

In a second stage of the analysis the spreadsheets were studied with the aim to find patterns and similarities in the data. Earlier studies were investigated, where similar questions were researched. Starting from the data, the findings could be connected to terminology already developed. The terminology served as a frame in the analysis, to structure the findings. In this way a content based analysis was performed. The first analysis gave two separate spreadsheets, corresponding to the two themes. Each theme and how the second phase of the analysis was performed for each of the themes, are described below.

The second stage of the analysis showed that the resources were used in qualitatively different ways. Two main approaches appeared, one interpreted as a search-for-meaning approach, and another interpreted as a

superficial approach to the use of resources. In all important aspects, these approaches correspond to the deep/surface dichotomy discussed in earlier research on students' approaches to learning. Marton and Säljö (1984, p. 44) discuss that the deep/surface dichotomy "emphasized *referential* aspects of students' experiences – their search for meaning or not" . The reason why I chose to use meaning/superficial terminology instead of deep/surface is to avoid confusion with how the concepts deep/surface *learning* are interpreted in some other contexts.

6.6 How the resources were used

Aiming at finding an overview in the analysis of the data, I looked for qualitative differences in the students' experiences of how they had used the resources, and how they experienced that the different use of resources influenced their learning. I grouped together and summarized with my own words similar ways of using resources. I found that these ways, or approaches, could be categorized into two main categories according to the search-for-meaning/superficial dichotomy. The approaches are all connected to the students' experiences of learning the course content, and I thus labelled the main categories *Superficial content related ways* which categorize two of the approaches found in the data, and *Search-for-meaning content related ways* with five of the approaches. The two categories with their belonging approaches are presented in Table 7. In the following sections the approaches are written in italics and illustrated with excerpts from the interviews.

Additional approaches were found, that are rather connected to the organisation of the course, than to learning of the course content. They describe how the students' go about studying. I call these approaches *Study organisation* and discuss them separately at the end of the section.

1. **Superficial content related ways.**
Looking for hints or pieces of code to copy
Writing without understanding in order not to fall behind
2. **Search-for-meaning content related ways.**
Read, write or listen aiming to understand
Ask questions, discuss problems
Explain to other students
Think before coding with the help of paper and pencil
Role play to get a better understanding
Learning by coding

Table 7: Students' different approaches to how they used the resources in their learning, written in italics, are grouped into two main categories.

Superficial content related ways

Only a few of the approaches found can be characterized as superficial.

- *looking for hints or pieces of code to copy*

Student H says about *peer students*:

> H: [...] sometimes you, well, just get the code directly and other times more like a hint, yeah, like, you might have a go at it a bit like this...

Student C discusses *program examples* handed out at the lectures:

> I: Well. But the examples handed out in the lectures, how did they help you to learn?
> C: That's nearly always just a matter of being able to copy the example code sections, that is one can see how, it was the stuff to do with graphics which one had no real clue about still, that's to say if you haven't written this stuff yourself it's pretty hard to see how they work and so forth, so it becomes very much a case of just directly copying the example.

The students express that they strive for copying fragments, or part of code, in contrast to understanding and doing the coding themselves. They seem to express an intention to reach a mechanical how-to-do level of their understanding.

- *writing without understanding in order not to fall behind*

This superficial approach is connected to taking *notes* during the lectures.

Student A says:

> A: [...] you write a lot just to keep up and write down everything, often you don't have time to think about it.

Search-for-meaning content related ways

A search-for-meaning approach to learning is characterised by "a search for the author's intention, relating the content to a larger context" (Marton and Säljö, 1984, p. 43). This is the most advanced approach to how the resources were used. The content of the course is in focus, and the resources are used with an aim to find a context and a whole, a coherent understanding. The approaches interpreted as search-for meaning approaches are presented below.

- *Read, write or listen aiming to understand*

This approach differs from the superficial *writing without understanding in order not to fall behind* in the sense that the students strive for a fuller and better understanding by use of several resources or by an extensive use of one resource. The students express endurance in their use of resources, where understanding and getting a larger context are the aims. It also differs significant from the superficial *looking for hints or pieces of code to copy* approach, where the procedure of copying is in focus, while striving for understanding is the focus of this approach.

Student O discussed the aim of attending the *lectures*. He or she listened at the lectures and took notes at the same time to facilitate the learning:

> O: [...] listen to it once too, explains... to write yourself. Then you learn in two ways, both listening and writing and then you have something to take home which you can read later.

Student A found the *textbook* important for learning. Student A read the textbook at one occasion to get an overall picture:

> A: [...] the real AHA experience, that was the text book. I sat down for a whole day and basically read all the sections we should read from cover to cover and I usually don't do that. But it connected the whole way, what I read here came back 20 pages later and, yes, it was built very pedagogically I think. Lots of repetition. That makes it easier to understand. So I thought it was great.

- *Ask questions, discuss problems*

Student C talks about that the purpose of using the *teacher* as a resource is to get a better grip on the context, in contrast to details and fragments of knowledge:

> C: to ask why it doesn't work, basically, and explain greater contexts.

Student K finds it helpful to ask *other students* when he or she doesn't understand. Student K expresses an endeavor that goes beyond a how-to-do approach found among the superficial approaches. Student K explicitly expresses that he or she prefers not copying but to construct his or her own knowledge:

> K: Yes, it's like in all subjects, if you have problem with something you can ask a friend and they might know it better exactly on that point. Show how one should do or say how one should do. So I think it's good to get a hint that, you could do like this instead. Then you can figure it out yourself.

- *Explain to other students*

A way to get a better understanding mentioned by the students is to explain to other students.

Student J uses *peer students* in this way:

> J: [...] you can learn pretty much by trying yourself to explain to someone.

- *Think before coding with the help of paper and pencil*

To use *pencil and paper* as a resource to reach a better understanding is a search-for-meaning approach.

Student O says:

> O: [...] I suppose you've learned most when you sat on your own and wrote at the computer or on paper. I sat and tried to write a bit on my own before I used the computer.
> I: Yes, what was your purpose then?
> O: Well, both that [the teacher] had mentioned at a lecture that it's difficult to go to the computer directly and know what to write, specially if you haven't done it before. So it was a hint from him so I tried to do it and I think that was good too. Then you got many things done when you had a lesson too, during lesson time. You did fairly much.

- *Role play to get a better understanding*

Role play at the lectures as a way to enhance the understanding is mentioned as search-for-meaning.

Student D says:

> I: You mentioned that he has been kind of visual or whatever it was, by playing games and such. How do you think that has helped?
> D: Yeah, anyway I think that has helped (laughter) better than if not having had it at all. Because otherwise it would just be this piece of theory and then you wouldn't have any practical example.
> I: What do you think they have given you?
> D: Something to refer to. Maybe you can think, okay, this is what he did at the lecture if you work with the method, and now you shall build something that functions like this as he showed in an example. I think so.

- *Learning by coding*

Many students emphasize the importance of their own work at the *computer*.

Student G expresses a search-for-meaning approach to using the *computer*. This approach is more than only copying code, which is described as a superficial approach. The student G uses the computer with endurance, as a tool to reach an understanding. He or she says:

> G: Yes, since I'm so dependent on sitting by myself in front of the computer to learn. It is then I learn. It's not for certain... some maybe... programming is such that one has to sit with it by oneself. It is not many that can attend lectures and then take the exam since it is a skill. One has to practice to learn.

Study organisation

Above, students' use of resources has been discussed with a focus on the learning. In the data ways to use resources that were more connected to studying than to learning were also found. Students discussed ways to go about studying in terms of course organisation. The subject content is not in focus in these descriptions.

- *Sitting on my own working*

Such a way to use resources found in my data it to work alone.

Student O discusses how he or she has worked with *the compulsory assignments*:

> O: It is a relief to be done with the course when it ends and partly in order to be allowed to sit at a computer and write oneself.

Student O has organized the work in the way that he or she sits at a computer on his or her own.

Earlier research that supports this finding is found in the ETL project (http://www.ed.ac.uk/etl). Five different approaches relating to students' learning and studying were identified. One of them was "organised studying" which is in line with the result presented.

- *As preparation for the final exam*

Student G has used *the textbook* only just before the final exam, in a way that made it possible to pass the exam:

> G: [...] I've hardly opened the text book until now before the exam. [...] and one is allowed to bring the text book to the exam. That is why I have used it. But apart from those things I've not used it at all.

Student M uses *paper and pencil* when he or she prepares for the final exam using old exams:

> M: [...]Then it becomes to try to solve the problems. [...] Old exam problems.
> I: How did you do that?
> M: With paper and pen [...]
> I: Okay. Have you worked alone or?
> M: Yes, now and then. A bit of both actually but a lot by myself also to be able to focus on all this, this is my problem to interpret written problems and get it over to computer symbols.

Student M seems to choose to use paper and pencil and work on his/her own since the final exam is performed individually, with paper and pencil.

6.7 How the resources were perceived to support learning

The analysis of how the students experienced that the resources had supported them in their learning was done in a similar way as described in Section 6.6. I looked for all different ways in which the students expressed that the resources had supported them, and described and summarised these ways with my own words. As in the previous analysis I found that the different ways, or approaches identified could also be categorised into two main groups as *Superficial support* and *Meaningful support*. The different approaches found are written in italics and categorized according to the superficial/search-for-meaning dichotomy in Table 8. Each approach is commented below and illustrated with excerpts from the interviews.

1. **Superficial support**
Fear of failure forces learning activity
Extrinsic rules as a support
2.**Meaningful support**
Subject interest as a support
Social stimulation as a support
Stimulation of personal activity

Table 8: Students' different approaches to how the resources were experienced as help in their learning are written in italics. The approaches are grouped into two main categories.

Superficial support

'Superficial supports' identified are closely related to superficial approaches to learning. They seem to have their origin in a feeling of compulsion, insecurity and fear.

- *Fear of failure forces learning activity*

Student J discusses the *compulsory assignments* as something that has to be done, but the aim is to pass even if he or she does not understand. The reason seems to be lack of time:

> I: What do you think is the goal, the aim, the most important for you when you work with assignments?
> J: Well, unfortunately when there is time pressure it feels like it is mostly about that one wants to be done with it and feel that one has done it, so in some cases it has maybe been, well we write this and it works, but one doesn't really understand what it was, unfortunately it is so. Because one, we have not had time to look into it all.

Student D also discusses the compulsory assignments:

> D: [...] they are compulsory and one has to in some way understand and be able to do them. And that I think very much feels like a stressful part of this course. This that one all the time has these assignments. It is very good since one learn this way but one always has, it is like they feel difficult I think.

Student D expresses that the compulsory assignments support him or her to learn, but there is a pressure behind the resource that is negative.

Student G says that he or she has used the *textbook* only as a support to pass the written exam:

> G: [...] and then one is allowed to use the text book at the exam. That is why I have used it. But except for those things I have not used it at all. [...] I want to be able find things in the text book as I go to the exam.

Earlier research in the area stresses that perceived assessment requirements are strong influences on the approach to learning. Ramsden (1984, p. 151) writes: "[i]nappropriate assessment procedures encourage surface approaches". Ramsden also discusses that anxiety has a negative effect on the learning. "Where students *felt* that the assessment situation was threatening [...] they were more likely to adopt a mechanical, rote learning approach to the learning task." (Ramsden, 1984, p. 149). My finding *fear of failure forces learning activity* supports this research. On the other hand, earlier research shows the positive role the teacher can play for students who experience anxiety and fear in their studies: "The lecturer's interest in students, and helpfulness with study difficulties, are the first important qualities influencing students' attitudes and approaches." (Ramsden, 1984, p. 152).

- *Extrinsic rules as a support*

Close to the experience described as 'fear of failure', is when students express that it is a support to be driven by rules and guidelines given by for example the educator.

Expressions like "what we are expected to know" and "where they have put the main focus" are typical in these quotes, and show that the responsibility seems to lie on the teacher more that on the student himself or herself. This way of experiencing support in the learning is mostly connected to the lectures and notes taken during the lectures. An example of this kind of support is student O who discusses the *notes* from the lectures:

> O: [...] then you can see where they put the main focus in course.
> I: Why is that good?
> O: Well because (laughter) that is useful later when the exam comes.

I: Okay.
O: Well then you know something about what to read in the book and what maybe is less important and you can pay less attention to.

The interviewer asks why student D finds it important to take *notes* during the lectures:

D: (sigh) No, but in order to know that I haven't missed anything (giggle), no I don't know [...] that you know they are examples that you can look back at when you shall do an assignment later.

Student D does not only express that he or she is afraid of missing information about what the teacher thinks is important in the course, but also that the notes are a support to pass the compulsory assignments. Student O in the quote above expresses similar experiences. The notes are a support to pass the written exam.

Student F has the teacher's instructions what to study as a support and guide. He or she says about the *textbook*:

F: [...] The Java book. I have in fact tried to read it from cover to cover, the chapters that are on our reading list. I have thus tried to go through it just to really know what we should know. [...]
I: What do you mean with what we should know?...
F: What we should know in the course, what we are expected to know after this course, I've tried to read those chapters.

Student D answers the question how the *compulsory assignments* has supported him or her in the learning:

D: [...] you have been forced to think about what you're doing and been forced to really start to study also or begun to understand what he has done in the course otherwise it is easy to almost put it aside. That you'll come to understand it, what he has covered so far. What it is we are expected to know so far and so. That you start again and start to study a little or do some programming.

Student D says that the compulsory assignments have been a support under constraint in the learning process, and that they show what the students 'are supposed to know'.

Booth and Ingerman (2002, p. 504) found, in a study on how first year physics students make sense of the engineering physics programme over one year that "different views of authority imply different views of knowledge." The authors discuss students' perception of authority as where the responsibility for the structure and outcome of the study is perceived to lie. They further write:

> A student who perceives authority for knowledge lying outside himself will seek ways of satisfying that authority [...] trying to build knowledge fragments into a coherent whole according to *their* plan by studying *their* exam solutions, by reading over and over *their* notes and text-books - a classic surface approach in which attention is paid to the tokens. (Booth and Ingerman, 2002, pp. 504-505)

Booth and Ingermans results emphasize that *Extrinsic rules as a support* is not desirable, since it does not facilitate a meaningful approach to the learning. Students who see the authority lying partly at least with himself or herself will focus on meaning and how "fragments can mesh to one another, reading notes and text-books to spy hitherto unremarked connections - the classic deep approach" (p. 504-505).

Meaningful support

Most of the ways in which students express that resources have supported them are described in terms of meaningful support. Students talk about an interest in the subject, about social motives and how the resources have supported their own activity and understanding while learning.

- *Subject interest as a support*

Some students describe how an interest in the subject itself supports them in their learning.

Student H expresses that the *lectures* have given him or her motivation and a feeling of the course as a whole. Student H says:

> H: yes, what is good with the lectures is this that it starts from the basics and that you get motivation [...] then you feel like it all holds together in some way. Then you don't want breaks in it.

The lectures and the subject itself support student H. Student B expresses the interest for programming as a triggering factor to work hard. The *computer* itself is a resource that supports student B:

> B: I just think it has been interesting to learn. I have sort of learned 10 times as much as I knew before. I have really really enjoyed it, so I have spent a lot of time at the computer.

Student B who expresses a pronounced interest in the subject, discusses the importance of the *lectures* and the *teacher*:

> B: Mm, I think they have been great. He has covered what one should do and explained it [...] So, the practice at the computers and the lectures have complemented each other.
> I: Yes. Why have you attended the lectures?
> B: Because otherwise I wouldn't understand anything when I sit down at the computer.

The examples show that use of resources can support subject interest, and thus facilitate for a meaningful approach to the learning. Earlier research has pointed to the importance of students interest in the subject, and the results from the present study implies how resources used when learning object-oriented programming can create such interest. Ramsden (1984, p. 147) writes: "a lack of interest in the material studied, or a failure to perceive the relevance in it, was associated with a surface approach, while interest was related to a deep approach." Ramsden also discusses the importance of "good teaching" (p. 151) and points to the importance of "if the lecturer can communicate interest and enthusiasm as well as information." (p. 152). This is found among the *subject interest* arguments. My result confirms the result from earlier research in students' approaches to their learning, that the teacher is an important resource.

- *Social stimulation as a support*

Support is sometimes expressed in terms of the positive effects the relation to other students have on the learning.

Student M discusses that he or she had probably been able to solve the *compulsory assignments* alone, but to work with other students has still given him or her joy and also alternative ways to work:

> M: To discuss then, the problems how you should, sort of, do when you should write. Which limitations you should have here and there in the program and how you should solve certain problems. Anyhow it feels like I should've solved them by myself if... I would really have done that. But it has been more fun to work together. [...] Yes, it is more fun to sit and discuss how many ways you can do all the things. Then you also get some help.

Student J discusses how a *friend* has supported him or her to understand concepts:

> J: [...] a guy that is a computer scientist student so he has explained to us what these objects and classes really are [...] That has been very good.

Student H expresses that *the group* he or she belongs to has been a support to come to the lecture. The lectures then gave him or her interest in the subject, but it was the willingness to belong to the group that initiated this subject interest:

> H: [...] when you're in a class than you sort of want to be in school because that is... it is not just because the Java but also kind of the whole thing. I think it's pretty fun to be in school and then... then it sort of leads to you going there and then when you been there a few times then you sort of don't want to miss the rest.

- *Stimulation of the personal activity*

Resources that stimulate the student's own activities are support in the learning.

Student G discusses *teachers* that help in the lab, but give the kind of support that makes the student think on his or her own:

> G: [...] when you're working on an assignment, then you want answers from the one you ask, like, you don't want people to say that it's like this and like this, you want perhaps to get some material and some hints. [...] You're there to learn and then you perhaps don't want to have the answers to the questions right away. You want hints.

Student H describes that the own work with the *compulsory assignments* has supported him or her in the learning:

> H: [...] You learn by trying to think yourself what should, how you should do. Then you could just borrow someone else's and write it down but then you wouldn't learn.

Student K discusses if there are any differences between peer students and teachers as resources when a question is asked:

> K: It should sort of be thus. [...] But from the teacher you get a sort of straight answer, No, this is wrong or do like this instead, but the friend can say if you make changes here, what might happen then? Sort of more fuzzy.
> I: Do you see any advantage with either of them?
> K: Yes it might be, the teacher, he says more like it is like this. Then you do so, but the friend, it isn't the same, you think more perhaps.

Peer students support student K to think on his or her own and thus to get a better understanding.

Activities like reading, writing, discussing and seeing which are described as support for students in their learning, are also stimulation of the own activity.

Student L discusses that the *program examples* handed out by the teacher during the lectures have supported him or her. For this student, visual stimuli are important as a support in the learning:

> L: To get a reminder about how it was, some statements and so, it should look in a specific way and I'm very visual in my learning, I have to see how things look.

Taking notes as a support in the learning is discussed by student O. The student has copied the notes by hand from another student's notes, in stead of using a copy machine. He or she expresses that writing is a support since it enhances the learning better than just reading the notes:

> O: [...] It has been good. I've got all lecture notes and I've copied them down.
> I: You have sort of got them from a friend?
> O: yes, I've sort of copied his.
> I: Then one learn. You haven't photo copied them?
> O: No, that doesn't give so much, sure one looks at them, but when one writes oneself I think.. Yes, that is what I've got.

Hindrance in contrast to support

When resources that support students in their learning are discussed, I also want to point to some hindrances for learning that several students mentioned.

- *Inadequate background knowledge*

Background knowledge is an important resource. Earlier research in the area of students approaches to learning has showed that inadequate background knowledge in the subject studied, specially "where the learning task demands that the student has grasped a fundamental concept" (Marton et al., 1984, p. 148), is associated with a superficial approach. This is especially relevant when students learn object-oriented programming were concepts like object and class are fundamental for the understanding of the idea behind the programming paradigm. These concepts are often introduced at an early stage of the education. Student H discusses what has been difficult with the course:

> H: The difficulties have sort of been that I haven't had any previous knowledge, which maybe hasn't been anticipated, but it has been a very high speed ahead so it has felt that you've known too little, because many had studied at high school [...]

And student G says:

> G: It is this how you, what do you say, well the pre-requisite knowledge, how important it is, how difficult it can be if you haven't even been at a computer before. To start programming when you don't even know what a file is, it is sort of... right, sort of, what a difference there is between different students. Those who have previous experience with computers have it easier to see why... if we say programming, how you sort of place things and stuff.

The one week compulsory introduction course to the computer system at the department, is experienced by some students as not giving sufficient background knowledge. Student G discusses background knowledge in terms of general knowledge of computers. A discussion on how Computer Science education should be adequately started, by learning programming or by

learning about the computer itself, is reflected in Computer Curricula 2001 (Joint Task Force on Computing Curricula, 2001) where several different introductions are suggested. This problem has been touched upon in this thesis in the discussion on how students understand the concepts *object* and *class* in Chapter 5, Section 4.2.2.

- *Too high pressure in studies*

Pressure can influence both the students' subject interest and steer how they study. Student M discusses how the work load in the educational program influences how and what he or she studies. The student says that if the compulsory assignments had not been compulsory, he or she had probably not have done them, although if he or she finds them very interesting:

> M: yes, but at the same time I know that I don't have time to do them anyway, because we don't have the time. There is none, we have lectures till 5pm and then we should try to study something between 6pm and 9pm and then one does what one has to and that is the compulsory assignments. That is what everybody else does.
> I: So, the schedule sort of controls you a great deal?
> M: yes, yes. I don't study anything but the compulsory assignment problem [...]
> M: yeah, I think it is interesting, I would be happy to do them even if they weren't compulsory but as it looks now I would not have time to do them.

This result supports earlier research in higher education. Ramsden (1984, p. 149) writes :

> assessment of an overwhelming amount of curricular material pushes students into surface approaches and an incomplete understanding of the subject matter

Gow and Kember (1990, p. 11) refer to several studies as well as their own study when pointing to the relationship between workload and surface approach to studying. They summarise the results from their study as "the swot, pass and forget syndrome" .

6.8 Discussion on students' use of resources for learning to program

In this section I discuss conclusions and implications for teaching drawn from the results presented in Section 6.6 - 6.7. I first discuss general conclusions. After that I point to conclusions from how the students expressed that they have used the resources, and how the resources supported them in their learning.

The results of the analysis on how the students used resources and how the resources supported them in the learning show a complex pattern. Different students can discuss the same resource, both with a superficial as with a search-for-meaning approach. Sometimes I found quotes where the same student discusses one resource first in one way and then in the other. This is important, and explicitly stressed by Ramsden when he says:

> The idea of an approach to learning is very frequently misunderstood. The most common mistakes are to believe that an approach is a characteristic of an individual person, like the colour of a student's hair; to believe that the approach can be inferred from a student's observable behaviour; to concatenate 'low ability' and surface approaches; or to think that surface and deep approaches to learning are in some way complementary or sequential. [...] Approaches to learning are not something a student *has:* they represent what a learning task or set of tasks *is* for the learner [...] Everyone is capable of both deep and surface approaches, from early childhood onwards. An approach describes a relation between the student and the learning he or she is doing. It has elements of the situation as perceived by the student and elements of the student in it (Ramsden, 1992, p. 44)

And later:

> The distinction between characteristics of students and the nature of different approaches to learning is an absolutely critical one for teachers to understand. Its implications run right through how we should teach. In trying to change approaches, we are not trying to change students, but to change the students' experiences, perceptions, or conceptions of something. (Ramsden, 1992, p. 45)

Another point to make is that the students who expressed that resources had supported them in ways interpreted as superficial, still expressed this in positive terms. Even if the students expressed these ways to use resources in positive terms, I still argue that they are not to prefer. Marton and Säljö (1984, p. 46) discuss this when they say that they do not argue that the deep approach "is always 'best': only that it is the best, indeed the only, way to *understand* learning materials.".

The results from the analysis on how the resources were used when learning object-oriented programming, imply that some resources are mainly used in **a search-for-meaning approach**, to get a better understanding and to find a whole. These resources are

- the teacher
- the computer
- the compulsory assignments, and

- peer students and friends.

Resources mainly used in **an superficial way**, or in a mixed way are

- the textbook
- the notes taken during the lectures and
- examples handed out by the teacher

Even if there are great differences between individuals, and also differences in the use of individual resources, depending on the situation, the data still indicates a clear connection between the resources that were mainly used in a search-for-meaning approach, and how the students described their experiences of meaningful support when learning object-oriented programming, as earlier shown in the discussion on Table 8. This connection will be discussed in the following paragraphs.

When students work at the computer with the compulsory assignments they are *active* in a learning process (third approach in Category 2, Table 8). Many students stress the importance of practicing programming using the computer to be able to learn to program.

The students are allowed to work collaboratively with the compulsory assignments, which I interpret as being a part of the *social stimulation as a support* (second approach in Category 2, Table 8). Interaction with other students and people is also a part of the social stimulation as a support. This is expressed in discussions, in asking questions, or in explaining things to other students, and is also an *active* process. I furthermore notice that the positive interaction with other students which I call *social stimulation*, sometimes cause an *interest for the subject* itself (the first approach in Category 2, Table 8). The positive experience of interaction with peer students seems for some students to be a breeding ground for a positive attitude to the subject learned.

Many students express that the teacher at the lectures was able to mediate a *subject interest* (the first approach in Category 2, Table 8), and he, the teacher in the course, was able to explain the subject so that many students experienced that they could understand. Since I found evidence in the data of students that experience a fear for the subject, the teacher can, in supporting, encouraging and generating a subject interest, be an important resource when learning object-oriented programming.

The resources that are mostly used with a search-for-meaning approach are closely connected to each other and the subject area. The students use the *computer* when they solve the *compulsory assignments*. They are allowed to collaborate with *peer students* for most of the compulsory assignments. *Teachers* and *peer students* are very important resources to be

able to understand concepts and discuss and solve problems when working with the compulsory assignments. Learning to program is not only learning concepts, but learning a skill. Becoming a skilled programmer in object-oriented programming requires both a good understanding of the paradigm with its central concepts and extensive practicing. The teacher can act as important mediator of the thoughts behind the paradigm, and a support when practicing at the computer. It follows from the nature of the subject that the computer with the compiler are not only important but necessary resources in the learning. Collaborate with other people is an important resource that has been pinpointed in this study and is well known as an important factor in software development industry.

Students emphasized that in learning the basic concepts in object-oriented programming, peer students, the lectures and compulsory assignments at the computer are valuable. Some students also mentioned the textbook, while other students stressed that the textbook did not give much support. The textbook is the resource that is described in the most diverse way. The differences in the descriptions of the value of the textbook, to what extent it is used, and how it is used, are large. It reaches from students who have hardly used the textbook at all, to students who have used it as one of the main sources for understanding concepts and to get a fuller picture of the course by reading it from cover to cover at one occasion.

The resources that are mainly used in a superficial way are interesting to study since the picture of how they are used varies so much. I found that even though the use of these resources were often described in a superficial way, in situations were students used a search-for-meaning approach to their learning, the textbook, examples handed out and notes from the lectures played a significant role. I found this for example when students worked with the compulsory assignments aiming at a better understanding. Even if the resources were used as references to find details in the syntax, or code from them were copied, they interacted with the other resources, and made a search-for-meaning approach possible for the students. This implies that the resources themselves do not necessarily lead to a meaningful or superficial approach. Resources that were mainly used in a superficial way contributed considerably to a search-for-meaning approach in some situations. The resources that were mainly used in a search-for-meaning approach could be used superficially if for example the students experienced too hard pressure due to lack of time or background knowledge. The aim for some students when working with the compulsory assignments was described as coming through the course rather than learning, due to lack of time, and students who are not allowed to work collaborative, might use a superficial approach when working with the compulsory assignments.

The most important aspect of the use of resources in the present course, seems to be the stimulation of students' own activity. The support I call *extrinsic rules as a support*, second approach in the first category in Ta-

ble 8, is in some sense the opposite of this. The students who express this approach express that some of the responsibility for their learning, what they should learn, is the teacher's or educational system's, not their own. This superficial approach does not enhance good learning. Responsibility for the own learning is characteristic for a search-for-meaning approach to the learning and is worth striving for. This claim is supported by earlier research (Ramsden, 1992; Booth and Ingerman, 2002; Berglund, 2005). Responsibility for the own education, and activity in the learning are closely connected. Ramsden (p. 155) writes: "Passivity and dependence on the teacher [..] provide an excellent basis for surface approaches" and "deep approaches are associated with activity and responsibility in learning". The superficial content related way to use resources *looking for hints or pieces of code to copy*, second approach in Category 1, Table 7, is an example of a passive way to use resources. I also want to point to the importance for teachers of presenting a *variety* of resources to the students, since different students seem to require different resources in different situations.

Superficial approaches to learning are closely connected to fragmentising. Resources like the compulsory assignments seem to help the students to put the pieces of information together. They seem to help the students into a search-for-meaning approach, where broad use of resources and knowledge from the whole course is used. In the students' description of how they used the compulsory assignments I can see how they were actively engaged in their learning, and how the different resources interacted. This is especially clear for the bigger assignments which demanded understanding of the whole course.

Anderson et al. (1996) discuss whether learning should be done on complex problems and in complex social environments. Referring to previous research they write "It is better to train independent parts of a task separately because fewer cognitive resources will then be required for performance, thereby reserving adequate capacity for learning." When learning to program this argument is in line with variation theory (Marton and Tsui, 2004), which discusses how to make critical aspects of phenomenon come to the fore in students' awareness, and thus make it possible for the student to discern them. This is possible when there is a variation in a dimension corresponding to the specific aspect, while at the same time other aspects are kept invariant. This requires smaller exercises where specific concepts, or aspects of concepts are trained. On the other hand Anderson et al. discuss that sometimes there are reasons "to practice skill in their complex setting. Some of the reasons [...] reflect the special skills that are unique to the complex situation." Problems that enlighten the object-oriented paradigm will of necessity become complex, and demand use of complex resources. Even in the first course in object-oriented programming, the students need to encounter such problems. To be able to learn to manage them, the students need to practise these "special skills that are unique to the complex situa-

tion" as Anderson et al. put it. This is in line with the observations from the data in the present study that the bigger assignments, which involve several resources and knowledge from the whole course, are important and trigger a search-for-meaning approach which enhance good learning.

Summarising the discussion in this section, the results from the study show that there are resources that are mainly used in a search-for-meaning way, while others are mainly used in a superficial way. The latter group show however to be of decisive importance when used together with the resources in the former group in a search-for-meaning approach. This becomes especially clear in the data when students' describe their experiences of their work with the larger assignments where the learning is often described as an active process where students take responsibility for their own learning. Use of and understanding of resources is a part of the subject itself, and learning activities that involves students' understanding and use of resources in a complex way are thus important to facilitate for a search-for-meaning approach when learning to program.

7 Students' motives for learning to program

7.1 Background

In an effort to address *why* students tackle their studies in certain ways and not other, the issue of *motive* is now discussed. The research question posed in this chapter is

- What are the students' motives for learning to program?

Psychology-based theories often discuss *motivations* while I in this thesis discuss *motives*. The difference has its roots in the different research paradigms. Berglund and Wiggberg write:

> It is sometimes tempting to draw parallels between phenomenography and psychologically based theories of learning. However, the differences are important: While psychology discuss what learning is, independent of what the learning is about and the context of that which is learnt, phenomenography studies the experience of learning something, in a particular setting. (Berglund and Wiggberg, 2006)

Motive is here interpreted in a contextualised way, not as personal characteristics. Students' motives are seen in relation to the course and learning outcomes, and thus depending on the study context. More precisely the motives discussed in this chapter are motives that seem to trigger search-for-meaning approach and active learning. The reason for not discussing other motives is that this chapter aims at discussing implication for teaching and learning derived from the positive motives.

The research question is closely connected to how students express their motives to use different resources, thus the motives to use resources are discussed to some extent.

Questions asked in the interview that informed the theme were *What was your aim/purpose when using this resource?* and *Why did you use this resource?* Other questions asked in the interview also informed the theme. Students expressed motives to learn to program for example when they discussed what it means to learn in the present course, but also when answering other questions the theme occurred.

7.1.1 Data analysis

Students' motives to learn belong to the *How*-aspect in the phenomenographic model presented in Chapter 2. In this model the motives constitute the *indirect object* of learning.

The analysis performed in this chapter is content analysis, see Section 6.3. The reason for using content analysis is that the question discussed is rather on students' different motives than on different aspects of one specific

motive. Content analysis is used as a method to investigate an aspect of the phenomenographic model discussed.

Like in the previous chapter, a categorisation was performed to group similar motives for a better overview of the data, not aiming at finding qualitatively different understandings as in a phenomenographic analysis. All data were investigated in order to categorise the different motives found. The motives and categories were not decided in advance.

As in the previous chapter, reliability check (Section 2.3) is performed by letting another researcher read suggested quotes, illustrating the categories, and discuss categories and quotes until we came to an agreement. Validity check is also done in a similar way as in the previous chapter.

7.1.2 Related work

As a background for the motives presented in the chapter, this section discusses earlier research related to students' motives to learn. A significant amount of research in this area has been performed, stemming from the research on students approaches to learning performed already in the early stage of the development of phenomenography, see for example (Marton, 1974; Svensson, 1977; Marton et al., 1984; Biggs, 1987). This research, as presented by Biggs (1987) has been further developed by Kember et el. (1999) . Kember et al. developed a tool for analysing complete data sets including motives that can enhance the learning process, but also other motives. Since the aim with this chapter is to discuss implication for education derived from specific motives, such analytical tool will not be used. Kember et al include three different motives in their model that can be positive. They are *intrinsic* motive, which includes subject interest, *achieving* motives that include for example determination to work hard, perseverance and a strong sense of loyalty to the group, and a third *career* motive.

Berglund (2005) gives examples on students' motives to learn in the field of computer science is . Berglund discusses, with a phenomenographic approach, students' motives to learn computer systems in a distributed project course. Among the different motives found, to learn the subject, to become a better professional and motives involving social competence were identified. Berglund's analysis aims at creating a theoretical framework for analysing students' learning experiences, while in the present study the discussion has an emphasis on implications for education.

7.2 Case students

As explained above, similar motives were grouped together in categories. To illustrate the categories, students that expressed specifically pronounced statements for one of the categories were selected as cases. A case, represented by one student, thus gives voice to one of the categories. This does

not exclude that the students selected expressed other motives as well, or that there were other students who expressed similar motives. The cases are presented below with the motives interpreted by me and written in italics.

Student B: "I love to program because it's fun to program"

Student B who has no previous experience of programming, is devoted to programming. He or she expresses clearly what fun it is to program and how much time he or she has spent in front of the computer. Student B does not need more motive than the programming language itself. Student B comments the aim with learning to program:

> B: To complete the compulsory assignments (laughter). I can't say I have a specific goal. I just think it has been interesting to learn. I have learned something like ten times as much as I knew before. I have really, really enjoyed myself and have thus put in quite a lot of time at the computer.

Student B says about what it means to learn in the course:

> B: I think that means that I should be able to sit by the computer and do small simple things by myself. I think that I've learned that. It is like, I think it is pretty good when you can sit at the computer, because then you can like test things and see if it works and see what happens and such.

Comparing to Kembers et al. study (1999), this is an intrinsic motive, it is fun to program. To some extent it is also an achieving motive. Student B expresses a willingness to work hard and perseverance. All students in the study do not be express that they have found this kind of motive. Many students express other motives for their learning than the challenge the programming language itself gives. Students who have expressed other types of positive motives are therefore interesting to take a closer look at.

Student A: "I'm interested in concepts and to get a theoretical overview"

Student A, who has no previous knowledge in programming, has worked hard but with another motive and approach than student B. Student A says about what has been most important in the course:

> A: (Sigh) I think it is the understanding of how programming languages are structured more than just the command, if you want to do this, it is sort of the thinking, the logical thinking

He or she discusses what learning means in the course:

> A: To learn. I think it is also to get the whole picture, to understand how a programming language is constructed, how the language is structured and how it is related with for instance C++ and that is also a bit more over arching. Detailed knowledge is also useful and to have tried, sort of to have had a taste, it is more the feeling and a bigger understanding than... yeah, that's probably what I think, yeah.

Student A says about the aim of attending the lectures:

> A: Yes, try to pick up something at least. I think it has been very difficult. It has been hard to grasp, it has been something completely new. The goal has thus been to sort of go to the lectures and really trying to get into this and then I can use the text book later. It is always a lot easier to get a first presentation before one sits down to read the text book and such. This gives you an initial holistic image.

Student A emphasizes the importance of getting an overview of the subject, and a good understanding.

When the interviewer asks for resources that have been useful when learning concepts like object and class student A says:

> A: [...] I sort of didn't get a grip on it until I read the text book and it as really the text book that gave me the first understanding [...] It thus took a while until things "clicked", as I said, the real eye-opener was the text book. I sat a whole day and read all the chapters we should read more or less from cover to cover and that is nothing I usually do. But it sort of connected all the time, the things I read here came back 20 pages later and, well, it was a very pedagogical structure I think, lots of repetition. That made it easy to understand. I thought that was really great.

Student A discusses the use of notes from the lectures and the textbook:

> A: I've tried to look at the notes before and so, but it was probably not until half the course had passed, maybe just under, that I decided to read the text book and since then it has been much easier to understand the notes, to understand what has been covered in class because you have to write a lot just to keep up if you write all, and thus have no time to reflect. [...] And before I read the book I had a lot of question marks, like, well, I've written this, but what does it mean. Later when I've read the text book, right, it was that I had read about there.
> I: Mm, so, this with the text book, you have talked a fair bit about the text book, you had a certain agenda when you bought it [...] and read...
> A: Yeah, I wanted sort of see if the text book could give me a better understanding than the notes and the lectures did. And I think I was pretty alone in doing that. [...] Yes, I did talk a fair bit with my class mates. Especially those that said as me, that they really didn't understand this. Like, like what is an object, how can you sort of know

> what it is. Then I tried several times to recommend to read the text book, but they just said God it is no good. But I, well, I thought it was really good.

This is an intrinsic motive, a desire to learn and understand, but also an achieving motive. Student A has put much time and effort in trying to understand concepts and to get an overview and understanding of the subject. He or she also discusses the importance of programming exercises in the learning. The emphasize is however on getting an overview and a good understanding of concepts, which permeates the interview.

Student K: "I want to learn to program, it will improve my problem solving ability"

Another intrinsic motive is the interest to learn the subject with the aim to get knowledge in problem solving in general, and in programming in particular.

Student K, who has previous programming knowledge in the programming language C++ answers the question what learning means in this course:

> K: What is good with courses like this is that one gets to practice problem solving. That is actually really good. One has a problem that one solves in different ways and thus one perhaps find the best way . That is one of the core things I think. Then that one should write in a programming language, this could perhaps be done in anyone. But the dealing with problems, the problem solving, that is something I think is important.

This motive is not limited to the programming course itself. Student K sees benefits of the course beyond the course and the course content. Problem solving is a useful ability in general.

Student H: "I'm going to the class because I want to join the fellowship in the group"

One more intrinsic motive worth noticing is a social motive. Some students seem to be engaged in the course, mainly because the peer students and the fellowship in the group.

Student H is one of the students who describe his or her motives as social:

> H: [..] When you are part of a class, then one wants to go to school, because that's... it is not just for the sake of Java, but also sort of the whole ting. I find it pretty fun to be at school [...]

Student M expresses a similar motive:

> M: [...] Because it's so concrete in some way, that this is my group, or my project and that shall work. That is sort of the thought when you start. This shall work in two days or a week, it doesn't matter, but it will work in the end and then one gets involved in what one do and then one does it.

Peer students and the group have a positive effect on these students to engage in the class.

Student E: "I'm not interested in programming, but I want to learn because I know I will benefit from it"

There are students who express a positive career motive for learning to program. This positive extrinsic motive is interesting from an educational perspective, since it should be relatively easy for teachers to present for the students.

Student E has no previous experience in programming. He or she does not appreciate programming per se, only if it is useful. The course content is not the driving force behind the learning. Student E discusses learning to program:

> E: I sort of think that it is a good thing that all learn to program. [...] I'm sort of not the programming type of person. I can find it fun only when one sees the logic and constructive in it, but.. I get frustrated when it isn't so.

Even though student E does not identifies him- or herself as a programmer, he or she can say about the purpose of learning to program:

> E: Well, but isn't it so that the more you know about computers, the less you become dependent on others.
> I: yeah..
> E: I don't know, if you work somewhere later on and have some insights into things, then I think it opens up a small window that let you at least know what's it all about, even if you don't, I mean, it is the professionals that will deal with the real stuff. We won't ever be really good at this. Or, unless you sort of choose it.

Student F expresses a similar motive. Student F who has not programmed before answers the question what is the point of learning to program:

> F: It is a pretty good tool when you shall do things, other things. To complement with programming, build programs, enter information or data or computations, so I guess it is good to know about it.

Student F expresses the usefulness of programming knowledge:

F: yes, it is a good knowledge to have. Even if it isn't all that fun all the time. [...] It is boring to be the one that should do the programming.

These students explicitly express that they can find programming boring. Nevertheless they express a motive to learn to program, based on a clear understanding of the personal benefits they can get from having programming knowledge.

Student J expresses similar feelings. He or she wants to learn to program only if it is useful:

I: What do you think is the purpose for you when you do...
J: To learn the thing I work.
I: Why should you learn?
J: To be able to use it.
I: To be able to use it in the course or ...
J: To be able to use it otherwise also, one does want to remember something.
I: Mm.
J: Otherwise it wouldn't feel meaningful. Now we will surely use it in labs and such that we have had.

7.3 Discussion on students' motives for learning to program

The previous sections have discussed some motives that seem to trigger search-for-meaning approach and active learning object-oriented programming. Following the terminology in Kembers et al. study (1999), the first four are intrinsic or achieving motivations: *to program is fun and challenging*, *strive for overview and theoretical understanding*, *to improve problem solving ability* and *a wish to join the fellowship of the group*. The last one is extrinsic, *personal benefits of programming knowledge*. As teachers we can benefit from becoming aware of different motives among students that enhance learning, and that we can influence. This section is a discussion on the motives, with implications for teaching.

Student B illustrates a group of students who probably are easy to teach and willing to learn, and who spend much time programming because he or she finds it fun. The programming language itself gives stimulation and challenge enough to work hard.

The empirical data does not give much information on why some students have this driving force. Some examples from the interviews are still interesting to study for educational purposes.

Student A stresses the insight how much can be achieved with programming. This insight seems to have act as an eye opener for how challenging and fun programming itself can be:

A: Well, okay. So, it is somewhat interesting. I can, you sort of gets ones eyes opened [...] *sort of when you sit with the computer, that you*

> *realizes how incredibly much you can do, do myself with the aid of a keyboard* and that is something I've come to realize and I've in the same way come to understand those that sort of are addicted programmers and who find it just so fun. (Author's italics)

Another factor why some students have this type of motive could be that some of them have previous knowledge in programming, and thus have a lead.

Student G says about his/her experience of the course:

> G: [...] Think it is a fun course, absolutely. I have always been interested in computers. It has sort of been fun to do other things also, so I'm very positive to the course.

A third factor worth mentioning is the teacher's impact on the students and the course. Student G explains how the teacher has inspired him or her to study:

> G: [..] It is our lecturer that sort of has started an interest in us.

The first factor mentioned is interesting for educators. It shows the importance of putting the course and course content in a wider context. Even if this is a first programming course, student A has discerned "how incredibly much you can do, do myself with the aid of a keyboard". This study shows that many students have vague ideas of what programming is even at the end of the course. Students own use of programming knowledge, what programming is and can be used for are factors educators can mediate in the education.

The second factor, students' previous knowledge, can often not be affected by educators. Factors possible to influence is yet the level where the course starts, and the pace of the course.

The third factor mentioned is the influence of the teacher. Research on students' learning shows that the teacher plays an important role for students' motive to learn. As mentioned before, Ramsden writes "Students may begin to experience the relevance of the content of the lecture for their own understanding if the lecturer can communicate interest and enthusiasm as well as information. [...] The lecturer's interest in students, and helpfulness with study difficulties, are the first important qualities influencing students' attitudes and approaches." (Ramsden, 1984, p. 152).

Student A might be representative for a group of students who strive for theoretical and conceptual understandings. If so, this indicates implications for teaching necessary for educators to be aware of. As educators we need to realise and meet the motive needed for these students. To provide students with an overview of the course and the course context, and also explicitly

teach what it means to program, and how it fits into the students' education is important. Table 1 can give some information on this. To facilitate for the students to understand the basic concepts within object-oriented programming is possible when we know what is critical in the understanding of these concepts, from a student's perspective.

Student B emphasizes programming as a **handicraft** while student A emphasizes the **conceptual understanding** underlying object-oriented programming. These are two faces of the same coin. Programming as a skill that needs much hands-on-training, and understanding of the paradigm behind the object-oriented programming are both needed and worth striving for. Neither of them can be neglected in the programming education, but there are obviously students who tend to prefer one of these sides to the other. As teachers we need to be aware of our own preference, and remember the importance of the other. There are students who need "the other side" to become motivated in their studies.

Student K is representative for a group of students in the present study by discussion problem solving. Problem solving abilities are either discussed regarding programming skills, or as a general ability. Table 1 shows educationally critical aspects of what it means to learn to program, where Category four and five include problem solving. As discussed in Section 4.3, it is important that the students' understanding of what learning to program means include this aspect. As educators we need to facilitate for the students to discern problem solving as an important aspect.

The social motive expressed by **Student H** gives interesting implications for the planning of programming courses. To encourage students to work in groups and help each other can give students motives for their studies that enhance the learning, and thus promote good learning. This might be one reason why pair programming has shown good results in several studies.

Student E is interesting from an educational perspective when we teach non-major computer science students. Student E has succeeded to find a motive for his or her learning that enhances the learning, even though no or little interest in the subject is expressed. There might be a considerable amount of students who are not interested in programming, but still have to study it. Everyone, both educators and students would benefit from finding such motives for the studies. The usefulness of programming knowledge in later studies and coming careers are worth considering and discussing with the students. This includes general knowledge of computers and the ability to communicate with professional programmers. It also includes sufficient programming knowledge to be able to change existing code for certain purposes, and to be able to write smaller programs or scripts on your own.

As teachers we meet students with different motives for learning to program. Unfortunately not all of them find programming, in terms of the programming language and its concepts, interesting and challenging. Stu-

dent N can express the understanding in Category three in Table 1, *Learning to program is experienced as to gain understanding of computer programs as they appear in everyday life*, but can nevertheless not find the course much meaningful, see Section 4.2.3. The third category in Table 1 describes an understanding of what it means to learn to program that has its focus outside the course and course context. It still seems to be too shallow an understanding to give a satisfactory feeling of meaningfulness. There seems to be a border between the first three and last two categories in Table 1. To identify motives that we can mediate to our students could be rewarding, both for students and educators. The data in this study point in the direction that students, who do not see the programming language itself and its concepts as a driving force, need to discern what it means to learn to program, as expressed in the last two categories in Table 1 to find motives for their learning. A teaching and learning environment that explicitly highlights the problem-solving aspect, encourage social behaviour, and reveals the personal benefits and uses the students can get of programming knowledge, is thus worth striving for.

8 Conclusions and Future work

The main focus of this thesis is novice students' learning of object-oriented programming. What does learning to program involve in a first programming course? In a qualitative study of first year students this question is investigated broadly by looking at *what* students learn as well as *how* they learn.

The *what*-aspect of the learning, the learning outcome, includes an overall understanding of what learning to program means and also students' understanding of concepts within the subject studied. The question on what learning to program means is central since many students have not studied the subject before their first university course. Conceptual understanding is important for learning of the subject. Many of the concepts students encounter already at an early stage in the course are fundamental to the specific programming paradigm, and constitute core knowledge of the subject (Bar-David, 1993). Examples of such concepts are *class* and *object*. One of the foci in this thesis is students' understanding of these concepts. The object-oriented paradigm is constructed for larger software systems, where the base units are the classes with their instances, the objects, to perform the task given.

Learning to program a computer does however not only involve learning concepts, but also, by definition, the use of complex resources like computers, compilers and editors, which many students do not have any deeper knowledge of. Furthermore learning is influenced by the learners' motive to learn. These two issues belong to the *how*-aspect of the learning.

The primary research approach in this thesis is phenomenography (Marton and Booth, 1997). Marton and Booth have, based on a discussion on learning theories, developed a model for analysing students' experiences of learning. This model includes the analytical separation of the *what*- and *how*-aspects and makes a theoretical investigation of the learning experience possible. The model has been used in the present study as a tool to analyse the complex picture of students' experience of learning object-oriented programming.

Phenomenography provides a tool for finding qualitatively different understandings of phenomena in a group of people. It allows the subject of interest, here object-oriented programming, to be in focus, and still taking the students' perspectives. Since the researcher who performs the analysis knows the subject it is possible to maintain an expert view during the analysis and not loose the focus on the subject. Taking students' perspectives give the advantages to show what aspects of the understanding of a phenomenon that are critical for the students in their learning. If teachers become aware of these aspects, it opens for the possibility for them to reflect these aspects in their teaching, see Section 2.2.

The result of a phenomenographic analysis is an outcome space describ-

ing the qualitatively different understandings found in the group of students studied. The outcome space is interesting per se for educators as it reveals educationally critical aspects of students' understanding of the phenomenon investigated. The results can further be taken into a discussion on implications for teaching by use of variation theory (Marton and Tsui, 2004) which I have done in the analysis of students' conceptual understanding. I have shown how the results on students' understanding of *object* and *class* might be implemented in the teaching.

This chapter summarizes the discussions and conclusions from the study of the four research questions in the thesis. Furthermore the chapter shows how the results are connected to each other. The four research questions are important aspects of the students' complex learning experience, and together they cover a broad range of the those experiences.

The results presented in this thesis are empirically built. They discuss novice students' understanding of what learning to program means, their conceptual understanding, and their use of resources and motives to learn. Conclusions on implications for object-oriented programming education are also discussed. The conclusions drawn are in this sense more relevant and applicable for educators in object-oriented programming than the earlier, more general research compared to. They also contribute to the understanding of students' learning of programming in a wider context than earlier research since they cover students' use of several resources, and not only one or a few resources as in previous work found.

8.1 Conclusions

The first two columns in Table 9 show the results of the two phenomenographic analyses presented in the thesis. The first column describes the results from the analysis on students' different understandings of some central concepts. The second column presents the results on students' understanding of what it means to learn to program. Each understanding presented is important and relevant when learning to program, and a good understanding includes them all. The last column is a summary of these analyses.

The analysis of students' understanding of concepts revealed three different ways to understand the concepts object and class, which are summarised in three categories of description, see the left-most column in Table 9. The first category however hardly corresponds 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 phenomena, the present study points to 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.

The analysis of how the students understand what it means to learn to program, revealed five different understandings described in the second column in Table 9. The first three categories describe shallow understandings of what it means to learn to program. Here, learning to program is seen as learning a programming language, and this is experienced as a difficult activity delimited to the course. Students who only expressed understandings corresponding to these categories often said that they had problem to know how to go about studying, and many experienced that learning to program involves a special way of thinking that they had not yet acquired. The last two groups in the second column differ from the first three. Students who expressed understandings in line with these seemed to know how to study computer programming and could express programming as an activity that is relevant outside the course and useful for them. The results indicate that it is crucial for students to get an understanding of what it means to learn to program that exceeds the first three categories described.

From the results of students' understanding of concepts and what learning to program means, two broad categories can be discern as explained above. This is presented in the third column in Table 9. The first category in the third column describes understandings that are delimited to the programming language itself, and to a narrow description of what learning to program means. This category describes understandings that, if they are the only understandings students are aware of, seem to lead to problems in the capability to program, and even in the capability to *study* programming. Problems and confusions are expressed by the students who express only these understandings.

There seems to be a critical boundary between this first category and the second. The second category describes understandings that have reached beyond the programming language and the course itself. A larger context and personal benefits are sought for in this category. The results indicate a connection between students' understanding of concepts and their understanding of what it means to learn to program, see Section 4.3. Students seem to need to discern not only more advanced ways to understand concepts, but also the more advanced ways to understand the general question what learning to program means. Teachers, aware of the critical understandings of what it means to learn to program, and critical understanding of concepts can use variation theory to facilitate for students to discern these more advanced understandings.

In the analyses of the data on students' use of resources, I found that I could categorize how the students *used* the resources and how the resources *supported* the students in their learning, using the search-for-meaning/superficial dichotomy. These analyses revealed that some resources are mainly used in a search-for-meaning way and thus come to enhance the learning. These resources are the teachers, the computers with the compulsory assignments,

Students' understanding of the concepts object and class	**Students' understanding of what it means to learn to program**	**Summary**
1. 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.	1. Learning to program is experienced as to understand some programming language, and to use it for writing program texts. 2. As above, and in addition learning to program is experienced as learning a way of thinking, which is experienced to be difficult to capture, and which is understood to be aligned with the programming language. 3. As above, and in addition learning to program is experienced as to gain understanding of computer programs as they appear in everyday life.	**1.** Understandings limited to the course and the programming language and program layout. Shallow discussions on the use of programs in everyday life.
2. 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. 3. Class is experienced as a description of properties and behaviour of objects, where object is understood as a model of some real world phenomenon.	4. As above, with the difference that learning to program is experienced as learning a way of thinking which enables problem solving, and which is experienced as a "method" of thinking. 5. As above, and in addition learning to program is experienced as learning a skill that can be used outside the programming course.	**2.** Understanding of concepts as abstract data types and models of the real world and programming as problem solving. It includes the world outside the course and later career.

Table 9: Comparison of the two phenomenographic outcome spaces. From an educational perspective they are divided in two qualitative categories. The categories in a column are inclusive; the latter categories include the understandings in the former.

and peer students and friends. Resources that are mainly used in a superficial way are the textbook, the notes taken during the lectures, and the program examples handed out at the lectures.

The students' experience of the use of resources are however diverse and depend on the situation. The same student can in some situation use resources in a search-for-meaning way, but in another situation in a superficial way. Moreover one specific resource can be used in some situations in a search-for-meaning way, and in other situations in a superficial way. The results also indicate that too high pressure in terms of lack of time and background knowledge trigger a superficial use, even for the group of resources that are mainly used in a search-for-meaning way. On the other hand, the group of resources that are mainly used in a superficial way seem to be important when they were used together with the resources typically used in a search-for-meaning way, in strive for understanding and finding a whole context.

This study's primary contribution concerning students' use of resources is the role of resources in the learning of object-oriented programming. The object-oriented paradigm has been developed for the solving of larger problems that require many interacting classes. In the data I found that learning is best promoted when the students have to solve the larger compulsory assignments. Some of them included several classes and thus mirrored the thoughts behind the paradigm better than the smaller assignments. The larger assignments seemed to facilitate personal responsibility since they demanded comprehension of larger parts of the course rather than just details. They seemed to help the students to put the pieces of information together and demanded the students not to use a superficial approach in the learning. They also seemed to encourage students to use several resources, primarily the ones that promote search-for-meaning learning, but also other resources.

If the student knew how to go about studying, which was often described as being able to do the compulsory assignments at the computer, he or she would probably have a better chance to use the other resources, teachers and peer students in a search-for-meaning way that enhance a better learning of the subject. Even the resources that primarily were used in a superficial way showed to be important for solving these tasks, when used together with the resources that promoted search-for-meaning learning. In this way the resources interact with each other, and the use of several resources together are of decisive importance for the learning.

The use and understanding of resources is a part of the subject itself. Learning that involves students' understanding and use of resources in a complex way are important to facilitate for a search-for-meaning approach to learning to program. With reference to the discussion on variation theory in Chapter 5, I want to emphasize that there is still a need for smaller exercises in the early stages of learning of concepts, to help students to discern specific critical aspects of the concepts. This can be done when smaller

assignments are constructed which focus on a specific critical aspect of a concept, and that aspect is lifted from the surrounding noise.

One of the themes investigated in this study is students' motives to learn to program. There are some interesting connections between students' motives to learn to program and the understanding of what it means to learn to program. The results indicate that if students do not see the programming language itself and its concepts as a driving force for the learning, or have a social motive to learn, they need to discern what it means to learn to program, as expressed in the last two categories in the second column in Table 9, to find positive motives for their learning.

As I have discussed in Chapter 7, there exist several motives that are rewarding for the learning of the subject. The motives discussed are subject interest in terms of a strive for conceptual understanding and/or an interest in learning the programming skill; social motives for attending the class; interest to get a general problem solving ability; and the benefit that knowledge of programming can give in later courses and career. As educators we can not expect all our students to start the course with a subject interest, or to get a subject interest during the course. For that reason the latter motives are interesting to study. In Section 7.3 I have argued that educators can help the students to discern these positive motives, and thus facilitate for students to learn the subject.

An interesting result is that there were students who found a subject interest starting with social motives to attend the class. Some students attended lectures because they wished to join the fellowship in the group. When the students came to the lecture, the lecturer succeeded to create an interest for the subject which gave the students a motive for learning. This shows the important role the teacher has for the learning, and the importance of creating a learning environment that involves the possibilities for social interaction. In this way teachers can help students who start with low interest in the subject to find an interest.

Students' motives to learn and resources provided in the course are thus of decisive importance for the learning. Students with a social motive to learn are encouraged by group activities, students with motives to learn theory need lectures where the fundamental thoughts behind the paradigm is highlighted, students with career motives need to know how the course fits in their education and how they can benefit from programming knowledge in coming careers, and students who want to enhance their problem solving ability need programming problems with connection to their reality.

As educators we can help or constrain our students' learning by means of which resources we provide. Implications from the results are that educators should provide a variety of resources so that resources should be made available that can connect to the different motives found in the study.

In the study there were students who expressed that they had problems not only to understand concepts, but also to know how to go about studying. Problems to know how to go about studying indicate problems to know how to use available resources efficiently. Students who do not know how to use resources like the computer with the compiler in an efficient way, will have difficulties to perform assignments, which is expressed by the students as very important for the learning of concepts.

This finding is connected to students' understanding of what learning to program means and the discussion on different levels of conceptual understanding. Data in the study, supported by earlier research in Mathematics education, suggest that many students need 'canonical procedures' in their learning to program (Hazzan, 2003). 'Canonical procedures' are step-by-step instructions that help students to practise the subject studied. They scaffold students' learning and are necessary for developing conceptual understanding. If such procedures are not discerned by the students they may get problems to advance in their learning. 'Canonical procedures' correspond to and are necessary for a 'process conception'. This is, according to Sfard (1991), a level of conceptual understanding that precedes the more advanced and desirable level of conceptual understanding, the 'object conception'. The result from the present study, where students say that they do not know how to study programming, indicates that these students do not discern the 'canonical procedures' necessary for the learning, and have thus not even reached a 'process conception'. This reasoning might be in line with the broad discussion among many students in the interviews about the special 'program thinking' that some of them express they have not yet achieved, but which they discuss is necessary for learning to program.

Understanding of concepts and use of resources are thus related to each other and to the understanding of what learning to program means and students' motive to learn. The four research questions cover a wide range of students' experiences of learning object-oriented programming, and the study empirically shows that they are all depending on each other and important in the learning process.

8.2 Future work

The results from the study show that the process of learning object-oriented programming includes an initial threshold that many novice students have problems to pass. In Chapter 4, I discuss the special thinking that many students mention, and name it 'programming thinking'. It is described by the students as something necessary to acquire to be able to understand programming, and often expressed as difficult to grasp. Students who have come to an understanding of what it means to learn to program as learning a way of thinking which enables problem solving, and a "method" of thinking, or that it is learning a skill that can be used outside the programming course,

seem to have passed a threshold in their understanding, see the second column in Table 9.

Taking the view that learning to program is to enter a new culture, I plan to further investigate this 'programming thinking'. Mullholland and Wallace (2000) discuss the necessity for novice science students to find bridges or 'entryways' into the new world of science "in order that professional development could continue." (p. 16). The authors continue by stating that "[a]lthough these bridges were based in the original world or perspective held, crossing them requires reorganisation of ideas and attitudes so that successful transition to the new world could be made." I plan to further investigate the 'programming thinking' in the light of 'entryways' into a new computer science culture. I consider the anthropological sense of culture discussed by Säljö (2000) "culture [means] the set of ideas, values, knowledge, and other resources that we acquire through interaction with the world around us." In this work the concept of *ways of thinking and practicing*, WTP's (Entwistle, 2003) will be central. It is intended to capture subject-specific ways of thinking and practicing that are central elements of the culture within a discipline. My goal is to identify *threshold concepts* within the subject area (Meyer and Land, 2005). Meyer and Land state that threshold concepts are " 'conceptual gateways' or 'portals' that lead to a previously inaccessible, and initially perhaps 'troublesome', way of thinking about something. A new way of understanding, interpreting, or reviewing something may thus emerge [...]" Threshold concepts can in this way be discussed as 'entryways' into the computer science culture, and by use of Meyer and Land's definition of threshold concepts my goal is to identify 'entryways' into the culture.

A second line of future work is how the use of resources is connected to the learning of concepts in object-oriented programming. The results from the present study indicate such a connection as discussed earlier in this chapter. I plan to discuss use of resources in terms of the P in WTP, 'Practicing'. Can identification of threshold concepts also be relevant for revealing aspects of ways of practicing? This thought is supported by Mullholland and Wallace (2000) when they write "the availability of resources can be thought of as entryways [...] to become a successful science student." (p. 18).

In the data I found evidence of students who had problems to know how to go about studying programming. This indicates that they do not even know how to use available resources, how to 'practise' in a way that promotes good learning. To identify 'canonical procedures' for novice students in object-oriented programming might be a way to help these students. 'Canonical procedures' are discussed in Chapter 4. They are step-by-step procedures students use when they are still at a low level of abstraction, at a process conception, in learning of concepts. The goal is to reach a higher level of abstraction, the object conception. If students discern these

step-by-step procedures, the procedures can act as tools for them to reach both a better conceptual understanding and better programming skills due to the possibilities to more efficient practicing of programming. In this way practising and conceptual understanding go hand-in-hand.

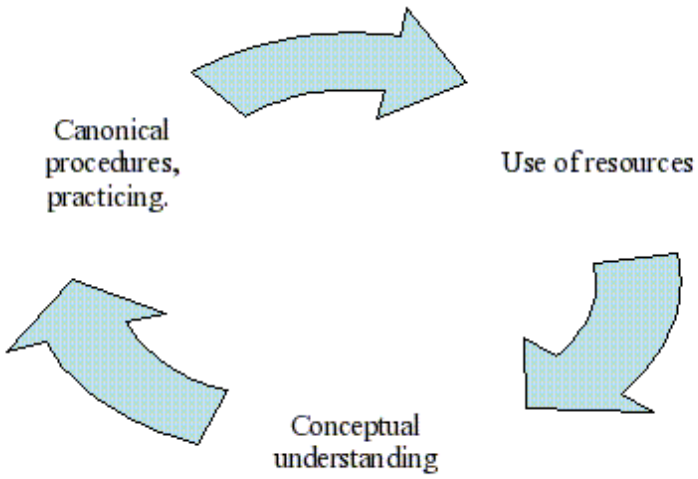

Figure 5: Illustration of the connection and development of learning to program as conceptual understanding and practicing.

Since the data provide evidence that there are students who do not even discern canonical procedures when learning to program, and thus hardly even reach a process conception, I plan to investigate this in the light of the core role resources like the computer and computing exercises play in the learning. The differences between learning to program and learning other subjects students have met before, may explain some of the difficulties many students express in this study. Computers, compilers and editors etc., are not only tools in the learning, but knowledge of them is part of the goal of the course. These resources' central role in the learning, and the fact that the necessary resources are advanced and not profoundly known by all students, may increase the importance of identifying and explicitly showing students canonical procedures. Canonical procedures can enhance the ability to use the resources, and the ability to use resources is a help for students to study which can enhance conceptual understanding. This is illustrated in Figure 5.

Here I discuss both tacit concepts and concepts explicitly taught in a first programming course. By tacit concepts I mean concepts that can not be

expected to be understood in the subject-specific meaning by non-experts and which are used *en passant* by the teacher and/or in the text book although they have a subject-specific meaning of fundamental importance for understanding the subject matter at hand.

In this way I plan to take the research questions *How do students understand abstract concepts in object-oriented programming?* and *How do students use resources and experience support of such in the learning?* further. The results from the present study show the close relationship between these questions in the learning of object-oriented programming, and point to interesting issues for further investigation.

References

Anderson, J., Reder, L., and Simon, H. (1996). Situated learning and education. *Educational Researcher*, 25(4):5–11.

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.

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional.

Ben-Ari, M. (1998). Constructivism in computer science education. In *ACM SIGCSE Bulletin , Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, Volume 30 Issue 1*.

Berglund, A. (2005). *Learning computer systems in a disributed project course. The what, why, how and where*. PhD thesis, Uppsala University, Department of Information Technology, Sweden.

Berglund, A. and Wiggberg, M. (2006). Students learn cs in different ways. insights from an empirical study. In *Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education*, pages 265 – 169.

Biggs, J. (1987). *Student Approaches to Learning and Studying*. Australian Council for Educational Research.

Booth, S. and Ingerman, Å. (2002). Making sense of physics in first year of study. *Learning and Instruction*, 12:493–507.

Booth, S. A. (1992). *Learning to Program. A phenomenographic perspective*. Number 89 in Göteborg Studies in Educational Science. Acta Universitatis Gothoburgensis, Göteborg, Sweden.

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.

Bruce, C., McMahon, C., Buckingham, L., Hynd, J., Roggenkamp, M., and Stoodly, I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education*, 3:143–160.

Chen, S. and Morris, S. (2005). Iconic programming for flowcharts, java, turing, etc. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*.

Dahlbom, B. (1993). Mind is artificial. In Dahlbom, B., editor, *Dennett and his critics. Demystifying mind.*, pages 161–183. Oxford: Blackwell.

Daly, C. and Waldron, J. (2004). Assessing the assessment of programming ability. In *Proceedings of the thirty-fifth SIGCSE technical symposium on Computer science education*, pages 210–213.

Daniels, M., Berglund, A., and Petre, M. (1999). Reflections on international projects in undergraduate cs education. *Computer Science Education*, 9(3):256 – 267.

Eckerdal, A. (2002). Om förståelse av begreppet inkapsling - en analys av en pilotstudie. Unpublished pilot study.

Eckerdal, A. and Berglund, A. (2005). What does it take to learn 'programming thinking'? In *Proceedings of the 1st International Computing Education Research Workshop*, pages 135 – 143.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., and Zander, C. (2006). Can graduating students design software systems? In *Proceedings of the thirty-seventh SIGCSE technical symposium on Computer science education*.

Eckerdal, A. and Thune', M. (2005). Novice java programmers´ conceptions of 'object' and 'class', and variation theory. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 89 – 93.

Ellis, A., Carswell, L., A., B., Deveaux, D., Frison, P., Meisalo, V., Meyer, J., Nulden, U., Rugelj, J., and Tarhio, J. (1998). Resources, tools, and techniques for problem based learning in computing. report of the iticse'98 working group on problem based learning. In *Proceedings of the 3th Annual Conference on Innovation and Technology in Computer Science Education*.

Entwistle, N. (2003). Concepts and conceptual frameworks underpinning the ETL project. Occasional Report 3 of the Enhancing Teaching-Learning Environments in Undergraduate Courses Project, School of Education, University of Edinburgh, March 2003.

Fleury, A. E. (1999). Student conceptions of object-oriented programming in java. *The Journal of Computing in Small Colleges*, 15(1).

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

Fleury, A. E. (2001). Encapsulation and reuse as viewed by java students. In *ACM SIGCSE Bulletin , Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, Volume 33 Issue 1*.

Goldman, K. J. (2004). A concepts-first introduction to computer science. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*.

Gow, L. and Kember, D. (1990). Does higher education promote independent learning? *Higher Education*, 19(3):307 – 322.

Gray, K. E. and Flatt, M. (2003). Professorj: a gradual introduction to java through language levels. In *Companion of 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

Guba, E. (1981). Criteria for assessing the trustworthiness of naturalistic inquiries. *Educational Communication and Technology Journal*, 29:75 – 91.

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*.

Hazzan, O. (2003). How students attempt to reduce abstraction in the learning of computer science. *Computer Science Education*, 13(2):95–122.

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.

Jenkins, T. (2001). Teaching programming - a journey from teacher to motivator. In *2nd Annual LTSN-ICS Conference, London.*

Joint Task Force on Computing Curricula (2001). Computing curriculum 2001, computer science volume. Technical report, IEEE Computer Society and Association for Computing Machinery. Available at http://www.sigcse.org/cc2001/.

Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., and Gehringer, E. (2004). On understanding compatibility of student pair programmers. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education.*

Kay, J. S. (2003). Teaching robotics from a computer science perspective. *Journal of Computing on Small Colleges*, 19(2):329–336.

Kember, D., Wong, A., and Leung, D. Y. P. (1999). Reconsidering the dimensions of approaches to learning. *British Journal of Educational Psychology.*

Kölling, M. (1999a). The problem of teaching object-oriented programming, part 1: Languages. *JOURNAL OF OBJECT-ORIENTED PROGRAMMING.*

Kölling, M. (1999b). The problem of teaching object-oriented programming, part 2: Environmentss. *JOURNAL OF OBJECT-ORIENTED PROGRAMMING.*

Kvale, S. (1996). *InterViews: An introduction to qualitative research interviewing.* Thousand Oaks, CA: Sage.

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

Lahtinen, E., Ala-Mutka, K., and Järvinen, H. M. (2005). A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education.*

Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. (2004a). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150.

Lister, R., Box, I., Morrison, B., Tenenberg, J., and Westbrook, D. (2004b). The dimensions of variation in the teaching of data structures. In *Proceedings of the 9th annual ACM SIGCSE Conference. ITiCSE: Innovation and Technology in Computer Science Education.*, pages 92–96.

Mahmoud, Q. H., Dodosiewicz, W., and Swayne, D. (2004). Redesigning introductory computer programming with html, javascript and java. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*.

Marton, F. (1974). Inlärning och studiefärdighet. Technical Report 121, Rapporter från Pedagogiska institutionen, Göteborgs universitet.

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

Marton, F., Hounsell, D., and Entwistle, N. (1984). *The Experience of Learning*. Scottish Academic Press.

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

Marton, F. and Säljö, R. (1984). Approaches to learning. In Marton, F., Hounsell, D., and Entwistle, N., editors, *The Experience of Learning*. Scottish Academic Press.

Marton, F. and Tsui, A. (2004). *Classroom Discourse and the Space of Learning*. Lawrence Erlbaum Ass., Mahwah, NJ.

Mayring, P. (2000). Qualitative content analysis. Forum: Qualitative Social Research [On-line Journal], 2000, 1(2) Available at: http://www.qualitative-research.net/fqs-texte/2-00/2-00mayring-e.htm.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *SIGCSE Bulletin*, 33(4):125–180.

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

Meyer, J. H. and Land, R. (2005). Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. *Higher Education*, 49:373–388.

Moskal, B., Lurie, D., and Cooper, S. (2004). Evaluating the effectiveness of a new instuctional approach. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*.

Mullholland, J. and Wallace, J. (2000). Restorying and the legitimation of research texts. In *The annual meeting of the National Association for Research in Science Teaching, New Orleans*.

Newman, I., Daniels, M., and Faulkner, X. (2003). Open ended group projects, a 'tool' for more effective teaching. In *Proceedings of the fifth Australasian conference on Computing education*.

Pong, W. Y. (1999). The dynamics of awareness. In *8th European Conference for Learning and Instruction*.

Powers, K., Cooper, S., Goldman, K., Carlisle, M., McNally, M., and Proulx, V. (2006). Tools for teaching introductory programming: What works? In *Proceedings of the thirty-seventh SIGCSE technical symposium on Computer science education*.

Ramsden, P. (1984). The context of learning. In Marton, F., Hounsell, D., and Entwistle, N., editors, *The Experience of Learning*. Scottish Academic Press.

Ramsden, P. (1992). *Learning to Teach in Higher Education*. Routledge, London, New York.

Roberts, E. (2004a). 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. (2004b). Resources to support the use of java in introductory computer science. In *Proceedings of the thirty-fifth SIGCSE technical symposium on Computer science education*.

Roberts, E. (2006). Special session the acm java taxk force: Final report. In *Proceedings of the thirty-seventh SIGCSE technical symposium on Computer science education*.

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

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

Säljö, R. (2000). *Lärande i praktiken Ett sociokulturellt perspektiv*. PRISMA.

Sandberg, J. (1997). Are phenomenographic results reliable? *Higher Education & Development*, 16(2):203–212.

Sfard, A. (1991). On the dual nature of mathematical conceptions: Reflections on processes and objects as different sides of the same coin. *Educational Studies in Mathematics*, 22:1–36.

Sommerville, I. (2004). *Software Engineering*. Addison Wesley.

Svensson, L. (1977). On qualitative differences in learning: Iii. study skill and learning. *British Journal of Educational Psychology*, 47:233–243.

Thomas, L., Ratcliffe, M., and Thomasson, B. (2004). Scaffolding with object diagrams in first year programming classes: Some unexpected results. In *Proceedings of the thirty-fifth SIGCSE technical symposium on Computer science education*.

VanDeGrift, T. (2004). Coupling pair programming and writing: Learning about students' perceptions and processes. In *Proceedings of the thirty-fifth SIGCSE technical symposium on Computer science education*.

Åkerlind, G. (2005). Variation and commonality in phenomenographic research methods. *Higher Education Research and Development*, 24:321–334.

A Interview questions

Interview with Aquatic and Environmental Engineering students, spring 2002

1. **Introduction**

   - Presentation and contact making
     - I introduce myself, my background and the present work.
     - Ask for the student's name.
     - Talk about the purpose with the study:

     How students learn object-oriented programming, specifically conceptual understanding and which resources students use that promote learning.

     Goals: to reveal individual understandings, to be able to map the whole group. Connect the results to education.

     - Inform: This is not at test!
     - Tell the student that he/she is welcome to talk freely, no need for waiting for my next question. The discussion is the most important for me, not "wrong" or "correct" answers.
     - If there is something I ask for that the student doesn't know, that's interesting too.
     - If there is something the student doesn't want to answer, it's AK.
   - The use of the tapes: the whole interview will be transcribed and analysed by me. The tape will only be listen to within the research group at the department. I guarantee that the students' teacher will not get access to the tape until the course is over.
   - Topics for the interview:
     - Resources you've used and how you've used them.
     - Some central concepts within object-oriented programming.
   - - Ask if the student has any questions.

2. **What does learning mean?**

   - - What has the course been about? Mention shortly what you've found most important.
     - What has been difficult?
     - You've learned a lot during the course. What does it mean to learn this course? (What does learning mean?)

- How many of the compulsory assignments have you finished up to now?

3. **Resources & use of resources.**

   - There are many available resources and tools you can use in the course. I'm interested in which resources and tools you have used and which one you experienced as most meaningful, useful?

     By resources and tools I mean everything you think has helped you when you've worked with the course, and specially when you've learned the concepts 'object' and 'class'.

     (**For the interviewer:** Write down a list with the resources the student mention. Ask the following questions for ech resource on the list:)

   - How did you use the resources? How did they help you in your learning? (**Help for the interviewer to think about:** Where are the resources directed? To pass the exam, learn a detail, understanding, "just" pass e.g. an assignment?)
   - Why did you use this resource? What was the goal and purpose? What has been gained by using this resource?
   - Which resources have been useful when you experienced that something was difficult in the course?
   - How do you experience that the resources have been connected to each other? (Textbook with assignments etc...) (**For the interviewer:** follow up question could be: What resources did you use if you didn't use the textbook? Etc...)
   - **For the interviewer:** check some central resources the institution provides if the students do not mention them: lectures, compulsory assignments, computer exercises, Internet, final exam, group discussions

4. **OOP-phenomena Object and class:**

   I have paper and pens here if you'd like to write something down or draw something.

   a) **What** do you experience an object to be? Draw, write and/or tell me. (**For the interviewer:** A picture, a simile using words, a Java program, an example, or...? Give the student paper and pens.)

   What is a class? What does an object contain?

   b) What do you think is the **purpose** of using objects/classes? Why do you create objects/classes?

c) The relation between object and class? (One object - several objects of the same/different classes?)

c) Did you find it difficult/easy to understand objects and classes? How did you come to the understanding you have now?

5. **Conclude**

   - How did you prepare yourself for the final exam? Which resources did you use?
   - **For the interviewer:** If necessary pick something from the interview as a conclusion and tuning: "Did I understand you correctly when you talked about ....?"
   - Tell the student how the interviews after the analysis will be presented: in my thesis, in scientific conferences and journals etc. All subjects will be anonymous.
   - If you like, you can be on an address list and get information of published material.
   - I don't have any more questions. Do you have anything more to add or any questions?

- After the interview: Write down, or tell the tape recorder my impressions of the interview: facial and body expressions, the "feeling" of the interaction between me and the subject in the interview.

Recent licentiate theses from the Department of Information Technology

2006-006 Anna Eckerdal: *Novice Students' Learning of Object-Oriented Programming*

2006-005 Arvid Kauppi: *A Human-Computer Interaction Approach to Train Traffic Control*

2006-004 Mikael Erlandsson: *Usability in Transportation – Improving the Analysis of Cognitive Work Tasks*

2006-003 Therese Berg: *Regular Inference for Reactive Systems*

2006-002 Anders Hessel: *Model-Based Test Case Selection and Generation for Real-Time Systems*

2006-001 Linda Brus: *Recursive Black-box Identification of Nonlinear State-space ODE Models*

2005-011 Björn Holmberg: *Towards Markerless Analysis of Human Motion*

2005-010 Paul Sjöberg: *Numerical Solution of the Fokker-Planck Approximation of the Chemical Master Equation*

2005-009 Magnus Evestedt: *Parameter and State Estimation using Audio and Video Signals*

2005-008 Niklas Johansson: *Usable IT Systems for Mobile Work*

2005-007 Mei Hong: *On Two Methods for Identifying Dynamic Errors-in-Variables Systems*

2005-006 Erik Bängtsson: *Robust Preconditioned Iterative Solution Methods for Large-Scale Nonsymmetric Problems*

2005-005 Peter Nauclér: *Modeling and Control of Vibration in Mechanical Structures*

UPPSALA
UNIVERSITET

Department of Information Technology, Uppsala University, Sweden