Ways of Thinking and Practising in Introductory Programming

Anna Eckerdal
Division of Scientific Computing,
Department of Information Technology,
Uppsala University, Uppsala, Sweden
Anna.Eckerdal@it.uu.se

Abstract

In computer programming education it is generally acknowledged that students learn practical skills and concepts largely by practising. In addition it is widely reported that many students face great difficulties in their learning, despite great efforts during many decades to improve programming education.

The paper investigates and discusses the relation between novice computer programming students' conceptual and practical learning. To this end the present research uses Ways of Thinking and Practising, WTP as a theoretical framework. In the present research Thinking is discussed in terms of students' learning of concepts, while Practising is discussed as common novice students' programming activities.

Based on two empirical studies it is argued that there exists a mutual and complex dependency between conceptual learning and practise in students' learning process. It is hard to learn one without the other, and either of them can become an obstacle that hinders further learning. Empirical findings point to the need to research the relationship between conceptual understanding and practise to better understand students' learning process.

The paper demonstrates a way to research how students' learning of practise and concepts are related. Results from a phenomenographic analysis on novice programming students' understanding of some central concepts are combined with an analysis based on elements from variation theory of the students' programming activities. It is shown that different levels of proficiency in programming activities as well as qualitatively different levels of conceptual understandings are related to dimensions of variation. The dimensions of variation serve as interfaces between the activities and conceptual understandings. If a dimension is discerned, this can facilitate coming to richer conceptual understandings *and* learning additional activities.

1 Introduction

As a teacher, both at upper secondary school and at university level, I have noticed how students learn through their thinking and reflection paired with practical work in the lab. In my teaching practise, it occurred to me that neither "learning by doing" nor "learning by thinking" seemed solely to fully cover the complex process of learning to program.

In this paper I investigate how learning of concepts[1] and learning of practise are related in novice programming students' learning. Practise is discussed in this paper with a focus on learning to *do* practise, while learning *through* practise is discussed merely as a background. The investigation is based on data from two empirical studies. Inspired by some salient findings in the data, the research question posed is:

How are conceptual learning and practise related in programming students' learning process?

As a theoretical framework I use Ways of Thinking and Practising, WTP, as introduced in the ETL project[2] (Entwistle, 2003) and further developed by McCune and Hounsell (2005). WTP highlights the fact that competence in a subject area involves the ability to master certain subject-specific ways of Thinking *and* Practising. WTP draws the attention to necessary understanding and skills in the area. These two aspects are very apparent in the subject area of computer programming, where good conceptual understanding and practical skillfulness are interwoven parts of the learning goals.

I have used a phenomenographic approach (Marton and Booth, 1997) to study students' understanding of fundamental concepts in introductory programming (see Section 5.1 for details). The phenomenographic analysis reveals qualitatively different categories of ways in which the students understand those concepts.

In a subsequent analysis of the practise I have used elements from variation theory (Marton and Tsui, 2004) to study students' learning of common lab activities (see Section 5.2 for details).

The two analyses made it possible to relate qualitatively different conceptual understanding to typical practical activities involved in novice computer programming. My analysis shows that activities at different levels of proficiency relate to different categories of understanding of the concepts through *dimensions of variation*, which serve like interfaces between different conceptual understandings and practises in students' learning.

An analytical model is proposed, where the concept dimensions of variation is used not only in relation to qualitatively different conceptual understandings, but also in relation to different levels of practical proficiencies. In this way the

[1] I will refer to learning of concepts as *conceptual learning* and understanding of concepts as *conceptual understanding* in the rest of the paper.

[2] Information about the ETL project, Enhancing Teaching-Learning Environments in Undergraduate Courses, is found at http://www.etl.tla.ed.ac.uk/ (Retrieved 2008-12-27)

traditional use of phenomenography and variation theory is extended to include not only conceptual understandings, but also the practise, and specifically, how these relate in the learning process. The present empirically based research contributes to a better understanding of the complex relation between conceptual understandings and activities in introductory programming students' learning.

The outline of the paper is as follows: the background of the research is presented in Section 2. This includes a discussion on what WTP means in the programming discipline, a description of the empirical studies, and related work. Section 3 presents the research approaches used in the present research, while the empirical data underpinning the research are discussed in Section 4. Section 5 proposes an analytical model that sheds light on the complex relation between learning of practise and learning of concepts. Section 6 presents conclusions and future work.

2 Background

The goal of the present study is to explore the roles of and relation between Thinking and Practising in novice programming students' learning process. To this end some salient findings from two empirical studies are highlighted by means of the conceptual framework WTP. This section first discusses WTP as it may apply to the programming discipline, then the empirical studies that the research builds upon, and finally research related to the present work.

2.1 Ways of Thinking and Practising in the programming discipline

What does WTP mean and involve when applied to the programming discipline? WTP is a very wide framework. It has been described by McCune and Hounsell in the following way:

> the richness, depth and breadth of what students might learn through engagement with a given subject area in a specific context. This might include, for example, coming to terms with particular understandings, forms of discourse, values or ways of acting which are regarded as central to graduate-level mastery of a discipline or subject area. [...] WTP can potentially encompass anything that students learn which helps them to develop a sense of what it might mean to be part of a particular disciplinary community (McCune and Hounsell, 2005, p. 257)

McCune and Hounsell's discussion implies that the WTP framework involves a whole subject-specific culture. The present research does not aim to fully cover WTP in all programming communities. Some aspects will be discussed, and these aspects will later be related to some aspects of WTP found in the empirical data.

The role of Thinking in the programming discipline

McCune and Hounsell (2005) discuss *Thinking* in the WTP framework in terms of students' learning experiences "through engagement with a given subject area in a specific context" which might include for example "coming to terms with particular understandings, forms of discourse, values" (McCune and Hounsell, 2005, p. 257). Conceptual understanding is one aspect of "coming to terms with particular understandings" and will be the focus of the present discussion of Thinking in computer programming.

Some of the central concepts within the programming discipline are commonly introduced early in programming education and many novice students find them difficult to learn (Eckerdal, 2006).

Thinking, as it is discussed above, appears in all phases of software development for example problem analysis, software design, implementation and testing. This presupposes good understanding of underlying concepts. In object-oriented programming, analysis and design involve for example identifying objects in the problem domain, while implementation means coding the design in a programming language to express the underlying concepts, and testing among other things involves checking the software with reference to these concepts. Thinking in this context thus means the way to think of and understand concepts that enables analysis, design, implementation and tests.

The role of Practising in the programming discipline

The practical side of programming becomes apparent when software development is discussed. Software development is a kind of problem solving process, traditionally divided into several phases: problem analysis, software design, implementation and testing. Programmers often work iteratively among these phases, going back and forth as the work advances. All phases have both practical and theoretical aspects, and some of the practical aspects will be discussed here.

The problem analysis and design phases depend on the character of the problem, and the context where the program will be used. The practise involved in analysis and design requires skills in such diverse areas as programming paradigms and languages, systematic approaches to analysis and design, knowledge of design languages like UML[3] with specific software to produce such design, and hardware knowledge etc.

The implementation and testing phases of software development require for example practical experience in use of several programming languages with appropriate IDEs[4]. The ability to read, write and test code, and knowledge about effective problem solving strategies, are other fundamental aspects of practise important for a programmer to master.

[3]UML, Unified Modeling Language "is a standardized specification language for object modeling. UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model." Retrieved 2007-10-08 from http://en.wikipedia.org.

[4]IDE, Integrated Development Environment, are software tools used for programming development, often integrating for example editor, compiler, and debugger.

In this paper I will focus on certain aspects of Practising relevant for novice students.

Thinking and Practising as ways to abstract a problem
What is then the core of WTP from a computer science perspective? Or more precisely, how does a computer scientist approach a programming problem? A computer scientist thinks in terms of *abstractions* (Kramer, 2007). These abstractions are often of two kinds. *Data structures* are abstract models of something, maybe in reality, or concrete in the sense that they concern basic, efficient ways to store data in computer memory. Abstraction is also required in *algorithms* expressed for example through control structures. When a computer scientist approaches a problem, he or she must distinguish between on the one hand data and data structures, and on the other hand how these can be used in algorithms with the help of control structures. In this way problem solving is to formulate algorithm-resembling systematic procedures, where a problem is abstracted into a computer-oriented solution. Abstraction is carried out through Thinking, as described above, and the results of the process of abstraction are concretised through Practise. Without Practise, software development will remain castle in the air, and without basic conceptual understanding, efficient programming solutions are hardly possible to accomplish. Practise and Thinking are thus interrelated and inevitably interwoven in the process of abstracting and implementing a solution to a programming problem.

2.2 The studies

In order to study the role of Practise and Thinking in novice computer programming learning, two empirical studies have been carried out. In the following, they will be referred to as Study 1 and Study 2, respectively.

The majority of the students who take an introductory programming course at Uppsala University are not computer science majors. Thus, in Study 1 fourteen first year students from a study program in Aquatic and Environmental Engineering were interviewed. The main focus of Study 1 was on students' understanding of central concepts. The interviews were transcribed verbatim and translated to English where necessary. The interviews were analysed mainly with a phenomenographic research approach (Marton and Booth, 1997). Results of analyses performed on data from this study are reported by Eckerdal and Thuné (2005), Eckerdal and Berglund (2005), Eckerdal (2006), and Thuné and Eckerdal (2009).

Study 2 was multi-national and multi-institutional. It aimed to investigate threshold concepts (Meyer and Land, 2005) in computer science. Seven researchers from universities in Sweden, the UK and the USA performed semi-structured interviews with a total of 16 graduating computer science students. The interviews were transcribed verbatim, and were translated to English where necessary. These interviews have been analysed from three different angles. The first analysis aimed to identify threshold concepts in the discipline. The second looked at the parts of the interviews where the students discussed strategies to

get unstuck. The last analysis took a theoretical standpoint. The investigation aimed to analyse what *liminal space* (Meyer and Land, 2005) means and involves in computer science, and thus searched for evidence of several characteristic aspects of such learning experience. Results from these analyses are reported in Boustedt et al. (2007), McCartney (2007), and Eckerdal (2007).

Although both Study 1 and Study 2 where constructed to focus on students' learning and understanding of concepts, the students in both studies talked much about the role of practise in their learning. This is reflected in the publications where McCartney et al. (2007) discuss practise in terms of students' strategies to get unstuck, and where Eckerdal et al. (2007) discuss different theoretical and practical parts of students' learning process.

2.3 Related work

In the computer science community it is generally acknowledged that learning to program requires learning concepts as well as practise. In order to better understand the complex relationship between the two, related work from several related areas has been investigated.

The relation between and nature of Thinking (concepts) and Practising in learning has long been debated and researched by philosophers and others, especially educationalists, interested in learning and learning goals. See Molander (1996, and references therein) for an overview from the area of philosophy. Since my research interest is in computer science education, this section will look at related work from educational research concerning conceptual and practical learning. Section 2.3.1 thus focuses on research on conceptual learning, Section 2.3.2 discusses research on the role of practise in learning, while Section 2.3.3 reports research that relates students' learning of concepts and practise.

2.3.1 Learning the concepts

Higher education and educational research have long had an emphasis on concepts and conceptual learning, and consequently there exists a huge body of research in this area. In the following I will discuss research on conceptual learning with a focus on phenomenography, which is my main research approach, and some research from the conceptual change research tradition.

Phenomenography
There are a number of phenomenographic studies reported in computer science where the focus is on students' understanding of concepts. In 1992 Shirley Booth published her influential thesis "Learning to Program. A phenomenographic perspective" where the main research question was "What does it mean and what does it take to learn to program?". Examples of other phenomenographic work related to programming education are Berglund (2005), where the author discusses senior computer science students' learning of computer systems in a distributed project course. Eckerdal (2006) studied novice programming students' understanding of some central object-oriented concepts, and their use

of resources, while Boustedt (2007) studied senior computer science students working with a large software system, and how the students in this situation understand some advanced object-oriented concepts. Examples of studies that take the computer science teachers' perspective are Carbone et al. (2007), where teachers' understanding of successful and unsuccessful teaching is investigated, and Pears et al. (2007) where the focus is on teachers' experiences of students in trouble and the course content that troubles them.

Conceptual change

Molander et al. (2001) discuss that "a central problem has been to account for the conditions underlying the process of *conceptual change*" (p. 115). According to the authors, this change has "been regarded as an almost unquestioned goal for instruction." (p. 115) Molander et al. refer to Posner et al. (1982) , which they call an "influential article". Posner et al. discuss a theory of conceptual change:

> Learning is concerned with ideas, their structure and the evidence for them. [...] We believe it follows that learning, like inquiry, is best viewed as a process of conceptual change. The basic question concerns how students' conceptions change under the impact of new ideas and new evidence. (Posner et al., 1982, p. 212).

Davies and Mangan (2007) discuss conceptual change in economic education in relation to threshold concepts (Meyer and Land, 2005) and WTP: "[threshold concepts] might best be seen as a web of concepts which link thinking and practise in a discipline."

A recent discussion on conceptual change as it applies to learning and instruction is found in Vosniadou (2007), wherein Entwistle (2007) presents a broad review of research that relates to and has contributed to the conceptual change movement.

2.3.2 Learning the practise

Practise is generally accepted as the most important means to reach the learning goals in computer science education, and good practical skills are seen as the most important learning goal beside good conceptual knowledge.

Examples from computer science education research on the role of practise are studies of students' use of technology based resources like the computer, compilers, editors, and different kinds of software tools. There exists a considerable body of research on such resources in programming education (Valentine, 2004). This research includes discussions on choice of programming paradigm and language, the development of software tools, how students understand them, for example how students understand compiler messages and debugging, and how students use them. For an overview of this line of research, see Eckerdal (2006).

Gross and Powers (2005) discuss novice programmers' learning difficulties saying that teachers "have developed a myriad of tools to help novices learn to program. Unfortunately, too little is known about the educational impact of these environments."

Students' ability to write, read and trace code is researched in some multi-national, multi-institutional studies: McCracken et al. (2001), Lister et al. (2004), and Lister et al. (2006). Students' ability to design is reported in a multi-national, multi-institutional study by Eckerdal et al. (2006). These papers all point to large deficiencies in students' practical skills.

2.3.3 Relating Thinking and Practising

There is little research on the relation between conceptual learning and practise within the subject area of computer science. In the area that Robins et al. (2003) call "psychological/educational study of programming", the complex relationship between conceptual learning and practise has however been recognized by du Boulay (1988) who discusses domains that programming students must learn to master. These include the syntax and semantics of a programming language and different programming skills. du Boulay writes:

> None of theses issues are entirely separable from each others, and much of the 'shock' [...] of the first few encounters between the learner and the system are compounded by the student's attempt to deal with all these different kinds of difficulty at once. (du Bolay, 1988, p. 284).

Rogalski and Samurçay (1990) write:

> Acquiring and developing knowledge about programming is a highly complex process. [...] Even at the level of computer literacy, it requires construction of conceptual knowledge, and the structuring of basic operations (such as loops, conditional statements, etc.) into schemas and plans. (Rogalski and Samurçay, 1990, p. 170)

In addition to the above mentioned research from computer science, I have looked at educational research in science, mathematics, and technology where practical exercises and lab work play significant roles in the education, and where the role of practise, and the relation between practise and concepts, have been researched and debated. I will specifically discuss research from the conceptual/procedural knowledge research tradition and research on students' practical and conceptual learning in science lab since they seem close to what I study.

Conceptual versus procedural knowledge

In mathematics education research that has been inspired by cognitive psychology, there exists a considerable body of research where knowledge is largely divided into two types, conceptual and procedural, which have similarities with the distinction between Thinking and Practising made in the present research. This duality goes back to work by Hiebert and Lefevre (1986), but has been expanded over the years.

Baroody, Fail and Johnson (2007) give a brief overview of the work. They describe, with reference to Star (2005), the two knowledge types, where "each type of knowledge - procedural and conceptual - can either have a superficial or deep quality." (p. 115)

> he proposed defining *conceptual knowledge* as "knowledge of concepts or principles" – as knowledge that involves relations or connections (but not necessarily rich ones). He defined *procedural knowledge* as "knowledge of procedures" and *deep procedural knowledge* as involving "comprehension, flexibility, and critical judgment and [as] distinct from (but possibly related to) knowledge of concepts" (ibid., p 116)

Baroody et al. (2007) further write with reference to Baroody (2003):

> although (relatively) superficial procedural and conceptual knowledge may exist (relatively) independently, (relatively) deep procedural knowledge cannot exist without (relatively) deep conceptual knowledge or vice verse (p. 123)

With reference to the same tradition, Hazzan (2003) discusses how the theory of process-object duality can be used to analyse students' understanding of computer science concepts.

Examples of research in technology education inspired by the same tradition is McCormick (1997). He writes: "technology education, in being concerned with both the practical and the intellectual, offers challenges to learning researchers." (p. 142) He claims that there is a need for researchers to "turn their attentions to the development of strategies for teaching problem solving and design" but "[m]ore complex still, they must show how conceptual knowledge relates to these procedures." (p. 155)

Practising and Thinking in the lab

Hofstein and Lunetta (2003) give a "critical review of the research on the school science laboratory", which follows up a similar study, (Hofstein and Lunetta, 1982). The authors discuss "the effectiveness and the role of laboratory work", that is not "as self-evident as it seemed" (p. 28). In their report from 1982 they write: "The research has failed to show simplistic relationships between experiences in the laboratory and student learning." (Hofstein and Lunetta, 1982, p. 212)

Séré (2002) reports on a project that intended "to address the problem of the effectiveness of lab work" in biology, physics, chemistry and mathematics. Research groups from seven European countries participated, and a number of case-studies were carried out at the upper secondary and undergraduate levels.

Hofstein and Lunetta, as well as Séré, discuss the importance of students' learning concepts and procedures through laboratory work, and both studies point to reported problems with laboratory activities: students do not necessarily learn concepts to the extent teachers expect and hope, and the procedures and technical details cause problems. Both studies also point to a lack of research in science education concerning the role of practise in laboratory work. Séré claims that this is less researched than conceptual learning in lab. Séré's report has its focus on the complex interaction between concepts and practise in laboratory work, and emphasizes the important role of procedures. She comments that this complexity "explains to a certain extent why conceptual learning

is not an automatic outcome of lab work." (p. 630) Hofstein and Lunetta have slightly different focus in their work than Séré. Their focus is on the importance of conceptual learning, that might be hidden when the main emphasis is on procedures and objects manipulated.

Séré's conclusion is that "To do" and "to learn to do" is as important as "to understand" and "to learn" (p. 638). Hofstein and Lunetta emphasize that teachers "need to be able to enable students to interact *intellectually* as well as *physically*, involving hands-on investigation and minds-on reflection" (p. 49, italics in original).

3 Research approaches

In order to better understand the complex relation between Thinking and Practising in programming education, I have re-analysed the empirical data from Study 1 and Study 2, using WTP as an overarching framework. In this section I will present my research approaches to explore Thinking and Practising.

3.1 Researching students' learning of Thinking

To research the role of Thinking in programming education I have focused on one particular aspect of Thinking, namely students' understanding of two central concepts, *object* and *class*. To this end, I have used phenomenography (Marton and Booth, 1997) and variation theory (Marton and Tsui, 2004). Phenomenography is a qualitative research approach which focuses on analysing and describing the variation in how people experience phenomena in the world[5]. A fundamental assumption in phenomenography is that for a given phenomenon there is a limited number of qualitatively different ways in which that phenomenon can be experienced in a certain situation.

Marton and Booth (1997) write about variation in peoples' capabilities for experiencing the world:

> 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 we consider to be the most important kind of learning. (Marton and Booth, 1997, p. 111)

The result of a phenomenographic analysis is an outcome space, with a limited number of categories of description which show a "hierarchical structure of increasing complexity, inclusivity, or specificity" (ibid. p. 126). The categories describe the qualitatively different ways of experiencing the phenomenon

[5] A phenomenographic analysis can only reveal the researchers' interpretation of students' expressed *experiences* of a phenomenon. In the following text I will use *understanding* as interchangeable with *experience* since the present research discusses students' understandings of concepts.

that the researcher has identified in the data. Different categories reflect different combinations of features of the phenomenon which are present in the focal awareness at a particular point in time (ibid. p. 126). Learning is understood as to develop richer ways to see the phenomenon, as represented in the more advanced categories of the phenomenographic outcome space.

Variation and *discernment* are the key words in this process. According to variation theory, which originates from phenomenography, a necessary but not sufficient condition for discerning a specific feature of a phenomenon is that the student gets the opportunity to experience variation in a *dimension* corresponding to that feature. In Thuné and Eckerdal (2009) we explain dimensions of variation in the following way:

> For example, if 'size' and 'colour' are the features of a phenomenon 'picture component', then there is a 'size' dimension and a 'colour' dimension of the corresponding feature space. A particular instance of 'picture component' can be represented by its values in those dimensions, i.e. by its particular size and colour.

There is a possible dimension of variation for each feature of a phenomenon that can take different values. These values thus constitute a dimension, corresponding to the feature. In a learning context, some of these features and their relations are more critical to discern than others. A phenomenographic analysis can identify educationally critical features of the phenomenon studied.

According to variation theory, learning means developing richer ways of seeing a phenomenon, by becoming aware of additional features of it and of relations between the features. This, in turn, requires discerning the dimensions of variation corresponding to these additional features.

3.2 Researching elements of Practise in introductory programming courses

Practise is a wide term that can be used with many different meanings depending on the context. The focus of the present research is learning to *do* practise rather than learning *through* practise.

The students in Study 2 made a clear distinction between practical and non-practical learning goals, see Section 4.2. They discussed learning of programming concepts in terms of achieving *abstract (or theoretical) understanding*, *concrete understandings* (the ability to practise programming without necessarily understanding the underlying theories and concepts), the ability to *go from an understanding of the abstract concept to software design or concrete implementation*, an understanding of the *rationale* for learning and using the concept, and an understanding of how to *apply the concept to new problems*. I discuss practise in terms of the *concrete understanding*, the ability to *go from an understanding of the abstract concept to software design or concrete implementation*, and to be able to *apply concepts to new problems*.

I will further distinguish between *exercises*, *skills*, and *activities*. Exercises are here discussed in terms of practises where the students follow more or less

detailed instructions prepared by the teacher. Exercises are less discussed than the other two, since they represent learning *through* practise.

Skills are practical knowledge students are supposed to acquire during their education. Typical programming skills novice students are expected to learn are to read, to write, and to debug simple computer program code. Each skill is manifested in many different activities that the students are supposed to learn, and these activities demand different levels of proficiency to be properly performed. Skillfulness in programming requires long experience and good theoretical understanding.

For the investigation of Practising in the present study, I have analysed three skills that are often introduced early in the education: to read, to write, and to test and debug code. These skills constitute cornerstones in the implementing and testing phase of software development, see Section 2.1.

The analysis of the practise aims at producing detailed lists of activities that reflect the three skills with the focus on novice students. At the same time this analysis reveals the wide distribution of levels of proficiency inherent in each skill.

The list of activities have been identified in the data, but complemented with activities typically found in text books, explicitly expressed or tacitly presupposed. The reason to look for typical novice activities outside the data is twofold. First, neither of the two studies had an emphasis on learning activities, which limit the discussion on activities found in the data. Second, many activities in programming are tacit and implicit, a kind of taken-for-granted knowledge that is seldom explicitly expressed. The activities can however be identified by for example close examination of examples and figures in text books presenting object-oriented programming for novices.

4 Empirical findings

Before I present the results of the phenomenographic analysis and the analysis on novice programming activities, respectively, some empirical findings are presented to the reader. The following quotes from interviews with students are organised around two identified themes that were salient in the data. Each theme is discussed from a few different aspects that emerged from the data and illuminate the themes.

The first theme, salient in both studies but primarily in Study 1, is that practise plays an important but problematic role in programming students' learning. The data gives evidence that learning of the practise is as important to research as learning of the concepts. This is discussed in Section 4.1.

The second theme is the complex relation between practise and conceptual learning in the students' learning process, which was specifically apparent in Study 2. This is discussed in Section 4.2.

In this way this section provides empirical evidence that students' learning of concepts and practise need to be researched simultaneously. Consequently the subsequent Section 5 demonstrates a way to research this relation by combining

the phenomenographic results and the results of the analysis of common novice students' activities.

The students from Study 1 will in the following section be referred to as Student A, Student B, Student C (A, B, C) etc, with no reference to their real names, while the students in Study 2 will be referred to as Student 1, Student 2, Student 3 (S1, S2, S3) etc.

4.1 The important but problematic role of Practise in programming education

Three aspects of this theme will be discussed. First we establish that practise is not merely a tool to reach the conceptual learning goals, but is part of the learning goals. Second, practise, in terms of exercises, is the generally accepted, and often presupposed, main road to learning programming in computer science education. Third, data from Study 1 and Study 2 indicate that students do not learn through the exercises to the extent they are expected to. This applies both to the conceptual and the practical learning.

First aspect: Practise as part of the learning goals
There is agreement in industry and among educators that profound practical knowledge is a central goal in programming education, and that this often requires many years of hands-on training. The novice students in Study 1 and the senior students in Study 2 are well aware of the important role of the practise.

The students in Study 1 were asked about what learning to program means. All students expressed that learning to program is experienced as learning to understand some programming language, and using it for writing code. This emphasises the practical use of a programming language. In addition, Student D emphasises the ability to read, trace and debug code:

> D:[...] 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.

Second aspect: Practise as means to reach the learning goals
Practise as a way to learn is emphasised by students in both studies. Student 7 in Study 2 emphasises the importance of learning step-by-step procedures, or "templates" for doing things. This "de-mystifies" the concept and helps the student to reach a better understanding. The student discusses pointers which is generally regarded as a hard concept to learn, even a threshold concept:

> S7: [...] when I got the idea of this format, this template for creating nodes and node datas and how they can go together and kind of a step by step then it de-mystifies it actually. I can see if I missed a step. "Oh, dummy, you forgot to hook these two together," and maybe that's what in a lot of programming with practise you mentally get these steps in your mind.

Third aspect: Problems with the practise hinder further learning
The third aspect, salient in the data from both Study 1 and Study 2, highlights the problematic role of practise in students' learning. If the students have problems with learning *through* practise, conceptual learning as well as learning to do practise might be hindered. A few examples of this are shown below.

Student D in Study 1 answers the question what has been difficult in the course. In the quote below he or she expresses problems in learning concepts as well as in mastering the practise, "to understand [...] how one should use different things in a program" and "how to study, when you implement the programs". The student seems to look for the meaning of the practise, which seems to be a major road block in the learning process:

> 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 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.

Student D further discusses the contrast between learning mathematics and learning computer programming:

> D: I've taken many math courses but math is kind of logical and you understand it but this is... no I don't know. [...] 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 basis from the beginning [...]

The examples student D talks about are provided by the teacher and intend to show some of the basic practises that are needed in programming: how to structure a program, how some data structures are implemented and used in algorithms through control structures etc. Still the student has not discerned the practise sufficiently to know "how to study", as he or she puts it in the quote above.

There was evidence also in Study 2 that the practise can be a major obstacle in learning to program. Student 7 in Study 2 emphasises that it is the practise that is the problem in the learning:

> S7: There's just some aspects to it that just seem to remain kind of mysterious to me at the programming level. Not the concept level, not the theory level, not the technology level but at the kind of code nuts and bolts level.

Practise is, according to many students, the main road or doorway into further learning in programming. But learning through programming exercises, implies an enormous theoretical, practical and technological challenge for many novice students. In fact, the first threshold novice students often encounter involves a variety of different sorts of practise, and without knowledge of how to master the practise, there might be no way into further learning, neither

conceptual nor practical. Practise is not merely the unproblematic road to the more advanced conceptual understanding.

4.2 The complex relation between Thinking and Practising in programming students' learning

Practise plays an important and often inevitable role in conceptual learning. This section discusses the second theme found in the data which deals with the complex relation between practise and conceptual learning in students' learning to program. Three aspects of the theme are discussed. The first aspect points to the close relation between concepts and practise in the learning which makes the novices discuss learning to program as learning a new way of thinking. The second aspect highlights the senior students' descriptions where distinct "parts" of the learning process have been identified. The third aspect shows that there is not one linear route through the learning process. There are rather individual routes, where different students get stuck at different places.

First aspect: The magic "Programming Thinking"
In both studies there was evidence that novice students found learning to program hard. Some students even experienced programming as "magic", or even frightening, in the beginning of their learning.

Strikingly many novice students in Study 1 discussed learning to program in terms of learning a special way of thinking. Inspired by the data, we introduced the term *programming thinking* to describe this phenomenon, different from, and for many novice students with troubling little coherence with, thinking in other subjects with which they had previous experience (Eckerdal and Berglund, 2005).

A quote from Student D in Study 1, which was cited in Section 4.1, is relevant also in this context. The student talks about his or her problem to learn to program:

> 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

Student A expresses this when answering the question what is most important in the course:

> A: It's more the thinking itself, the logical thinking. Everything you need to know you must think of when it comes to programming. [...] 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 D connects the problems associated with this way of thinking with conceptual learning as well as the practise "how one should use different things in a program". Student A moreover points to the problem of understanding the hardware, the computer itself, "how the computer works".

The novice students' descriptions of "programming thinking" illustrate how learning to program involves both conceptual learning and practise. Furthermore they show how closely related, and how dependent on each other the two aspects are. The rest of this section will focus on the senior students in Study 2.

Second aspect: Parts of the conceptual and practical learning space
As the novices, the senior students discussed the complex relation between concepts and practise when learning to program, but unlike the novices the senior students could articulate such experiences in detail. For many novice students, the learning experience was "magic" and fragmented, while many senior students explicitly described crucial "parts" of the learning process.

Eckerdal et al. (2007) analysed the data from Study 2 in order to identify specific characteristics for computer science students who are in the midst of learning a threshold concept (Meyer and Land, 2005). The analysis was inspired by Meyer and Land's description of the *liminal space* which is a space in which someone is transformed, acquires new knowledge, and acquires a new status and identity within a community. We found that the students discussed the learning process of threshold concepts in terms of acquiring different distinct parts of a full understanding. The different partial understandings found are:

- abstract (or theoretical) understanding
- concrete understanding (the ability to practise programming without necessarily understanding the underlying concepts)
- the ability to go from an understanding of the abstract concept to software design or concrete implementation
- an understanding of the rationale for learning and using the concept
- and an understanding of how to apply the concept to new problems.

The senior students clearly pointed to theoretical as well as practical learning goals.

A few quotes from the students will illustrate some of the understandings expressed above.

Student 8 expresses the third understanding, the need of both an abstract understanding of a concept and the ability to relate it to implementation, which requires practise:

> S8: [...]the abstract understanding is something you learn by education, by reading, you can learn that in class, but the understanding of actually applying it to programs you can't, you must, you must learn it by, by, by using it

The last understanding in the list above, "an understanding of how to apply the concept to new problems", mirrors that students distinguish relating abstract concepts to implementation of routine problems, and applying concepts to *new* problems. The latter is experienced as specifically problematic for some students. This is expressed by Student 2:

> S2: You understand how a theory works but how do you take that theory and how it works and apply it to a practical sense? I think that is one of the hardest leaps to make.

Third aspect: Individual routes in a practical and theoretical learning space

In the analysis of students in the liminal space (Eckerdal et al., 2007) we found that the different "parts" of the learning process were discussed by the students as inseparable facets of one and the same learning experience. Different students struggled with different parts, depending on, for example, background knowledge. Some students expressed that they first learned the theory behind the concepts, and then how to use them, while other students expressed the opposite. Some discussed that the major problem was to understand the rationale behind some concepts, while other students did not mention that aspect at all. The observed differences in the students' learning processes need however to be further researched before we can draw any deeper inferences from them.

The data clearly indicated that there is no common route for the students through the learning space. Rather a complex pattern of individual learning routes appeared. Still, both concepts and practise seem important, and both can cause problem in the learning process.

This section has discussed some salient themes in the empirical data: practise and concepts are both parts of the learning goals and dependent of each other in the learning process. They mutually support each other in the learning, but at the same time either of them can become an obstacle for further learning. They are complexly interwoven in the learning process, and play different roles depending on the individual. For some novice students this was experienced as an alien and magic "programming thinking" which was difficult to grasp. The senior students could articulate a complex learning space where both concepts and practise played important roles. In the following section, the data is re-analysed to get a better understanding of how Thinking and Practising relate in students' learning of computer programming.

5 Ways of Thinking and Practising - an analytic model

I will now continue with an in-depth analysis of WTPs from the novice programming students' perspective using data mainly from Study 1. The aim is to analyse how Thinking, here discussed as students' understanding of concepts, and Practising, here discussed as common novice students' programming activities, are related. First however, Thinking and Practising are discussed separately. Based on these two discussions, I will then demonstrate how variation theory can be used to analyse and display the relation between Thinking and Practising in programming students' learning process.

5.1 Thinking manifested in students' understanding of concepts

To investigate Thinking in the form of conceptual understanding, we made a phenomenographic analysis of novice students' understanding of the central concepts *object* and *class*, based on data from Study 1. The two concepts object and class are closely related, and the analysis resulted in one phenomenographic outcome space, see Table 1. The categories in the outcome space are descriptions of the qualitatively different ways in which the concepts object and class are seen by the group of interviewees on a collective level. These categories are inclusive which means that the understandings described in the latter categories include the understandings described in the former. The latter categories thus reflect richer ways to understand the concepts.

Having formulated the phenomenographic outcome space summarised in Table 1, we made a subsequent analysis to identify dimensions of variation related to the two concepts considered here. As discussed in Section 3.1, possible values of a certain feature of a phenomenon constitute a dimension of variation for this phenomenon. In order to identify dimensions of variation we consequently re-examined the categories of description in Table 1, to find what features of the two concepts that are in focus in the different understandings expressed. For a more comprehensive presentation of the phenomenographic analysis where the dimensions of variation are identified, see Eckerdal and Thuné (2005).

Below I discuss and label the different dimensions of variation identified.

Object is experienced as a piece of program text, and class as an entity of the program that structures the code and describes the object.
Object and class are experienced as expressed in the previous category. In addition, class is experienced as a description of properties and behavior of objects, and object as something that is active when the program is executed.
Object and class are experienced as expressed in the previous category. In addition, class is experienced as a description of properties and behavior of objects, where the object is a model of some real world phenomenon.

Table 1: A phenomenographic outcome space on novice students' understanding of the concepts *object* and *class*. For details, see Eckerdal and Thuné (2005) and Eckerdal (2006).

In the first category of description in Table 1, the feature in focus is the textual representation of the concepts. I will refer to the corresponding dimension of variation as TEXT.

In the second category, the new feature added is the active behaviour when the program is executed. I will refer to the corresponding dimension of variation as ACTION.

The new feature described in the third category is the modeling aspects of the concepts, and I will refer to corresponding dimension of variation as MODEL.

5.2 Practising manifested in common students' activities

This section aims at analysing the role of practise in novice programming students' learning. The analysis of the practise is not a traditional phenomenographic analysis, but an analysis based on elements from variation theory on common and important novice students' programming activities.

In Study 1 and Study 2 we found students discussing different skills that are important when learning to program. Examples of such skills, typical for novice programming are to write code, read code, test code, debug code, design software, and use advanced technical resources, for example different IDEs.

In this section I will focus on a few of these skills namely to read, to write, and to test and debug code. To test and debug code will be treated as one skill since they are closely connected. The skills focused on in the present discussion are often seen as the most fundamental skills in programming and essential for novice programming students to learn.

The skills are manifested in several more or less advanced activities. Aiming at relating the activities and students' understanding of concepts, I will first list such activities that are important and frequently occurring in novice programming courses, see Table 2. In addition to the activities mentioned by the students in the data from Study 1 and Study 2, activities commonly found in text books are added.

Although some activities do not cause problems for most students, they are still included to demonstrate the breadth of new activities and tools novice students meet and are expected to learn and use at an early stage of their education. The detailed list is in line with suggestions from science education research, where Séré writes: "A first step in research should now be to describe what happens during lab work as exhaustively as possible." (Séré, 2002, p. 628). A detailed list concretises what practise means in novice students' programming. The list above covers common novice programming activities reasonably well for the present discussion.

Why do some students have such big problems in performing some activities? Activities carry meaning and it is important that students discern this meaning. Runesson (2006), with reference to Svensson (1984), discusses meaning in relation to learning:

> [meaning] is constituted as a relation between the object to which I direct my awareness and me, the subject. The meaning emerges as I direct my awareness to the object. (Runesson, 2006, p. 400)

Runesson (2006) furthermore explains the relation between meaning and dimensions of variation:

Read code: to discern main parts of short programs; to read code and recognize key words; to read code and understand what will happen when the instructions are executed; to read and relate code to the application and the problem domain.
Write code: to use an editor to emphasis the structure of a program by means of indents, empty lines etc.; to write common programming building blocks in a syntactically correct way; to design a short algorithm; to express a short algorithm in pseudo code; to implement pseudo code in a programming language; to design a solution to a whole problem and transfer the design to pseudo code, using common programming building blocks; to implement the solution to a problem according to basic software quality requirements.
Test and debug code: to use a compiler to find and correct minor syntax errors; to use the computer to execute code to verify expected output; to use a compiler to get executable code; to read and understand error messages about simple syntax errors, such as missing semicolon; to correct simple syntax errors, for example missing semicolon; to hand execute a program on paper before coding; to diagnose semantic errors in the code; to test code in relation to the problem domain and usability.

Table 2: Common novice programming skills with associated activities.

> the meaning we assign something is constituted as a pattern of simultaneously discerned dimensions of variation. (Runesson, 2006, p. 403)

In my interpretation of practise, an activity is the object towards which the awareness is directed. To discern the meaning of a certain activity the student thus has to simultaneously be focally aware of certain dimensions of variation related to this activity.

To elaborate on this argument I will discuss the activities that are related to the skill *read*, and leave to the reader to extrapolate the discussion to the activities related to the other skills. The activities "to discern main parts in short programs", and "to read code and recognize key words" relate to the text-representation of the code. To discern the meaning of these activities, the students need to become aware of the TEXT dimension of variation discussed in the previous section. On the other hand, in order to master the activity "to read code and understand what will happen when the instructions are executed" the students need to become aware of the ACTION dimension of variation in addition to the TEXT dimension. Finally, the activity "to read and relate code to the application and the problem domain" requires that the students discern not only the TEXT and the ACTION dimensions of variation, but also the MODEL dimension. It is necessary in this discussion to say that there might be other dimensions of variation related to the activities mentioned that have

not come to the foreground in this analysis, but I claim that the students need to discern at least these dimensions.

Activities related to the TEXT dimension of variation discern main parts of short programs; read code and recognize key words; use an editor to emphasize the structured of a program by means of indents, empty lines etc.; use the computer to execute code to verify expected output; write common programming building blocks in a syntactically correct way; use a compiler to find and correct minor syntax errors; use a compiler to get executable code
Activities related to the TEXT and ACTION dimensions of variation read code and understand what will happen when the instructions are executed; design a short algorithm; express a short algorithm in pseudo code; implement pseudo code in a programming language; hand execute a program on paper before coding; diagnose semantic errors in the code
Activities related to the TEXT, ACTION, and MODEL dimensions of variation read and relate code to the application and the problem domain; test code in relation to the problem domain and usability; design a solution to a whole problem and transfer the design to pseudo code, using common programming building blocks; implement code according to basic software quality requirements

Table 3: Common novice programming activities at different levels of proficiency. The activities can be experienced as meaningful when related dimensions of variation are discerned.

This discussion can contribute to explain students' problems with reading code as reported by Lister et al. (2006). They conclude that students need to be able to for example manually trace code, which requires that the TEXT and the ACTION dimensions have been discerned, according to the above reasoning. This is however not sufficient "if they are to develop as programmers." (ibid. p. 122). The students furthermore need to be able to "read several lines of code and integrate them into a coherent structure - to see the forest, not just the trees." (ibid. p. 122). The activity described in this quote probably requires that the MODEL dimension, as well as the ACTION and TEXT dimensions have been discerned, and many novice students have problems reaching this level of proficiency. Similar discussions can be made for all activities in Table 2.

In Table 3 the activities are re-grouped according to different combinations of dimensions of variation. Students can become aware of the meaning embedded in an activity if the dimensions of variation related to the activity are discerned. By studying Table 3 it becomes visible that activities that relate to more dimensions of variation correspond to *a higher level of proficiency* than the activities that relate to fewer dimensions.

5.3 Developing an analytical model for relating Thinking and Practising

The discussions in Section 5.1 and Section 5.2 show that both conceptual understandings and activities are related to dimensions of variation. There are qualitatively more advanced ways to understand the concepts that relate to more dimensions of variation. In similar ways there are activities at higher level of proficiency which relate to more dimensions of variation. Table 4 merges Table 1 and Table 3 and illustrates that the dimensions of variation are in the center of the analysis, relating conceptual understandings and activities to each other.

The structure of Table 1 and Table 3 is reflected in Table 4. The left column includes the categories of the phenomenographic outcome space from Table 1. The activities listed in Table 3 are found in the right column. In the middle column are the dimensions of variation, related to the qualitatively different conceptual understandings as well as to the activities at different levels of proficiency.

The relations described in Table 4 between the activities at different levels of proficiency and the different understandings of the concepts *object* and *class* are examples of how Practise and Thinking are related, and a different example could have been chosen to make my point. The significant implication is that *both* to discern a certain feature of a concept and to make an activity meaningful require that certain dimensions of variation in the learning space is opened for the student. It is the dimensions of variation that are at the center of this discussion.

The empirical findings indicate that when students have reached a certain level of practical proficiency, this can help them in their learning of new concepts, and that understanding of concepts can help them to master new practise, see Section 4.2. These empirical findings are in line with Table 4 where the dimensions of variation are at the center. The learning of concepts *and* activities presupposes that related dimensions are discerned. Once a dimension of variation is discerned, this can help the students to understand related concepts (Marton and Tsui, 2004) *and* to learn related activities.

Table 4 illustrates my analytical model where the dimensions of variation are at the center of the discussion on students' learning. Figure 1 is a more abstract and general illustration of the model. It shows that several activities as well as several concepts can be related to a certain dimension of variation. This finding is explicitly shown in Table 4. Moreover, Figure 1 shows that activities as well as concepts can be related to more than one dimension of variation.

An example of this is that we found both the TEXT and the ACTION dimensions in the phenomenographic analysis of the students' experiences of what computer programming means (Thuné and Eckerdal, 2009). These dimensions are also found in the present analysis of the same students' understanding of the concepts object and class, and the dimensions are further related to programming activities at certain level of proficiency. An inference of this finding is that the student can discern for example the ACTION dimension either through a re-

Understanding class and object as	Dimensions of variation	Typically novice students' activities that are examples of the skills read, write, and test and debug code
Object: piece of program text. Class: entity of the program that structures code and describes the object.	TEXT	- discern main parts of short programs - read code and recognize key words - use an editor to emphasize the structure of a program by means of indents, empty lines etc. - execute code to verify expected output - write common programming building blocks in a syntactically correct way - use a compiler: find and correct minor syntax errors - use a compiler to get executable code
In addition, object: active during execution. Class: describe properties and behaviour.	TEXT and ACTION	- read code and understand what will happen when the instructions are executed - design a short algorithm - express a short algorithm in pseudo code - implement pseudo code in programming language - hand execute program on paper before coding - diagnose semantic errors in the code
In addition, object: model real world phenomenon. Class: describes properties and behaviour	TEXT, ACTION and MODEL	- read and relate code to the application and the problem domain - test code in relation to the problem domain and usability - design a solution to a whole problem and transfer the design to pseudo code, using common programming building blocks - implement code according to basic software quality requirements

Table 4: Categories describing students' understanding of the concepts object and class (left column), and novice students' activities at different levels of proficiency (right column) are related to dimensions of variation (middle column).

lated activity (for example one of those mentioned in Table 4), or when studying the concepts object and class, or when learning what computer programming means, or through learning other concepts or activities. Furthermore, once the student has discerned the dimension, the discernment can help the student to learn the other concepts and activities that are related to this dimension.

The present work has identified that practise as well as concepts have related dimensions of variation. Fazey and Marton (2002) discuss dimensions of variation related to practise. The authors summarise a number of studies on systematic variation of practising motor skills saying:

> What comes out of these examples for us is not simply that varying practise conditions can have positive, longer term effects for learners, but also that there are several dimensions along which practise conditions might

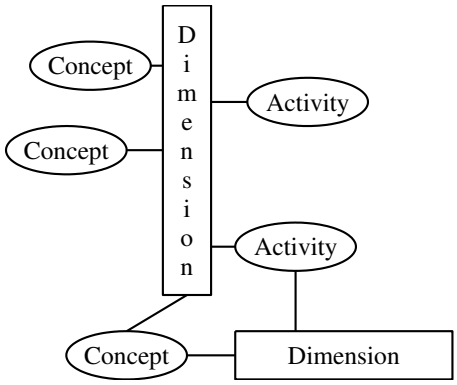

Figure 1: The figure illustrates two dimensions of variation of a learning space. Empirical findings indicate that the dimensions can act as interfaces between different conceptual understandings and activities. When a dimension is discerned, this can open for learning concepts *and* practises in new ways.

> vary. Some of these are embedded [...] in the way learners are exposed to variations in practise. (ibid. p. 244).

The new significant finding in the present research is however that practise and conceptual understandings are related through *common* dimensions of variation.

Table 4 indicates that depending on the dimensions of variation discerned by the student, it is likely that the student can learn activities (practises) related to only those dimensions, since it is then possible for the student to see the meaning of the activities. In a similar way, some ways to understand the concepts are possible for the student to discern. The higher level of practical proficiency and the more advanced level of conceptual understanding, the more features, or the more dimensions of variation and their relations, the student needs to discern.

In conclusion Figure 1 demonstrates the complex relation between Thinking and Practising in the learning process, where individual students take different routes. Students can discern a dimension of variation when studying concepts *or* working with the activities. When a dimension is discerned, this opens the possibility for a wider conceptual understanding *and* for learning related activities. This is in line with the finding that some students express that they have discerned the activities first, and then the concepts, while other students express the opposite learning experience.

6 Conclusions and future work

The research presented in this paper demonstrates the complex relationships and mutual dependence between novice programming students' conceptual and practical learning. I have used Ways of Thinking and Practising, WTP, as a theoretical framework in the investigation of this relationship. In the present work

Thinking refers students' conceptual understandings, while Practising refers to common novice students' programming activities. In this context the relation between Thinking and Practising means how conceptual learning and learning of programming activities depend on each other and mutually carry meaning to each other and make learning possible or hindered.

Empirically the research builds on two interview studies with computer science students. For the analysis of the data I have primarily used phenomenography and variation theory.

The main findings are the following:

- The practise is experienced by many students as equally, or even more troublesome to learn than the concepts. Furthermore it is shown that students experience programming activities and conceptual understanding as equally important. If they face problems with one of them, they are likely to face problems with the other.
- An analytical model of students' learning is proposed that demonstrates that activities as well as conceptual understandings relate to dimensions of variation.
- Higher level of practical proficiency relate to more dimensions of variation in a similar way as more advanced ways to understanding concepts relate to more dimensions of variation.
- The most significant finding in the present research is that I have demonstrated that practise and conceptual understandings have dimensions of variation *in common*. This has been possible since I have showed a way to identify dimensions of variation related to practises.
- Consequently, when a dimension of variation is opened for a student, this creates an opportunity for the student to learn concepts *and* activities in new ways.

These results can explain how students' learning sometimes seems to go from concepts to practise, and sometimes from practise to concepts. In addition this explains that when a certain concept *or* activity is learned, it can open up for learning new concepts and activities, related to those already learned, via the dimensions of variation.

Moreover, the results of the present research can to some extent explain why activities, performed for example in the lab, do not necessarily lead to deepened conceptual understanding, and why studying of concepts do not necessarily lead to a higher level of skillfulness in programming education. If the student in the learning situation does not discern the related dimensions of variation, there might be no corresponding conceptual and practical learning.

Phenomenography and variation theory (Marton and Booth, 1997; Marton and Tsui, 2004) traditionally discuss ways to identify critical features of concepts,

and ways to open a space of learning for students by means of patterns of variation in the teaching. The present work contributes to the body of knowledge of students' learning by proposing an analytical model of how dimensions of variation relate to conceptual and practical learning. To use the model as a basis for further empirical studies about the relation between practise and concepts in a learning context would be an interesting line of future research.

Acknowledgment

I want to thank my supervisor Professor Michael Thuné for his support during the research process; in rewarding discussions and for constructive ideas, and for his patience and help during the process of writing the paper.

References

Baroody, A. (2003). The development of adaptive expertise and flexibility: The integration of conceptual and procedural knowledge. In Baroody, A. and Dowker, A., editors, *The development of arithmetic concepts and skills: Constructing adaptive expertise*, pages 1–34. Erlbaum, Mahwah, NJ.

Baroody, A., Feil, Y., and Johnson, A. (2007). An alternative reconceptualization of procedural and conceptual knowledge. *Journal for Research in Mathematics Education*, 38(2):115–131.

Berglund, A. (2005). *Learning computer systems in a distributed project course. The what, why, how and where.* Number 62 in Uppsala Dissertations from the Faculty of Science and Technology. Acta Universitatis Upsaliensis, Uppsala, Sweden.

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

Boustedt, J. (2007). *Students Working with a Large Software System: Experiences and Understandings.* Licentiate thesis, Uppsala University, Uppsala, Sweden.

Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., and Zander, C. (2007). Threshold concepts in computer science: do they exist and are they useful? *SIGCSE Bulletin*, 39(1):504–508.

Carbone, A., Mannila, L., and Fitzgerald, S. (2007). Computer science and it teachers' conceptions of successful and unsuccessful teaching: A phenomenographic study. *Computer Science Education*, 17(4):275–299.

Davies, P. and Mangan, J. (2007). Threshold concepts and the integration of understanding in economics. *Studies in Higher Education*, 32(6):711–726.

du Bolay, B. (1988). Some difficulties of learning to program. In Soloway, E. and Spohrer, J., editors, *Studying the Novice programmer*, pages 283–299. Lawrence Erlbaum Associaties Inc.

Eckerdal, A. (2006). *Novice Students' Learning of Object-Orientd Programming.* Licentiate thesis, Uppsala University, Uppsala, Sweden.

Eckerdal, A. and Berglund, A. (2005). What Does It Take to Learn 'Programming Thinking'? In *Proceedings of the 1st International Computing Education Research Workshop, ICER*, pages 135–143, Seattle, Washington, USA.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliff, M., and Zander, C. (2006). Categorizing student software designs: Methods, results, and implications. *Computer Science Education*, 16(3).

Eckerdal, A., McCartney, R., Moström, J. E., Sanders, K., Thomas, L., and Zander, C. (2007). From Limen to Lumen: Computing students in liminal spaces. In *Proceedings of the 3rd International Workshop on Computing Education Research*, pages 123–132. ACM.

Eckerdal, A. and Thuné, M. (2005). Novice java programmers' conceptions of "object" and "class", and variation theory. *SIGCSE Bulletin*, 37(3):89–93.

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.

Entwistle, N. (2007). Conceptions of learning and the experience of understanding: Thresholds, contextual influences, and knowledge objects. In Vosniadou, S., Baltas, A., and Vamvakoussi, X., editors, *Re-framing the Conceptual Change Approach in Learning and Instruction*, pages 123–143. ELSEVIER.

Fazey, J. and Marton, F. (2002). Understanding the space of experiential variation. *Active Learning in Higher Education*, 3(3):234–250.

Gross, P. and Powers, K. (2005). Evaluating assessments of novice programming environments. In *Proceedings of the 1st International Computing Education Research Workshop, ICER, Seattle, Washington, USA*, pages 99–110.

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

Hiebert, J. and Lefevre, P. (1986). Conceptual and procedural knowledge in mathematics: An introductory analysis. In Hiebert, J., editor, *Conceptual and procedural knowledge: The case of mathematics*, pages 1–27. Erlbaum, Hillsdale, NJ.

Hofstein, A. and Lunetta, V. (1982). The role of the laboratory in science teaching: Neglected aspects of research. *Review of Educational Research*, 52(2):201–217.

Hofstein, A. and Lunetta, V. (2003). The laboratory in science education: Foundations for the twenty-first centry. *Science Education*, 88(1):28–54.

Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42.

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. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150.

Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C., and Whalley, J. L. (2006). Research perspectives on the objects-early debate. In *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 146–165, New York, NY, USA. ACM.

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

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

McCartney, R., Eckerdal, A., Mostrom, J. E., Sanders, K., and Zander, C. (2007). Successful students' strategies for getting unstuck. *SIGCSE Bulletin*, 39(3):156–160.

McCormick, R. (1997). Conceptual and procedural knowledge. *International Journal of Technology and Design Education*, 7(1–2):141–159.

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.

McCune, V. and Hounsell, D. (2005). The development of students' ways of thinking and practising in three final-year biology courses. *Higher Education*, 49:255–289.

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(3):373–388.

Molander, B. (1996). *Kunskap i handling*. DAIDALOS.

Molander, B., Halldén, O., and Pedersen, S. (2001). Understanding a Phenomenon in Two Domains as a Result of Contextualization. *Scandinavian Journal of Educational Research*, 45(2):115–123.

Pears, A., Berglund, A., Eckerdal, A., East, P., Kinnunen, P., Malmi, L., McCartney, R., Moström, J., Murphy, L., Ratcliffe, M., Schulte, C., Simon, B., Stamouli, I., and Thomas, L. (2008). What's the problem? teachers' experience of student learning successes and failures. *Proc. 7th Baltic Sea Conference on Computing Education Research: Koli Calling, CRPIT, Australian Computer Society*, 88.

Posner, G., Strike, K., Hewson, P., and Gertzog, W. (1982). Accommodation of a scientific conception: toward a theory of conceptual change. *Science Education*, 66(2):211–227.

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

Rogalski, J. and Samurçay, R. (1990). Acquisition of programming knowledge and skills. In Hoc, J., Green, T., Samurçay, R., and Gillmore, D., editors, *Psychology of programming*, pages 157–174. Academic Press.

Runesson, U. (2006). What is it Possible to Learn? On Variation as a Necessary Condition for Learning. *Scandinavian Journal of Educational Research*, 50(4):397–410.

Star, J. R. (2005). Reconceptualizing procedural knowledge. *Journal for Research in Mathematics Education*, 36:401–411.

Svensson, L. (1984). Människobilden i INOM-gruppens forskning: Den lärande människan. [The image of man in research in the INOM-group; in Swedish]. Technical report, Göteborgs universitet, Institutionen för pedagogik, Sweden.

Séré, M. (2002). Towards renewed research questions from the outcomes of the european project *Labwork in Science Education*. *Science Education*, 86(5):624–644.

Thuné, M. and Eckerdal, A. (2009). Variation Theory Applied to Students' Conceptions of Computer Programming. *European Journal of Engineering Education*, Accepted for publication.

Valentine, D. (2004). CS educational research: a meta-analysis of SIGCSE technical symposium proceedings. *SIGCSE Bulletin*, 36(1):255–259.

Vosniadou, S., Baltas, A., and Vamvakoussi, X. (2007). *Reframing the Conceptual Change Approach in Learning and Instruction (Advances in Learning and Instruction)*. Elsevier.