UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 600

Novice Programming Students' Learning of Concepts and Practise

ANNA ECKERDAL

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2009

ISSN 1651-6214
ISBN 978-91-554-7406-5
urn:nbn:se:uu:diva-9551

Dissertation presented at Uppsala University to be publicly examined in room 2446, Polacksbacken, Uppsala, Friday, March 6, 2009 at 10:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract
Eckerdal, A. 2009. Novice Programming Students' Learning of Concepts and Practise. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 600. 76 pp. Uppsala. ISBN 978-91-554-7406-5.

Computer programming is a core area in computer science education that involves practical as well as conceptual learning goals. The literature in programming education reports however that novice students have great problems in their learning. These problems apply to concepts as well as to practise.

The empirically based research presented in this thesis contributes to the body of knowledge on students' learning by investigating the relationship between conceptual and practical learning in novice student learning of programming. Previous research in programming education has focused either on students' practical or conceptual learning. The present research indicates however that students' problems with learning to program partly depend on a complex relationship and mutual dependence between the two.

The most significant finding is that practise, in terms of activities at different levels of proficiency, and qualitatively different conceptual understandings, have dimensions of variation in common.

An analytical model is suggested where the dimensions of variation relate both to concepts and activities. The implications of the model are several. With the dimensions of variation at the center of learning this implies that when students discern a dimension of variation, related conceptual understandings and the meaning embedded in related practises can be discerned.

Activities as well as concepts can relate to more than one dimension. Activities at a higher level of proficiency, as well as qualitatively richer understandings of concepts, relate to more dimensions of variation.

Concrete examples are given on how variation theory and patterns of variation can be applied in teaching programming. The results can be used by educators to help students discern dimensions of variation, and thus facilitate practical as well as conceptual learning.

Keywords: Computer science education, computer science education research, object-oriented programming, novice students, phenomenography, variation theory, dimensions of variation, learning, higher education, concepts, practise, Ways of Thinking and Practising

Anna Eckerdal, Department of Information Technology, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

ISSN 1651-6214
ISBN 978-91-554-7406-5
urn:nbn:se:uu:diva-9551 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-9551)

To my children Per, Nils, and Olof

List of Papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Eckerdal, A., Thuné, M. (2005) Novice Java Progammers' Conceptions of "Object" and "Class", and Variation Theory. *SIGCSE Bulletin*, 37(3), pp.89–93

II Eckerdal, A., McCartney, R., Moström, J.E., Ratcliffe, M., Sanders, K., Zander, C. (2006) Putting Threshold Concepts into Context in Computer Science Education. *SIGCSE Bulletin*, 38(3), pp. 103–107

III Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Ratcliff, M., Sanders, K., Zander, C. (2007) Threshold Concepts in Computer Science: do they exist and are they useful? *SIGCSE Bulletin*, 39(1), pp. 504–508

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

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

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

VII Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Zander, C. (2006) Categorizing Student Software Designs: Methods, results, and implications. *Computer Science Education*, 16(3), pp. 197–209.

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

IX Eckerdal, A. (2009) Ways of Thinking and Pratising in Introductory Programming. Technical Report 2009-002, Department of Information Technology, Uppsala University, Sweden.

Comments on my contributions

In this section I list my main contributions for the papers included in this thesis.

I I planned and performed the data gathering and the writing, but in close discussion with the second author. Both authors separately analysed the data, and discussed the results until we came to an agreement.

II The six authors contributed equally to the background studies and writing of the paper.

III Data for this paper was gathered at three occasions. The first gathering was performed by four of the seven authors, me included. The second by two of the authors, not including me, and the last and most important and time consuming data gathering was planned and performed by all seven authors. The data analyses and writing were performed jointly by the seven authors of the paper.

IV I planned and performed the data gathering, but in close discussion with the first author. We analysed the data jointly. The first author outlined and wrote the paper, in discussion with me.

V I planned and performed the data gathering and the analysis, suggested the topic and wrote the paper, but in close discussion with the second author.

VI Data for this study was gathered by six researchers, where I was one. Five of the researchers, me included, performed the data analysis and writing jointly. The idea for the research came from one of the other authors.

VII Data for this paper was gathered by twentyone researchers. A subgroup of four researchers, me included, together with an additional researcher analysed a partial set of the data. The analysis and writing was jointly performed by the five authors of the paper.

VIII Data for this study was gathered by six researchers, where I was one. Five of these, plus a new member of the group are authors of the paper. I was the initiator of the paper. I suggested the topic, the analysis method, and outlined the structure of the paper. The data analysis the research builds on and the writing was performed jointly by the authors.

IX I am the sole author of this paper.

Contents

1. Introduction

This thesis addresses the role of concepts and the role of practise in computer programming students' learning. Specifically, the relationship between practise and concepts in students' learning process is investigated. The research is empirically based on studies with students from several countries.

Computer science has conceptual[1] as well as practical learning goals (Roberts and Engel, 2001). Many computer programming students claim that "learning through practise" is by far the best way to learn to program, both regarding "learning to do practise" as regarding learning the concepts (Eckerdal, 2006). This claim seems to be supported by many educators, considering the huge amount of papers that have been published on how to help students to "learn through" and "learn to do" practise (Valentine, 2004; Gross and Powers, 2005). Still, after decades of attempts to improve the learning outcome (from an early reference from the childhood of programming education in 1969[2] it is reported about a "high withdrawal rate" from a course), mainly by focusing on the role of the practise, serious learning problems seem to prevail (Eckerdal et al., 2006b; Fleury, 2000; Fleury, 2001; Lister et al., 2004; McCracken et al., 2001; Robins et al., 2003). In contrast to this focus on practise, research in higher education in general has had a strong focus on students' conceptual learning (Entwistle, 2003; Entwistle, 2007; Molander, 1996; Molander et al., 2001; Posner et al., 1982).

The primary contribution of the present research lies in the investigation of the relationship between students' conceptual and practical learning, where the focus of the latter is on "learning to do" practise. First the role of practise and the role of concepts are investigated separately. It is shown that students have great difficulties in learning both concepts and practise. The investigations further strongly indicate complex relationships and mutual dependencies between practise and concepts in students' learning process. The results thus point to a need to further explore this relation. This has also previously been pointed to in science and technology education research (Séré, 2002; McCormick, 1997).

The thesis also contributes to the body of knowledge on learning in higher education by presenting an analytical model of how the learning of concepts and practise relate in novice programming students' learning.

The present research builds on three empirical investigations. Data from the investigations have been analysed from several different perspectives in order

[1] By conceptual learning goals I mean subject specific *concepts* students are supposed to learn, see Section 1.2.

[2] Retrieved 080812 from http://portalparts.acm.org/880000/873609/fm/frontmatter.pdf

to provide insights into students' learning of concepts as well as their learning of the practise.

There are two objectives of this thesis. The first is to contribute to the body of knowledge of the learning process in programming education. The second is pragmatic: to give concrete advice to programming educators on how teaching and learning can be improved.

1.1 Research questions

As a teacher in computer science my interest in students' learning led me to research on teaching and learning. A first investigation revealed novice students' understanding of some central concepts. At the same time the data showed the first traces of a complex relationship between practise and conceptual learning. One of the novice students in the study comments the course he or she had just finished, a first programming course:

> 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. (Eckerdal and Berglund, 2005, p. 138)

This student finds it difficult to learn the concepts, but what he or she emphasises is problems with "learning through practise", the same practise that is developed to help students learn the concepts. The novice students in this study specifically stressed the importance of practise in learning the concepts. But if "learning through practise" causes the largest problem, the students will neither learn "to do the practise", nor the concepts. Keeping in mind the substantial efforts in the last decades to improve "learning through practise" to facilitate novice students' learning (Valentine, 2004; Gross and Powers, 2005), it becomes obvious that practise is not merely the unproblematic road to conceptual learning.

The subsequently performed investigations established the important but problematic role of practise in programming education and led to my overarching research question:

- Which roles do practise and concepts play in programming students' learning process?

This question is broad, and to investigate all aspects of it is beyond the scope of a thesis. I have thus limited the question. To this end I have used a conceptual framework, Ways of Thinking and Practising, WTP (Entwistle, 2003; McCune and Hounsell, 2005), to embrace both the practical and the conceptual aspects of learning to program, and how these two are related.

The research questions highlighted in this thesis focuses on three themes related to the WTP framework. The first theme concerns students conceptual learning:

- How do novice programming students understand programming concepts?
- How do novice students understand what computer programming means, and how do they understand what *learning* to program means?
- How can the results from a phenomenographic outcome space inform teaching and thus improve learning in computer programming education?

The second theme concerns the role of practise in computer science education:

- What strategies do computer science students use when they are stuck in their learning?
- Can computer science students design software?

The third theme deals with how conceptual learning and practise are related in the learning process:

- How do students experience the process of learning threshold concepts, the so called *liminal space* (Meyer and Land, 2005) in computer science?
- How do practise and concepts relate in novice programming students' learning?

The first theme is discussed in Paper I through Paper V, the second in Paper VI and Paper VII, and the third theme is the topic of Paper VIII and Paper IX. The papers are separately described in Section 1.4.

1.2 Terminology used in the thesis

The focus of the thesis is to discuss the role of concepts and the role of practise in programming students' learning. My use of the word *concept* is broad. I will discuss concept as "an abstract or generic idea generalized from particular instances", and as "any idea of what a thing ought to be"[3]. Some of the concepts discussed in the thesis are lexical, word-sized, for example "class" and "object" while others are broader, for example "computer programming".

Practise is a broad term. In Paper IX I distinguish between skills, activities and exercises. Programming students are supposed to learn new practical skills like reading, writing, and debugging 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. For example the skill of reading code can, for a novice student, mean the recognition of key words in a program, while at a higher level of proficiency reading code implies being able to relate the code to a problem domain. Exercises on the other hand are here discussed in terms of practises where 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", while the focus of this thesis is on "learning to do the practise".

Practise, in terms of exercises, is the main means to reach both conceptual and practical learning goals, for example reading, writing, and debugging

[3]Retrieved 090110 from Merriam-Webster Online Dictionary, http://www.merriam-webster.com/dictionary/

code. Computer programming thus involves "learning through the practise" as well as "learning to do the practise".

The rest of this section explains some computer specific terms used in the thesis. Other computing terms are defined when they are introduced in the text.

Computer science is a discipline in higher education which involves methods and theories underlying computers and software systems. *Software* comprises 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.

The software development process is traditionally divided into several phases: problem analysis, software design, implementation and testing. Computer programming, which is sometimes used synonymously with implementation and sometimes in the broader sense as software development, is a core area in computer science. When *implementing*, the programmer writes *code* in a certain programming language, where code refers to the instructions which tell the computer what to do. These instructions follow rules from the particular programming language used. *Syntax* is the description of the possible combinations of symbols and specific words that are accepted in a programming language.

Software development involves use of certain software tools. For example, in the implementation and testing phase specific text *editors* are used that are developed to facilitate the implementation by, for example, recognising the syntax of the language. The editor and the compiler are often integrated in a development environment. The *compiler* is the software that translates (compiles) the code to a representation that is executable for a computer. When the code is tested it is checked to determine if it meets the requirements given. This involves *debugging* the code, which means finding and removing errors.

There exist 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 currently dominant in industry and university education. Examples of programming languages within the object-oriented paradigm are Java and C++.

Meyer (1988) describes the thoughts behind the *object oriented paradigm*:

> 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).

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 the features that underpinned the development of the object-oriented paradigm (Meyer, 1988).

1.3 Methodology

My interest in understanding the student learning process, which appeared so difficult to penetrate, led me to investigate students' learning of concepts and practise, as presented in Section 1.1. The research aims to give a broad picture of students' learning experiences, emanating from the students' perspectives. Students experience learning as a whole, and in order to untangle the complex experience, several studies were performed. In this section this is described as three different investigations, although they together form the pool of empirical data that the research builds upon, and from which conclusions are drawn.

The data in the first investigation are interviews with novice programming students. The second investigation involves several data collections, including informal interviews and a questionnaire administrated to educators, and interviews with senior students. The third investigation includes a large set of data from senior students' doing a design task. In this way, data showing students' understanding of concepts as well as the role of practise in programming education were gathered.

1.3.1 The first investigation

The first investigation included in the present research is a study with 14 Swedish first year non-major computer science students. It is common at Swedish universities that non-major computer science students in technical and natural science education take at least one computing course where they are given an introduction to programming. The students had just finished their first programming course, using Java as the programming language. The aim of the investigation was to get a rich description of the variation in the students' different experiences of some concepts in object-oriented programming. The students were thus interviewed for example on their understanding of the concepts *object* and *class*, and what it means to learn to program. The answers to these questions were transcribed verbatim and translated to English where needed. The analysis was performed using a phenomenographic research approach, see Section 3.

The research questions informed by this study are how novice students understand what programming is and what learning to program means, how they understand central concepts in the object-oriented paradigm, and how the results from a phenomenographic outcome space can inform teaching. Furthermore data from the first investigation informed the question on how conceptual learning and practise relate in programming students' learning process.

1.3.2 The second investigation

The second investigation was performed by a group of researchers from Sweden, the United Kingdom, and the United States. The work was motivated by an interest in threshold concepts in computer science (Meyer and Land, 2005). Two pre-studies were performed with educators at two international conferences during the summer and fall of 2005. Educators were informally interviewed, and some answered a questionnaire. The aim was to find threshold concept candidates for further investigation. These two studies laid the foundation for an interview study with students, aiming at identifying threshold concepts from the students' perspectives. A subsequent multinational study with students from seven universities in the three countries was performed during spring 2006. 16 graduating computer science students were interviewed.

The interviews have been analysed from three different perspectives and inform the following research questions. The first analysis aimed at identifying threshold concepts in the discipline. The second analysed the parts of the interviews where the students discussed strategies for getting unstuck in their studies. The last analysis took a theoretical standpoint, aiming at investigating what *liminal space* means and involves in computer science. The theory of liminal space was used as a tool in the search for learning experiences characteristic of computer programming. Furthermore data from the second investigation informed the question on how conceptual learning and practise relate in programming students' learning process.

1.3.3 The third investigation

A multinational study was performed by 21 researchers at 21 institutions in the United States, the United Kingdom, Sweden, and New Zealand. This study involved 314 participants from three levels of education; students with low competence, graduating seniors, and educators (Tenenberg et al., 2005). The research presented in the present thesis was performed by a subgroup of the original 21 researchers, plus one researcher not participating in the original investigation. The data used for this research were software designs produced by a subset of the participants, the 149 near-graduation seniors. The participants were asked to design a "Super alarm clock" according to a number of criteria that were to be met. Beside these criteria, there was little guidance on how to perform the task. The designs were made on paper.

This investigation informs the research question on graduating computer science students' ability to design.

1.4 Overview of the thesis

As explained above, the papers in the thesis are organized around three themes. Concepts and practise are two inseparable and equally important learning goals in programming education. The themes thus focus on student

learning of concepts, student learning of practise, and the relationship between the two in students' learning process.

The first theme on student learning of concepts is discussed in the first five papers of the thesis.

The second theme, dealing with students' learning practise, is illuminated in Paper VI and Paper VII.

The last theme, how conceptual learning and learning practise are related in students' learning process is examined in Paper VIII and Paper IX.

1.4.1 Student learning of concepts

Student learning of concepts is researched at different levels of granularity. The analysis from a bird's eye view discusses students' understanding of what computer programming means, while the analysis at the next level aims at identifying central, threshold concepts in computer programming. Finally, the analysis that focuses primarily on details look at students' understanding of a few, possible threshold concepts. Below follows a description on the papers that belong to this theme.

At the most coarse-grained level Paper IV, *Variation Theory applied to Students' Conceptions of Computer Programming*, investigates students' understanding of the whole subject area, computer programming. This is followed in Paper V, *What does it take to learn 'programming thinking'?*, by an investigation of the same students' understanding of what *learning* computer programming means.

At the next level of granularity, Paper II, *Putting Threshold Concepts into Context in Computer Science Education*, identifies so called threshold concepts in computer science. Further Paper III, *Threshold Concepts in Computer Science: Do they exist and are they useful?*, investigates students' learning of such concepts.

Finally, at the most fine-grained level Paper I, *Novice Java Programmers' Conception of "Object" and "Class" and Variation Theory* presents an in-depth study of students' understanding of a few central concepts, concepts that are possible threshold concepts in object-oriented programming.

1.4.2 Student learning of practise

Learning computer programming concerns learning practical skills. In the present thesis Paper VII, *Categorizing Student Software Designs: Methods, results, and implications*, focuses on one specific skill, software design. Design is, beside writing code, reading code, and debugging code, considered as a core skill in programming education. The investigation on senior students' ability to design software is an important contribution to the body of knowledge of students' skillfulness.

Our investigation, together with related projects (McCracken et al., 2001; Whalley et al., 2006; Fitzgerald et al., 2008), all point to the conclusion that students have great problems in learning the practise. The learning outcome

in programming education has been argued to be closely related to good programming strategies (Robins et al., 2003; Davies, 1993). We have thus investigated such strategies in terms of what graduating students do when they are stuck in their learning. This line of research is presented in Paper VI, *Successful students' strategies for getting unstuck*. Some of the strategies identified and labeled in the paper have an abstract character, like "Be persistent/don't stop" or "See patterns". Others have a more concrete, practical nature, for example "Use a [software] tool", "Write programs" or "Trace [code]". Many of the strategies found in the analysis are thus related to the practical aspect of programming. In this way Paper VI emphasis the importance of students' learning practise and broadens the research presented in Paper VII which focuses on one particular aspect of practise, students' ability to design.

1.4.3 The relationship between conceptual and practical learning

The last theme presented in the thesis focuses on the complex relationship between conceptual and practical learning. The theme is highlighted by results from Paper I *Novice Java Programmers' Conception of "Object" and "Class" and Variation Theory*, Paper IV *Variation Theory applied to Students' Conceptions of Computer Programming*, and Paper V *What does it take to learn 'programming thinking'?* with novice students, but is established and elaborated in Paper VIII *From* Limen *to* Lumen*: Computing students in liminal spaces*, with senior students. The results of this analysis reveals a broad and rich picture of the students' learning experiences where the practise as well as the concepts play important but problematic roles in the students' learning process.

In this way Paper VIII gives a background for the analysis presented in Paper IX, *Ways of Thinking and Practising in Introductory Programming*. The paper is the synthesis of my thesis work. Important results from the first two investigations on conceptual and practical learning are discussed and further developed. The focus, discussed and analysed in depth, is however on how conceptual and practical learning relate in students' learning process.

2. The research in context

2.1 The Computer Science Education research field

Computer science[1] is a young discipline, only half a century old. As a discipline of its own, computer science education is even younger. Computer science has developed with an "astonishing pace" which has had "a profound effect on computer science education, affecting both content and pedagogy." (Roberts and Engel, 2001, Chapter 2) The rapid change of the subject matter taught has inevitably affected also the computer science education research discipline. The discipline has however encountered several problems. Berglund (2005) identifies some of them. First, the discipline is cross-disciplinary. It encompasses computer science, but in addition a range of other disciplines including pedagogy, psychology, learning technology, and more. According to Berglund, the lowest common denominator in this diverse field is "the *aim to improve learning and teaching within computer science,* and thereby to contribute to computer science." (Berglund, 2005, p. 23, italics in original) This is in line with the aim of the present research.

Berglund further points to the problem of knowing "who is 'in' the community." He writes, with reference to Clancy et al. (2001):

> As many of the leading researchers within the field are better known for their contribution to other sub-areas of computer science, it is also hard to determine where the edges of the community are. (Berglund, 2005, p. 23)

Another problem recognized, relevant for the present thesis, is that there has been, and still is a need of more qualitative research in computer science education research (Berglund et al., 2006). Berglund et al. (2006, p. 25) claim that "research into student learning is strengthened by increased awareness of the role and relevance of qualitative research approaches in CER."

A question that has been discussed in the CER community, and still is an issue, is how to define research in computer science education. What distinguishes research in teaching and learning from mere ideas of good teaching practise based on personal teaching experiences? This is debated for example in Goldweber et al. (2004), where one of the authors writes: "CSEd research is new. It

[1] Computer science is commonly abbreviated CS. Accordingly, computer science education is abbreviated CSEd or CSE, and computing education research CER.

co-exists in places with other sorts of publications (like SIGCSE) and where it starts and stops, where its edges are, are not yet clear."[2]

Fincher and Petre discuss how computer science education research has emerged as an "identifiable area" (Fincher and Petre, 2004, p. 1) during the past decades. The growth has come from different places like computer science practitioner conferences, sub-specialist areas like psychology of programming, and computer science research groups at different academic institutions. Another factor contributing to the shattered picture is the contributors, who have diverse expert knowledge like education, psychology, and different areas of computer science, and consequently have published in different research fora. Fincher and Petre write about this sprawling research field: "Despite this growth–and because of it–we are struggling to find the shape and culture of our literature." (p. 2)

Fincher and Petre discuss the characteristics of the publications that can be referred to as research: "they can be thought of as having two components: a dimension of rationale, argumentation or 'theory', and a dimension of empirical evidence." (p. 2)

The research presented in this thesis is well in line with the two criteria discussed by Fincher and Petre. All the papers build on empirical data (except Paper II, *Putting Threshold Concepts into Context in Computer Science Education*, which is a literature review) and they all include arguments, or theories, which the interpretations and inferences build on. Furthermore, all papers are published in well established fora, where the papers have been peer-reviewed by relevant specialists in computer science and/or education.

2.2 The present research and the Computer Science Education research field

Students' learning of computer science has been investigated from different perspectives. This section will put the present research in a context of research in computer science education, and specifically regarding research on students' learning computer programming which is a sub-field of the wider computer science education research field.

Pears et al. (2007) report on a literature survey on teaching of introductory programming. The following areas are investigated in the survey: Curricula, Pedagogy, Language choice, and Tools for teaching.

Randolph (2007) presents, from a positivistic perspective rooted in psychological research, a major overview of articles in computer science education. The author reviewed 352 computer science education articles published between 2000 and 2005. Randolph claims among other things that "several dif-

[2]SIGCSE mission statement, http://sigcse.org/about/, says: "The ACM Special Interest Group on Computer Science Education provides a forum for educators to discuss issues related to the development, implementation, and/or evaluation of computing programs, curricula, and courses, as well as syllabi, laboratories, and other elements of teaching and pedagogy."

ferences in research practises across the fields of computer science education, educational technology, and education research proper were found." (p. iv) Randolph furthermore found that one third of the articles reviewed "did not report research on human participants" and most of them "were program descriptions" (p. 173).

An older survey is by Austing et al. (1977) who report on literature in computer science education from the publication of the first ACM Computing Curricula 1968 (ACM Curriculum Committee on Computer Science, 1968), up to 1977, including for example survey reports, descriptions of programs, and descriptions of courses and other material.

A psychological/educational perspective on learning is the focus of Robins et al.'s review (2003) which compares "novice and expert programmers, programming knowledge and strategies, program generation and comprehension, and object-oriented versus procedural programming." (p. 137) Robins et al. specifically focus on "novice programming and topics relating to novice teaching and learning." (p. 137)

Simon (2007) summarises the range of different types of publications in computer science education. He presents an overview of classification of papers in the field that have been published in different fora. For example, Pears et al. (2005) present a classification which, with reference to Fincher and Petre (2004), suggests the following areas for computer science education *research*: studies in teaching, learning, and assessment; institutions and educational settings; problems and solutions; computing education research as a discipline.

The focus of the present research is on learning, namely programming students' learning of concepts and practise. Computer programming is one of the core areas in computer science education, which is established in the influential ACM/IEEE Computing Curricula 2001 (Roberts and Engel, 2001)[3]. Even though computer programming is a young discipline in higher education, students' difficulties are widely reported in the literature (Ben-Ari, 1998; Eckerdal and Thuné, 2005; Fleury, 1999; Fleury, 2000; Lister et al., 2004; McCracken et al., 2001; Robins et al., 2003).

The present thesis work is put in a research context below in the following way: first I present studies on students' conceptual understanding, which include questions on student understanding of single concepts as well as questions at a more coarse-grained level including student understanding of what computer programming is. Studies on student learning of practise is discussed from two perspectives. First, studies that investigate practise as a learning goal in terms of programming skills are presented. Then, studies investigating practise as a means to reach learning goals, conceptual as well as practical, are discussed. Because of the many published articles related to the psycholog-

[3]This curriculum is one in a series of curricula developed for computer science education, the first dating back to 1968 (ACM Curriculum Committee on Computer Science, 1968).

ical/educational study of programming, I will finally briefly touch upon this area of research, although it is not within the scope of the present thesis.

Student learning of concepts

As described in Section 1.4 the thesis presents research on student learning of concepts at different levels of granularity. At the most coarse-grained level are Paper IV, which discusses novice students' understanding of computer programming, and Paper V, which discusses the same students' understanding of what *learning* computer programming means.

Examples of research related to these questions are Booth (1992), who in her influential thesis investigates what it means and what it takes to learn to program, and Bruce et al. (2004) and Thuné and Eckerdal (2009), (Paper IV), who follow this line of research, studying students' understanding of what programming means. Similar research is presented by Eckerdal and Berglund (2005), (Paper V), and Stamouli and Huggard (2006) who investigate students' understanding of what learning to program means. The studies show very similar findings. Students' understandings vary from a narrow language-syntax-centered understanding to more desirable broader understandings including programming as problem solving, a skill that can be used outside of computing education.

Many studies point to the necessity of a good understanding of the central concepts within object-oriented programming. Ragonis and Ben-Ari (2005) present a long-term study on high school students' learning of concepts in object-oriented programming including "class vs. object, instantiation and constructors, simple vs. composed classes, and program flow. In total, 58 conceptions and difficulties were identified." (Ragonis and Ben-Ari, 2005, p. 203) 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. In a multinational study Sanders et al. (2008) investigated what novice object-oriented programming students see as the most important concepts, and how they express the relationships among those concepts. Some results from the study are that "[u]nlike earlier research, we found that our students generally connect classes with both data and behavior" (p. 332), but "few students see modeling as one of the most important OO concepts." (p. 336) Another multinational study, presented by Sanders et al (2005), involved 20 researchers and 276 participants from 20 different institutions. The study aimed to elicit novice object-oriented programmers' knowledge of programming concepts by using a "multiple, participant-defined, single-criterion card sort". The authors point to "the unexpected result that there were few discernible systematic differences in the population." (p. 121)

Examples of studies on students' conceptual understanding with a phenomenographic approach are Berglund (2005) who investigated senior students' understanding of concepts within computer systems, Boustedt (2007) who studied senior students' understanding of some advanced object-oriented

concepts, and Eckerdal and Thuné (2005), (Paper I), who investigated novice students' understanding of central object-oriented concepts.

Holmboe (1999) emphasises that good understanding in programming requires both practical skills and conceptual understanding, and a connection between the two. This mirrors the three foci of the present thesis. The following two sections will discuss the role of practise in computer science education.

Learning programming skills

There exists a considerable body of research on the role of practise in computer science education, both as means to reach the learning goals, and as a goal in itself. The latter is discussed in this section, in terms of skills students are supposed to learn.

A well known multinational study is McCracken et al. (2001) who investigated novice students' ability to write code. The authors concluded that many students can not program after their first introductory programming course, but lacked evidence for an explanation. Lister et al. (2004) continued the McCracken study and found that students' problems with programming "relate more to the ability of students to read code than to write it." (p. 139) This line of research has been extended by Whalley et al. (2006), who also study students' ability to read code. The authors found that "[s]tudents who cannot read a short piece of code and describe it in relational terms are not well equipped intellectually to write code of their own." (p. 251) In the same line of research is Lopez et al. (2008) who investigated the relationship between reading, tracing and writing code in novice students' learning. The authors found correlation between performance on "code tracing tasks" and "performance on code writing tasks" and also between "performance on 'explain in plain English' tasks and code writing." (p. 101)

Students' ability to debug code is investigated in a multinational study by Fitzgerald et al. (2008). The authors found that students that can debug are often good novice programmers, but the opposite does not always apply. On the other hand, "once students find bugs, they can fix them." (p. 93) Senior computer science students' ability to design software is investigated in a multinational study by Eckerdal et al. (2006), (Paper VII). The authors found that that only 9 % of the students produced partial or complete designs. We furthermore found that the number of academic courses taken by the students, the time the students spend on the design task, and the number of programming languages well known by the students were significantly correlated with the result of the design task. In summary, all the studies point to students' difficulties in learning the practical skills. This applies to novices as well as to senior students.

Practise as means for learning to program

Practise is often seen as an inevitable means to reach learning goals. Resources that enhance practise for learning are frequently discussed topics in conference papers and journal articles. Such a resource which is expected to have

high impact on object-oriented educations is the Java Task Force that was appointed by the ACM Education Board in 2004[4]. The mission was to develop a s collection of pedagogical resources that would support the use of Java in first-year computer science courses.

There is a strong focus on technology based learning support in the computer science education literature. This is pointed to by Valentine (2004) who did a meta-analysis on twenty years of proceedings from the largest conference in computer science education. The author categorized research papers dealing with beginning programming courses. During 1994-2003, 42% of the number of papers in the proceedings described software that was developed by the author of the paper to enhance learning.

Technology supported resources developed to enhance learning to program are discussed by Powers et al. (2006). According to the authors software resources developed to help novices to learn to program can be divided into several groups, for example *Narrative tools*, which "support programming to tell a story" and *Visual programming tools*, which "support the construction of programs through a drag-and drop interface". An example of the former is Alice (Powers et al., 2007) and an example of the latter is JPie (Goldman, 2004).

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.

How do we know that technology based learning resources lead to good conceptual or practical learning? Gross and Powers (2005) performed an extensive literature search for assessments of the educational impact of novice programming environments. The authors relate their literature search to 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."

Pair programming has been greatly discussed in the computer science community during recent years. 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). Studies on the learning outcome of pair programming, and how pairs best are selected have been performed. Examples of this are VanDeGrift (2004) and Katira (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 practises also include other factors like pair programming and programmers' collective ownership of the code in the system (Beck and Andres, 2004).

[4]The reports from the Java Task Force with associated material are available from http://jtf.acm.org/index.html Retrieved November 18, 2008.

Other aspects of the practise discussed in the literature are the role of projects and programming assignments, and the roles of the programming language and programming environment. The former are discussed for example by Daly (2004) and Newman (2003). The latter are discussed in for example Kölling (1999a) and Kölling (1999b) where the author discusses where different programming languages and different programming environments are suitable.

Psychological/educational study of programming

As a contrast to my own research I will mention two large research areas in computer science education: research on students' misconceptions, and research comparing novices and experts behaviour. These research areas are close to the present research, but not the exact focus.

Robins et al. (2003) discuss "literature relating to the psychological/educational study of programming." The authors discuss "general trends", for example regarding comparison of novice and expert programmers. Examples from this line of research are Gugerty and Olson (1986) who compare expert and novice debuggers, Kahney (1983) who investigate novices' and experts' understanding of recursive procedures, Zou and Godfrey (2008) who investigate differences between newcomers' and experts' interaction with software development tools, and Winslow (1996) who, based on an overview of psychological research in programming pedagogy, claim that it takes 10 years of experience to turn a novice programmer into an expert.

Students' misconceptions are frequently reported in studies on students learning to program. I will mention a few. Ragonis and Ben-Ari (2005) present a large study with high school students learning to program. The article includes detailed lists of difficulties and misconceptions related to several concepts in object-oriented programming. 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." Sanders and Thomas (2007) describe a close examination of student programs from an introductory programming course, in which they found evidence of misconceptions. Among other things they found difficulties in distinguishing between classes and objects, and in modelling.

Other programming paradigms are also discussed in this context. Spohrer and Soloway (1986) studied novice Pascal students and investigated "whether or not most novice programming bugs arise because students have misconceptions about the semantics of particular language constructs." (p. 183) The authors found that for most of the bugs investigated that was not the case. Bayman and Mayer (1983) report on a study on beginning BASIC programmers misconceptions of statements they had learned, and Fung et al. (1990) report on "novices' misconceptions about the interpreter in Prolog" (p. 311).

3. Research approaches

My research interest is in programming students' learning, and specifically in the students' own experiences of their learning. I first investigated students' experiences of some programming concepts, and how they went about learning the concepts. Aiming at describing this from the students' perspective the first two investigations mainly used interviews with students. Interviews with educators and a brief survey were initially used in the second investigation, but the results from initial analyses pointed to qualitative, student centered research methods, and consequently interviews with students were performed.

From the initial research questions, the data and the analyses led to new research questions concerning the role of practise in programming students' learning, but still related to how the students experience their learning.

The research questions presented in Section 1.1 suggest a predominantly qualitative research approach since the focus is on how-questions, see Section 3.1 below. Quantitative methods have been used to a minor extent, and only as a complement to a primary, qualitative approach.

In the present section I will briefly introduce the reader to qualitative research in general, and in particular to phenomenography and variation theory. Parts of the content analysis tradition will be discussed, namely qualitative content analysis, which has been applied in the present research. In addition, trustworthiness in qualitative research is introduced, and discussed in relation to the present work.

3.1 Qualitative research

Qualitative research is spread widely and cross cuts many disciplines, using a variety of methods and approaches. Denzin and Lincoln (2005) discuss the development of qualitative research. Considering its complex development, qualitative research is difficult to define. The authors still offer an "initial, generic definition" (p. 3):

> qualitative research involves an interpretive naturalistic approach to the world. This means that qualitative researchers study things in their natural settings, attempting to make sense of, or interpret, phenomena in terms of the meanings people bring to them. (Denzin and Lincoln, 2005, p. 3)[1]

[1] In the following text I will refer to this definition when I discuss the naturalistic paradigm.

According to Denzin and Lincoln (1994) qualitative researchers "seek answers to questions that stress how social experience is created and given meaning" in contrast to quantitative studies which "emphasize the measurement and analysis of causal relationships between variables, not processes." (Denzin and Lincoln, 1994, p. 4) Qualitative studies often focus on *Why?* and *How?* questions, less on *How much?* which is common in quantitative studies. The aim of qualitative research is rather to give "thick descriptions" of phenomena than to measure variables. To "make sense of" and look for "the meaning people bring" to phenomena are watchwords.

Examples of data collection methods used in qualitative research are participant observation and video recordings, but as Denzin and Lincoln (2005) write: "No specific method or practise can be privileged over any other." (p. 7). Empirical materials used involve for example interviews, artifacts, and historical texts "that describe routine and problematic moments and meanings in individuals' lives." (Denzin and Lincoln, 2005, p. 3-4)

The research questions (see Section 1.1) require analysis methods that can elicit the meaning embedded in the material. Content analysis is such a method that has been used on interviews and written artifacts to elicit meaning by categorisation. Phenomenography, here applied in the analyses of interviews, is another approach, used in educational research to understand differences in learning outcome by eliciting qualitatively different ways in which people experience phenomena.

3.2 Phenomenography

Phenomenography is a qualitative research approach, intended for educational research. 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 aiming at understanding the differences in students' understandings. 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 on learning 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, which focuses on describing and understanding the variation in how people experience phenomena in the world[2] *Phenomena* are described by Marton and Booth (1997) as the units that exceed a situation, bind it together with other situations and give it a meaning.

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

[2]In the following text I will use *understanding* as interchangeable with *experience* since the present research discusses students' understandings of phenomena.

> 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 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 understandings, 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 (1979) claim that in this perspective, the world as the student experiences it, becomes visible. The experience is a relation between the student and his or her world, it is not two independent descriptions, one of the student and one of the world. "[W]e have one description which is of a relational character." (Marton and Svensson, 1979, p. 472)

In phenomenographic studies, data are often gathered in the form of interviews where people are encouraged to describe their different experiences, or understandings of some phenomenon. The interviews are transcribed verbatim and the data, as text, are analysed. The analysis aims at identifying different understandings of the phenomenon discussed. 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 which form an outcome space. 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.

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 categories in the outcome space show a "hierarchical structure of increasing complexity, inclusivity, or specificity" (Marton and Booth, 1997, p. 126). The categories describe the qualitatively different ways of experiencing the phenomenon 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 (Marton and Booth, 1997, p. 126). Marton, Runesson and Tsui (2004, p. 22) describe critical features: "the features that must be discerned in order to constitute the meaning aimed for."

The phenomenographic 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.

Learning is understood as developing richer ways to see a phenomenon, as represented in the more advanced categories of the phenomenographic outcome space. Variation theory, which originates from phenomenography, emphasises *variation* and *discernment* as key words in this process. 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 Paper IV we explain *dimension of variation*, or for short dimension, 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.

Each feature of the phenomenon studied that appears in an outcome space corresponds in this way to a dimension. Marton, Runesson and Tsui (2004, p. 21) discuss the need to create a space, which means "opening up a dimension of variation (as compared to the taken-for-granted nature of the absence of variation)." The authors describe such a space:

> A *space of learning* comprises any number of dimensions of variation and denotes the aspects of a situation, or the phenomena embedded in that situation, that can be discerned due to the variation present in the situation. [...] [The space] delimits what can be possible learned (in sense of discerning) in that particular situation. (Marton et al., 2004, p. 21) (Italics in original)

3.3 Content analysis

Content analysis is described by Mostyn (1985) as "a very ordinary, everyday activity we all engage in [...] when we draw conclusions from unstructured communications" (p. 115) Content analysis originally dealt with quantitative analysis of data (*what*-, *where*-, and *how many*- questions) but has developed to include qualitative analysis (*why*-questions).

Qualitative content analysis as a research method deals with analysing artifacts, often texts, with focus on the content and meaning embedded in the text. The goal of qualitative content analysis is to understand the meaning of unstructured communication, and through a process of condensing raw data into categories come to a better understanding of the phenomenon studied. This

process involves inferences and interpretations that require knowledge of the context and subject studied. Hsieh and Shannon (2005) define qualitative content analysis in the following way:

> a research method for the subjective interpretation of the content of text data through the systematic classification process of coding and identifying themes or patterns (Hsieh and Shannon, 2005, p. 1278)

Mostyn describes qualitative content analysis as "the 'diagnostic tool' of qualitative researchers" (p. 117). As such, content analysis is used in a variety of research methods including discourse analysis, ethnographic research, and computer text analysis (Krippendorff, 2004, p. 19).

The object of interest in qualitative content analysis is often a text, for example transcribed interviews, but data can equally come from observing behavior, artifacts, etc. (Mostyn, 1985, p. 124). To gain insights into the meaning embedded in the data, the analysis requires interpretation that goes beyond inference. Mostyn writes:

> we become concerned with content as a reflection of deeper phenomena. Words are treated as symbols and the data has attributes of its own; we are analyzing both manifest and latent data. (p. 116)

Graneheim and Lundman (2004) describe manifest as "what the text says [...] the visible, obvious components", while latent data is described as "what the text talks about [...] an interpretation of the underlying aspects of the text". (p. 106)

The researcher scrutinizes the data, looking for regularities "in terms of single words, themes, or concepts." (Mostyn, 1985, p. 118) This in-depth analysis leads to identification of *categories*, which is the heart of content analysis.

Mayring (2000) describes content analysis as "a bundle of techniques for systematic text analysis". He specifically describes two approaches that offer procedures for data analysis. The first is *inductive category development*. It is described as a "reductive" process:

> the material is worked through and categories are tentative and step by step deduced. Within a feedback loop those categories are revised, eventually reduced to main categories [...]

In this way, data can be categorised with an explorative approach. The categories are developed as the researcher delves deeper into the data and lets the data speak. The categories developed during the process guide the researcher in his or her interpretations and inferences in the analysis.

The second approach is *deductive category application*. Mayring writes: "Deductive category application works with prior formulated, theoretical derived aspects of analysis, bringing them in connection with the text."

Meaning embedded in the data is, with this approach, unveiled when parts of the data are fitted into pre-existing categories or theories, and the result

subsequently is interpreted. The researcher uses his or her knowledge of the data in terms of the participants and context of the data gathering in the process of interpretations and inferences.

3.4 Trustworthiness in Qualitative Research

Evaluation of research and its trustworthiness depends on the research paradigm used. This is due to the fact that different research paradigms have different knowledge claims. Lincoln and Guba (1985) write that "criteria for what counts as significant knowledge vary from paradigm to paradigm." (Lincoln and Guba, 1985, p. 301)

Lincoln and Guba contrast criteria related to what they call the conventional paradigm, where most research in the area of computer science belongs, with criteria appropriate for the naturalistic paradigm, where the present, qualitative research belongs. The trustworthiness criteria of the conventional paradigm are often discussed in terms of "internal validity", "external validity", "reliability", and "objectivity". Internal validity "refers to the extent to which the findings accurately describe reality" (Hoepfl, 1997, p. 58), and external validity "refers to the ability to generalize findings across different settings." (Hoepfl, 1997, p. 59). Hoepfl writes with reference to Kirk and Miller (1986, p. 41-42), that three different types of reliability have been identified in conventional research:

> 1) the degree to which a measurement, given repeatedly, remains the same; 2) the stability of a measurement over time; and 3) the similarity of measurements within a given time period (Hoepfl, 1997, p. 59–60)

Lincoln and Guba write that the usual criterion for objectivity is "intersubjective agreement; if multiple observers can agree on a phenomenon their collective judgment can be said to be objective." (Lincoln and Guba, 1985, p. 292) Another approach to establish objectivity is "through methodology; to use methods that by their character render the study beyond contamination by human foibles." (p. 292–293)

The comparable criteria in the naturalistic paradigm are "credibility", "transferability", "dependability", and "confirmability" (Lincoln and Guba, 1985, p. 300).

There have been several phases in the development of qualitative research, and the discussions in the literature on how to certify trustworthiness have consequently developed over the years.

In the present discussion on evaluation of qualitative research I will use the trustworthiness criteria suggested by Lincoln and Guba (1985). There are other approaches suggested to ensure trustworthiness in qualitative research, for example applying ideas from the conventional paradigm criteria. A recent example of a discussion on trustworthiness as an alternative construct to validity, reliability, and generalisability in phenomenographic research is Collier et al. (2008).

In the following I will discuss each of the four criteria; credibility, transferability, dependability, and confirmability as they, according to Lincoln and Guba, can be used to evaluate trustworthiness of research within the naturalistic paradigm.

The first criterion, *credibility*, deals with carrying out an inquiry in a way that enhances the chances for the findings to be found credible or believable. This can involve credibility from the participants' perspective. Credibility has less to do with the size of the sample than the quality of the data, the analysis and the written report. Lincoln and Guba suggest techniques to address credibility. Examples of these techniques are prolonged engagement, persistent observations, peer debriefing and triangulation, where the latter can refer to triangulation of data, methods, multiple analysts, and theory.

The second criterion, *transferability*, refers to the degree to which the results of the research can be transferred to other settings, contexts, or populations. Lincoln and Guba discuss transferability in terms of where "the burden of the proof lies" (Lincoln and Guba, 1985, p. 298). Instead of making generalizations, the researcher should provide what Lincoln and Guba call a "thick" description of the research. This description can include description of preparation of the study and the underlying research questions and assumptions, data gathering including choice of data collection methods and participants, quotations from interviews, analysis methods and decisions taken during the process of analysis, and the inferences the researcher has come to, and more. The thick description is presented to the reader so that he or she can determine whether the findings are applicable to his or her situation. The investigator does not know the context of the receiver. Only the receiver of the research knows and can judge the transferability of the research. Lincoln and Guba write:

> The best advice to give to anyone seeking to make a transfer is to accumulate *empirical* evidence about contextual similarity; the responsibility of the original investigator ends in providing sufficient descriptive data to make such similarity judgments possible. (Lincoln and Guba, 1985, p. 298) (Italics in original)

The authors emphasise that the researcher should provide "the thick description necessary to enable someone interested in making a transfer to reach a conclusion about whether transfer can be contemplated as a possibility." (Lincoln and Guba, 1985, p. 316)

The third criterion discussed is *dependability*, which is the naturalistic correspondence to reliability. Dependability emphasises that the researcher needs to account for the changing context in which the research occurs. Lincoln and Guba write: "The naturalist sees reliability as part of a larger set of factors that are associated with observed changes." Aiming to demonstrate dependability "the naturalist seeks means for taking into account both factors of instability *and* factors of phenomenal or design induced change." (p. 299) Lincoln and Guba write that it is argued that if credibility is fulfilled, dependability is also fulfilled: "Since there can be no validity without reliability (and thus no credibility without dependability), a demonstration of the former is sufficient

to establish the latter." (p. 317) Techniques related to credibility thus ensures that dependability is fulfilled.

The fourth criterion is *confirmability*, which corresponds to objectivity in the conventional paradigm. To what degree can the results of the research be confirmed or corroborated by others? Lincoln and Guba write that there are three different perspectives on objectivity (p. 299). The perspective that is often preferred by naturalists is "Objectivity exists when an appropriate methodology is employed that maintains an adequate distance between observer and observed." (p. 300) The authors conclude that this definition

> removes the emphasis from the investigator (it is no longer his or her objectivity that is at stake) and places it where, as it seems to the naturalist, it ought more logically to be: on the data themselves. [...] Are they or are they not *confirmable?* (p. 300) (Italics in original)

Techniques to ensure confirmability suggested by Lincoln and Guba are for example triangulation and the keeping of a reflexive journal. (p. 319)

4. The present research

This section discusses how the research approaches presented in Section 3.2 and Section 3.3 are applied in the present thesis. Subsequently the section discusses how trustworthiness, as discussed in Section 3.4, has been ensured in the different investigations involved in the thesis.

4.1 Research approaches applied in the present research

The thesis has three themes as is discussed in Section 1.1. The first theme, student learning of concepts, is analysed by means of deductive content analysis (Paper III), by means of phenomenography (Paper I, Paper IV, and Paper V), and by means of variation theory (Paper I and Paper IV). Paper II is mainly a literature review of work related to "threshold concept" (Meyer and Land, 2005) and will not be discussed below.

The second theme, student learning of practise, is analysed by means of inductive content analysis (Paper VI and Paper VII).

The third theme, the relationship between students' conceptual and practical learning, is analysed by means of deductive content analysis (Paper VIII), and by means of phenomenography and variation theory (Paper IX).

4.1.1 Phenomenography and variation theory in the present research

Paper I includes a traditional phenomenographic analysis of novice students' understanding of two central concepts in object-oriented programming, object and class. The analysis is described, and quotes from the students illustrate the different understandings identified. Two sets of categories of description are presented, and features of the different understandings are identified. The paper further includes a discussion of how variation theory can be applied to the phenomenographic results, and implications for teaching are inferred from the latter discussion.

Paper IV includes a traditional phenomenographic analysis of students' understanding of the phenomenon "programming" including quotes that illustrates the different categories of description. The paper further shows how the phenomenographic results can be used to design learning activities that support students' learning, by use of variation theory and patterns of variation. To this end we introduce the theory of phenomenography and variation theory,

and give a thorough review of the steps taken to identify the dimensions of variation that are related to the phenomenographic outcome space discussed. Finally we suggest appropriate patterns of variation that can be used to help students discern some of the identified dimensions of variation, and how these patterns can be applied in teaching novice programming.

Paper V includes a traditional phenomenographic analysis of students' understanding of the phenomenon "learning to program" with a description of the analysis performed and with quotes from the students to illustrate the categories of description identified in the analysis. We relate the results from the analysis to the "process-object duality" theory from mathematics education. We show that the phenomenographic analysis reveals problems students experience in learning object-oriented programming, not indicated in the "process-object duality" theory.

Paper IX builds mostly on the results presented in Paper I, Paper IV, Paper V, and Paper VIII. Paper IX uses phenomenography and variation theory to build an analytical model of students' learning of practise and concepts. Dimensions of variation are in the center of the discussion, tying together students' conceptual and practical learning. The conceptual understandings used to illuminate the analysis are novice students' different understandings of the concepts object and class and related dimensions of variation, as presented in Paper I. The practise is analysed by identifying common novice programming activities at different level of proficiency. Subsequently it is argued that these activities also are related to the same dimensions of variation. In this way practise, expressed as activities at different level of proficiency, and qualitatively different conceptual understandings are related through dimensions of variation, and a model of the complex learning process of novice programming is developed.

4.1.2 Content analysis in the present research

Paper III is based on deductive content analysis of semi-structured interviews with students from the second investigation who discuss important and difficult concepts they have met during their education. The interview questions were constructed to capture Meyer and Land's definition of threshold concepts (Meyer and Land, 2005). For each concept discussed by the students we analysed whether the criteria that characterize threshold concepts were met. In this way we identified two threshold concepts in computer science, pointers and object-oriented programming.

In Paper VI inductive content analysis is used on parts of the interviews discussed above. The goal with the research was to identify strategies that students use successfully in their computing studies, and to categorize the strategies in ways that made them useful for future students and educators (Paper VI, p. 156). Some interview questions concerned what the students did when they were stuck in their learning process. We aimed at identifying all strategies to get unstuck mentioned by the students. In an iterative process we grouped the strategies in categories. We first created many small cate-

gories where only a few strategies that appeared more or less the same were grouped together. The smaller categories were subsequently grouped together into broader, more abstract categories that covered large numbers of related strategies. The categories, the many small as well as the broader and more abstract, are presented in the paper.

Deductive content analysis is used in paper VIII on the interview data from the second investigation. Meyer and Land (2005) use the metaphor *the liminal space* to capture important features of students' experiences of being in the midst of learning threshold concepts (Meyer and Land, 2005). We analysed the parts of the interviews where students discussed their learning of the threshold concepts identified in Paper III. The analysis aimed at investigating the liminal space criteria, as discussed by Meyer and Land, and discussing how these criteria appear in computer science students' learning process.

In Paper VII, inductive content analysis was used to categorise artifacts produced by senior students. The research examined how students' software designs can be compared, and in addition investigated senior students' ability to design. The students were asked to design a software system. The designs, made on papers, were the artifacts analysed by the research group. Starting with a sample of 20 of the total 149 designs, we made an initial categorisation of the designs, based on their semantics and guided by our experiences as computer scientists, researchers, and teachers. We subsequently categorised the remaining designs, each researcher categorising 70 designs. Agreement was reached in close discussions within the research group. The identified categories were subsequently discussed in relation to academic and demographic background data gathered from the participants, which supported the interpretation of the results of the categorisation. In this part of the analysis we used quantitative methods for parts of the interpretation of the categories and inferences drawn.

4.2 Trustworthiness of the present research

In this section I will discuss how trustworthiness, as presented in Section 3.4, have been ensured in the present thesis. The three investigations will be discussed separately. For each of the three criteria suggested by Lincoln and Guba (1985), credibility, transferability, and confirmability, I will discuss how I have used techniques to ensure trustworthiness in the investigations. Since the dependability criterion is fulfilled with the credibility criterion, dependability will not be discussed (Lincoln and Guba, 1985, p 317). The techniques I discuss relate to the techniques suggested by Lincoln and Guba, but with minor modifications in my applications of them.

4.2.1 Trustworthiness in the first investigation

Data in the first investigation were collected from a series of interviews with 14 first year students who had just finished their first programming course.

The interviews aimed to elicit students' different understandings of some central concept. Research approaches used on the data are phenomenography in Paper I, Paper IV, Paper V, and Paper IX, and variation theory in Paper I, Paper IV and Paper IX. Below I describe how I have established trustworthiness in the phenomenographic analyses which includes variation theory, by fulfilling the three criteria.

Data collection and the analysis of the data from the first investigation was done in close discussion with a colleague. Both of us have long experience of teaching programming, and can thus be said to have prolonged engagement in the field. The phenomenographic analysis was partly performed by me, and later scrutinised by my colleague, and partly done by both of us separately, and then joined in discussions where we came to agreement on the results. The analyses have further been discussed in seminars with researchers in the CER field, and conference and journal papers have been peer reviewed. Peer debriefing has thus been used and the credibility criterion ensured.

The second criterion, transferability, has been ensured in the presentations through a thick description in the following ways. Great effort was made to ensure that the group of students investigated, the choice of participants for the interviews, the course they studied, the questions asked, the research approach taken, and the analyses performed were described in as much detail as possible considering page limitations and other practical limitations. In this way a thick description was provided for the reader.

Keeping of a reflexive journal is one technique suggested by Lincoln and Guba to ensure the confirmability criterion (p. 319). In the first investigation reflexivity, as discussed by Finlay (2002), has been used to some extent to ensure confirmability. The researcher always influences both collection and interpretation of data. Reflexivity means explicitly, with self-awareness, analysing one's own role in the research process (Finlay, 2002). Finlay writes: "Reflexive analysis in research encompasses continual evaluation of subjective responses, intersubjective dynamics, and the research process itself." (p. 532) This has been carried out in close dialog with my co-authors, by presenting the research at conferences, and specifically in educational situations where people from outside the phenomenographic community have been introduced to this research approach partly through my own research. This has encouraged me to analyse my own role in the research process. In this way the third criterion, confirmability, has been ensured.

4.2.2 Trustworthiness in the second investigation

The second investigation was multinational. This implies that the pool of data is possibly richer than if the data came from one institution only. The background of the participants is more diverse, and the education differs between countries and institutions. To ensure stringent interview conditions so that the data can be treated as one pool of information, certain steps were taken concerning choice of participants, and preparation of the interviews. To ensure that the participants were at comparable level in their educations, and stud-

ied at similar study programs, the issues of the background of participants and their study programs were thoroughly discussed within the group of researchers. To ensure trustworthiness in the performance of the study, the interview questions were worked out in close discussion among the researchers. One of the researchers subsequently performed a pilot interview with the rest of the researchers present, listening, but not interfering. The pilot interview was followed by a discussion among the researchers to resolve possible questions. The pilot interview was not added to the pool of information used in the subsequent analyses. The purpose of the pilot interview was solely to ensure trustworthiness in the performance of the study.

Research approaches used on the data are deductive content analysis in Paper III and Paper VIII, and inductive content analysis in Paper VI. Below I describe how my use of content analysis ensured trustworthiness in ways that fulfill the criteria discussed.

All researchers involved in the second investigation have long experience of teaching computer science. In addition, the work has been triangulated concerning data gathering methods in the following way: we have performed informal interviews with educators, an instructor survey, interviews with students, and a literature survey. Prolonged engagement in the field and triangulation ensures the fulfillment of the first criterion, credibility. Triangulation furthermore ensures the third criterion, confirmability.

In our reports we provided thick descriptions of interviewees, interview and survey questions, excerpts from the data, and how the analyses were performed. This ensures transferability.

4.2.3 Trustworthiness in the third investigation

The third investigation was performed by a group of 21 researchers from four countries. The investigation was led and designed by three of the researchers. Data was gathered by all researchers. A multinational study implies rich data in the sense described above. To ensure stringent interview conditions so that the data can be treated as one pool of information, certain steps were taken concerning choice of participants, and preparation of the "design brief", that is the task given to the participants which describe the problem and provide instructions (see Paper VII, p. 198). All researchers were given the same, detailed information on what was regarded as appropriate level of education of expected participants. The performance of the study was described in detail, and before the study was performed all researchers tried the "design brief" themselves, followed by a discussion among the researchers. In this way the researchers were given an opportunity to discover ambiguities in the setup of the study, and resolve differing understandings among themselves. The "design brief" was thereafter reviewed by the leaders of the investigation in line with the responses from the researchers before given to the participants.

The research approach used in Paper VII is deductive content analysis, which, as described below, has been used in a way that ensures trustworthiness.

As in the second investigation, prolonged engagement in the field is fulfilled since all researchers involved in the data gathering as well as those involved in the analysis of the data have long experience of teaching computer science. The subset of data used in Paper VII, designs from senior students, is triangulated consider that the participants came from 21 institutions in four countries. In this way the first criterion, credibility, as well as the third criterion, confirmability, have been fulfilled.

The report of the research provides the reader with a thick description which ensures transferability. The study is clearly described, the "design brief" given to the students is included in the paper, and the paper gives a rich description of the categorization procedure performed.

5. Results

This section presents results from the nine papers included in the thesis, organised around its three themes. The first theme, *student learning of concepts*, form a major part of the thesis. There is a large body of previous research on the second theme, *students learning of practise*, and the present research on this theme thus focuses on one specific skill, software design, which is less well researched than other central skills like reading and writing code. In the third theme, *the relationship between conceptual and practical learning*, I synthesise the results from the first two themes using the conceptual framework Ways of Thinking and Practising.

Results concerning possible implications for teaching are brought together in Section 6.

5.1 Learning of concepts

The first theme deals with problems concerning student learning of concepts. In particular three specific aspects of student learning of concepts are researched, reflecting the research questions in Section 1.1.

The first research question presented in Section 1.1 concerns novice students' understanding of programming concepts. The phenomenographic analysis presented in Paper I shows that the novice students' understanding of the concepts object and class vary from a narrow textual representation of the concepts, to a broader understanding of the concepts including the active behaviour of the objects when the program is executed, to the most desirable understanding that includes the two previous, but also the modeling aspects of the concepts. Very few students seem to have reached this understanding although it is fundamental in object-oriented programming. In particular our results show that:

- There are particular ways to understand the concepts *object* and *class* that are critical for students to discern. Few students seem to have reached the full understanding of the concepts described in the phenomenographic outcome spaces. The results from Paper I can be used by educators in the way that they can accentuate the identified features of the concepts in their teaching, and thus facilitate for students' learning and further studies.

The investigation presented in Paper I of students' understandings of the concepts object and class is in line with results presented in Paper II and Paper III, where conceptual learning is discussed at a higher level of granularity. While the first paper focuses on two specific concepts in the object-oriented

paradigm, namely object and class, Paper II and Paper III aim at discussing and identifying important concepts, so called "threshold concepts" (Meyer and Land, 2005) in computer science in general. Threshold concepts are described by Meyer and Land as a subset of core concepts in a discipline that might be used to organize and focus education.

Paper II discusses threshold concepts in relation to a number of other computer science education research areas. Paper II specifically discusses how the idea of threshold concepts relates to, and differs from, constructivism, mental models, student misconceptions, breadth-first approaches to introductory computer science, and fundamental ideas. We found support in the literature for the conclusion that abstraction and object-orientation fulfill the criteria of being threshold concepts.

Paper III continues the discussion started in Paper II. The paper presents an empirical investigation that aimed at identifying possible threshold concepts in computer science. We found empirical evidence of two threshold concepts: object-oriented programming, which was suggested from the literature survey in Paper II, and pointers.

We have not yet pinpointed which aspect(s) of object-oriented programming is(are) threshold concept(s). Object and class, which are the concepts investigated in Paper I, seem however to be reasonable candidates to study since they are not only central, but often introduced early in programming education. In particular our results show that:

- There exist threshold concepts in computer programming. The identification of such concepts gives valuable information for educators. Object and class, which are threshold concepts candidates, can act as nodes around which introductory programming education can be organised.

Paper IV and Paper V investigate student understanding of what programming and what learning to program means. This is the second research question.

The action dimension described in the outcome spaces in Paper I reappear in the outcome space in Paper IV. There seems to be similarities between how the phenomena "object" and "class" are understood relative to the phenomenon "computer programming". Or to phrase it with the variation theory terminology: there are dimensions in the programming learning space that seem to be common to several phenomena. If a specific feature of one phenomenon is discerned, for example a feature related to the action dimension, this can facilitate for the learning of other phenomena which are related to the same dimension. This is further developed in Paper IX, where variation theory is used to research students' learning process.

Another interesting connection between Paper I, Paper IV, and Paper V is the similarities in their respective outcome spaces. The understandings described go from a narrow, programs-as-text and programming-as-coding foci, to broader understandings that include the reality outside the computer and the course. The importance of students reaching the more advanced ways of understanding these phenomena are pointed to for example in Computing Curricula 2001 Section 7.2 (Roberts and Engel, 2001).

Students' understandings of the subject studied, here computer programming (Paper IV), *and* their understanding of what it means to learn that subject (Paper V), have been shown to be important for how students approach their studies, and thus have impact on the learning outcome (Booth, 1992, p. 261–262). Paper IV consequently discusses how variation theory, and specifically the patterns of variation discussed by Marton and Tsui (2004), can be applied to a phenomenographic outcome space. The discussion gives examples on how a phenomenographic analysis can be applied in teaching to help students advance their understanding of what computer programming means, which is the third research question.

In particular our results show that:

- There is a wide variety in students' understanding of what programming and learning to program means. Related work has emphasised the importance of students coming to a good understanding of what programming, and learning to program means. The phenomenographic outcome spaces and variation theory can be used by educators to facilitate for students' learning of these matters.

Paper V points to similarities with Hazzan's (2003) work on the process-object duality learning theory. Paper V however also points to differences in terms of problems novice students encounter, which are not observed in the process-object duality theory. The practise, discussed as processes by Hazzan, might be the major obstacle for some programming students in their learning. If the practise is experienced as too difficult to master, it can not serve as the a means for students to reach the learning goals. In this way results from Paper V point to the next theme of the thesis, students' learning of the practise.

5.2 Learning of practise

The second theme concerns the role of practise in computer science education. Practise is important for students' learning to program, both as a means to reach the learning goals, and as a learning goal in itself. Paper VI focuses on the former aspect in terms of students' learning strategies, which is the first research question in the second theme of the thesis, see Section 1.1. Paper VII discusses the latter aspect of practise, namely one specific learning goal, software design. This reflects the second research question in the second theme of the thesis. Software design is one of several important skills programming students are supposed to learn, as discussed in Paper IX.

Students' approaches to their learning have shown to be important for the learning outcome (Marton et al., 1984). According to for example Trigwell et al. (1994), approaches to learning can be discussed in terms of intention and strategy, where intention is what the student attempts to do, while strategy is what the student does to fulfill the intention. Paper VI identifies students' learning strategies. We discuss strategies in terms of what the students said they did when they were stuck in their learning.

The list of identified strategies is long, altogether 35 strategies. We grouped the strategies into four super-categories. The super-categories found are In-

puts/interactions where the students talk about getting help from elsewhere, Concrete/do stuff which is in line with what I call learning through practising, Abstract/understand stuff which captures when the students discuss "learning and getting unstuck at a higher level" (p. 158), as opposed to the former category, and the last category, "Use the Force", which involves strategies where students use "their willpower or character" (p. 158).

Programming strategies have been claimed to be important parts of programming skills (Davies, 1993). Robins et al. (2003) found in their literature review that lack of programming strategies caused problems for novice students. Our analysis is focused on learning strategies. Davies' and Robins et al.'s discussions are still relevant for our research since the strategies discussed in our study often concerns problems with programming concepts.

Although all students in the study but one described that they sometimes were stuck in their learning, they all had several strategies for getting unstuck, and the strategies were surprisingly diverse.

In particular our results from the research on practise as a means to reach learning goals, show that:

- It is important that students learn a variety of strategies for mastering the problems they meet in their learning. We specifically noticed the "importance of social interaction, and the active responsibility taken by the students". (Paper VI, p. 160)

Paper VII discusses one particular programming practise, software design. The focus of the paper is the problem of assessing designs: how can students' software designs be analysed and compared? The focus of the investigation and the data collection, and my focus in this theme, is however on how students learn to design. This is also present in the paper, but given as a background for the discussion on how to analyse rich artifacts like written and drawn designs.

The paper shows how the technique of semantic categorization can be used to organize such rich artifacts. 149 designs produced by near-graduating students were categorized into six groups of similar designs, depending on their semantics, and "ordered relative to the degree to which the stated requirements were met" (p. 199).

The overall question of the study was: Can students near graduation design software systems? Only 9% of the designs were assessed as reasonable designs. They fell into the two highest categories, labeled Partial design and Complete, of which only 2% were Complete. 29% of the designs fell into the category First step and showed some progress toward a design. The remaining 62% had no or very little information added beyond the specification given.

We saw a significant increase in number of syntactic features in the higher categories compared to the lower. Also the length of the designs increased in the higher categories, except for the highest, which slightly decreased, and the more advanced designs more often than the others included "an overview, details on part responsibilities, and communication between the parts" (p. 199). Other observations of interest are that there was a positive correlation between number of computer science courses taken and the category of the design: the more courses taken the higher category; academic performance measured by

grades on computer science courses seemed though to have little or no relationship to the design produced.

We found that semantic categorization is possible, but time-consuming and some designs required extensive discussions between the researchers to be categorized.

In particular our results from the research on practise as a learning goal show that:

- Computer science students near graduation have great difficulties in mastering design tasks, even though design is one of the core skills the students are supposed to learn during their education.

The present research points to the problematic role of practise in programming students' learning. Practise is not merely the means to reach the conceptual learning goals. The role of practise in the learning process needs to be further researched.

5.3 Ways of Thinking and Practising

The research presented in Paper VIII, From *Limen* to *Lumen*: Computing students in liminal spaces, investigates the learning process related to threshold concepts in computer science, which is the first research question in the third theme in the thesis. Paper III identifies two such concepts, pointers and object-oriented programming. In Paper VIII we take the analyses from Paper III one step further. We use the theory of *liminal space* as it is discussed by Meyer and Land (2005). The liminal space is described as "the transitional period between beginning to learn a concept and fully mastering it" (Paper VIII, p. 124). We applied the theory to our data as a framework to highlight certain features of the learning experience which, looking at the data as a whole, are difficult to discern. We looked for the standard features of the liminal space as described by Meyer and Land, but in addition we found some that may be specific to computer science. The result of the analysis reveals a broad and rich picture of the students' learning experiences.

The picture thus unfolded shows a transformative process which often takes long time, involves strong emotions and elements of mimicry, and contains specific parts, or sorts of understandings, which can become stuck places for students. The parts identified include an abstract, or theoretical, understanding of the concept; a concrete understanding – the ability to implement the concept (without necessarily having the abstract understanding); the ability to go back and forth between the abstract and the concrete understandings; an understanding of the rationale for learning and using the concept; and an understanding of how to apply the concept to new problems. Students need to attain all these understandings. This can explain why they obviously get stuck at different places, and "why the path through this space is not a simple linear progression." (p. 130) Students rather seem to "need to go back and forth between the theoretical and the practical" (p. 130), and different students take different "routes" depending on individual stuck places. We further point

to a result characteristic for computer science: "we commonly observed the particular partial understanding of not being able to translate from an abstract understanding to concrete implementation or design" (p. 130)

Beside this we discuss students' expressions of how, and if, they know that they know a concept. The students express the experience of mastering a concepts sometimes as emotional, sometimes as being able to visualize their understanding, and sometimes as being able to master the handicraft of programming. We further found evidence in the data that there were students who said they knew a concept that they apparently did not fully know, and students who doubted their own knowledge even though it seemed as if they knew.

The learning is experienced as a complex whole by the students, and thus difficult to fully discern. We found that the liminal space, as an analytic tool, provided a way to theoretically separate several important features of the process, and thus untangle some of the complexity of the learning.

In particular our results give empirical evidence that:

- The practise as well as the concepts are problematic for the students to learn. Both are important in the learning process and can become stuck places for the students. If the students face a problem with one of them, it is expected to have negative influence on the other.

In this way Paper VIII gives an empirical background for the analysis presented in Paper IX.

Paper IX, Ways of Thinking and Practising in Introductory Programming, is the synthesis of my thesis work. The paper builds on results from the first and the second investigations. Students' conceptual and practical learning are investigated, specifically how practise and concepts relate in student learning, which is the second research question in the third theme of the thesis. I argue from the empirical data that concepts and practise are equally important parts of the learning goals, and equally difficult for students to learn. Furthermore, there is a mutual dependency and complex relationship between the two. This discussion points to the need to research this relation. In particular:

- The research identifies dimensions of variation related to qualitatively different conceptual understandings. The research further identifies dimensions of variation related to practise in terms of programming activities at different levels of proficiency.
- Based on results from the analyses of the data and elements from phenomenography and variation theory, an analytical model is proposed. The model shows that activities as well as conceptual understandings relate to dimensions of variation.

Previous research has discussed dimensions of variation related to concepts as well as to practise (Marton and Tsui, 2004; Fazey and Marton, 2002). The present research takes this one step further. In particular:

- The most significant finding is that practise, in terms of programming activities, and conceptual understandings have dimensions of variation *in common*. This was possible to show since the research proposes a way to identify dimensions of variation related to practises.

- The dimensions of variation are thus like interfaces between conceptual understandings and activities. If a dimension of variation is discerned, this can open a possibility for students to discern concepts *and* to learn activities in new ways.

This finding can to some extent explain the complex learning of computer programming, where some students seem to first learn the concepts and then the practise, while other students seem to learn in the opposite order. The model can further to some extent explain why programming activities not always facilitate for students' learning. If the activity is at a level of proficiency that presupposes dimensions of variation not yet discerned by the student, the student might have problems to learn through the activity. This can be phrased using terminology from variation theory: if the patterns of variation involved in the learning situation are too complex, students might not discern the dimensions of variation involved in the situation.

The result shows that the dimensions of variation can relate to several concepts and activities. This carries implications for learning, in particular:

- If, for example, one way to understand a concept is discerned through a dimension of variation, this learning experience can facilitate for discernment of other related ways to understand concepts, and for the learning of related activities.

The results also indicate that activities as well as concepts can be related to more than one dimension of variation. It is fundamental in variation theory that concepts can relate to more than one dimension of variation (Marton and Tsui, 2004). The result that activities can relate to more than one dimension of variation indicates in particular that:

- 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.

6. Discussion - from a teaching perspective

As educators we know that many students, specially the novices, have great difficulty learning to program. This section will discuss how results from analyses inspired by phenomenography and variation theory can be implemented in programming teaching to facilitate for students' learning.

I will first discuss and develop the phenomenographic analysis presented in Paper I, see Section 6.1. This discussion is inspired by the research presented in Paper IV. In Section 6.2 I further discuss implications for teaching emanating from the analysis on how students' conceptual and practical learning relate, which is presented in Paper IX.

6.1 Phenomenography in practise - an empirical example

To exemplify how a phenomenographic outcome space can be used by educators, I will show an outcome space of novice students' understandings of the concepts object and class, and discuss a process that starts with a phenomenographic outcome space, identifies critical features of the phenomena (in this example two object-oriented concepts), discusses corresponding dimensions of variation, and arrives at implications for teaching in terms of concrete advice for educators. For a comprehensive description of the data and the analysis that gave the outcome space, see Paper I and Eckerdal (2006).

The results presented in Paper I indicate that many students have a problem fully grasping the investigated concepts object and class. The phenomenographic outcome spaces give however valuable information to educators on the different ways in which our students can understand these concepts. In addition the outcome spaces provide information necessary for retrieving what is *educationally critical* for a good understanding of the concepts. Educationally critical means that there are certain ways to understand a concept that are critical in the sense that if the student has not discerned these ways of seeing the concept, something important is missing, something that might be critical for the students future studies, or critical for the development of the student's capabilities in the subject studied, here computer programming. Educationally critical features of a concept can be identified by use of other techniques, see for example Runesson (2006), but in the present thesis phenomenography has been used.

6.1.1 The phenomenographic outcome space

The concepts object and class are closely related, and can hardly be understood without each other. When describing the different understandings found in the data, it is not surprising to find similar patterns for the understandings of the two concepts. The initial two outcome spaces presented in Paper I have thus been collapsed in one outcome space in Table 6.1 below.

Class is understood 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 understood 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 understood as a description of properties and behaviour of objects, where object is understood as a model of some real world phenomenon.

Table 6.1: *Summary of categories describing the different ways to understand the concepts* object *and* class *found in a group of novice students. The latter categories include the understandings in the former.*

The analysis indicated inclusive categories, as expressed in Table 6.1. This means that an understanding expressed in one of the latter categories includes the understandings expressed in the former, and thus expresses a richer understanding of the concepts. 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 behaviors, expressed in the code.

What can we as educators do to facilitate for the students to develop their conceptual understanding? The following three sections, inspired by the research presented in Paper IV, discuss how the empirical results presented in Table 6.1 can be further analysed and give implications for teaching.

6.1.2 Discernment and variation - identification of critical features

As discussed in Section 3.2, different categories in an outcome space represent combinations of features of the phenomenon, which are present in the focal awareness at a particular point in time (Marton and Booth, 1997, p. 126).

Learning is understood as developing richer ways to see the phenomenon, as represented in the more advanced categories of the phenomenographic outcome space. A necessary, but not always 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. (Marton et al., 2004, p. 31).

The first category in Table 6.1 reflects the students' understanding of 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 is the appearance of the structure of the program text. The focus of the understanding of objects, is on the program text. The feature, critical in this category is thus the textual representation of the concepts.

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

The last category includes, in addition to what is described above, that classes are understood as descriptions of properties and behaviour of objects, where objects are understood as models 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 new feature expressed in this category is the modeling aspects of the concepts.

The students' foci, and consequently the critical features of the concepts, are hence identified. Variation in a dimension corresponding to a feature is, as discussed above, a prerequisite for learning to take place. Having expressed the identified critical features of the concepts, as captured by the categories of description in Table 6.1, it is now possible to discuss what dimensions of variation correspond to each feature.

6.1.3 Dimensions of variation - open a space for learning

When there is a variation in a dimension that corresponds to a critical feature, this opens a possibility for students to discern the feature and thus learn the concept in a new way. In the first category in Table 6.1 the critical feature is the textual representation of the concepts. To be able to discern this feature, students need to discern that in different programs objects and classes appear in different ways. In that sense, the textual representation of programs constitutes a relevant dimension related to this feature. Different, specific program

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

texts constitute values along this dimension and if students discern such variation, it opens the possibility of understanding object and class in this way.

The new feature expressed in the second category that the students need to discern is the active behavior of the program during execution. Different actions resulting from different program executions constitute values in the corresponding dimension of variation.

In the last category in Table 6.1, the new feature added is the modeling aspect of the concepts. In this case, different real-life phenomena modeled as classes and corresponding objects, constitute values along this dimension.

The line of reasoning above is summarized in Table 6.2. It includes the students' different understanding of the concepts object and class, as expressed in Table 6.1, see the left column in Table 6.2. The right column includes the corresponding dimensions of variation.

Students' understandings of the concepts object and class	Corresponding dimensions of variation
Class is understood as an entity of the program, contributing to the structure of the code and describing the object.	The textual representation of the concepts.
As above, and in addition, class is understood as a description of properties and behaviour of objects, where object is understood as something that is active in the program.	As above, and in addition, the action of the program.
As above, and in addition, class is understood as a description of properties and behaviour of the object, where object is understood as a model of some real world phenomenon.	As above, and in addition, the modeling aspects of the concepts.

Table 6.2: *Categories describing the different understandings of the concepts* object *and* class*, and the corresponding dimensions of variation related to the critical features of the identified understandings.*

6.1.4 Implications for education - patterns of variation

Table 6.2 carries implications for teaching. Teaching is here defined in a wide sense, not restricted to lecturing, but may include for example programming assignments given to students, software tools introduced to students, lectures, Internet and fellow students, anything the students meet and choose to use in their learning. The whole organisation of the learning environment is in this sense teaching.

The educator can create learning conditions that enable students to discern new features of the concepts. In this context it means creating possibilities for experiencing variation in dimensions related to features. We know from the analysis of our data that *any* variation is not sufficient. By varying some things and keeping others invariant, we can create the conditions necessary for learning. As educators we know this can be done in several different ways, and yet it is a difficult task.

The claim that not any variation is sufficient for creating good learning conditions is important in computer programming and counter-intuitive to how we often teach. For example Kölling and Rosenberg (2001) write that novice students should read code, not only simple code but large programs including many classes which can help them understand what object-oriented programming is. The downside of this approach is that large programs often mean that a number of different features of several concepts are present and vary simultaneously. The present research, together with a number of classroom studies reported by Marton and Tsui (2004), indicate that if students are not introduced to the critical features in adequate ways, they may not discern these features, and simultaneous variation of several features may not always provide good learning conditions. This is evident in the following quotes from one of the students in the first investigation, when he or she discusses the contrast between learning mathematics and learning computer programming:

> Here [in the programming course] 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

All the examples have obviously not helped the student sufficiently since he or she says about the programming course:

> 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

Marton et al. (2004, p. 16–17) discuss so called *patterns of variation* which are identified from empirical studies. The patterns are ways of systematically combining variation and invariance in the teaching. Four different patterns are identified that can be used by educators as a toolbox. The patterns are *contrast, generalization, separation,* and *fusion* respectively. In short the patterns means (quoting Paper IV):

contrast to contrast a phenomenon *P* to other related phenomena, to make it possible to discern *P* as a phenomenon distinct from other phenomena.

generalization to exhibit varying specific appearances of *P*, in order to open the possibility to discern the general meaning of *P*.

separation there is variation in precisely one dimension, to create the possibility to discern that particular dimension, keeping the other dimensions invariant.

fusion to exhibit variation in several dimensions simultaneously, to open the possibility to discern the relations between these dimensions.

Patterns of variation: some examples

The content of Table 6.2 can be implemented in the teaching and learning environment by use of patterns of variation in a number of ways. There is great freedom and possibility to adapt the results to the need and desire of each educator, study group and learning resources. The following paragraphs discuss possible ways to achieve this, by showing a few examples.

For the first category, the students need to become aware that different programs represent classes and objects differently, at a textual level. This corresponds to the second part of the first category. In the first part of the first category, the focus is on the structure of the program text. There are 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 need to be exposed in teaching. A way to achieve this is to use the *generalization* pattern, in a variety of simple UML class diagrams[2] (Rumbaugh et al., 1999). By exhibiting various specific appearances of classes, the general meaning of class and object as text can be discerned. 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 and will show varying textual examples. This is often the case in the examples considered in the beginning of a programming course. The feature 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.

For the second category, the new feature focuses on actions during program execution. Different actions of the program taking place when the program is executed make a dimension of variation related to this feature. I will first discuss the *separation* pattern. The general idea of this pattern is that there is variation in precisely one dimension, so there is a possibility for the student to discern that particular dimension. This seems to be an appropriate pattern for our purpose. It is however difficult to achieve variation in one dimension only since a change in action requires a change in the program text. In Paper IV we suggest the notion of *pseudo separation*. In this context this means that the textual differences between two programs is kept small, but still causes a change in action when the program is executed. This will give the student the possibility of discerning the action dimension separately. Pseudo separation is a form of fusion pattern since in fact there is a variation in both the action dimension and the textual dimension, even though the latter is not prominent. When the student has discerned the action dimension, the proper fusion pattern can be used to show the *relation* between the program's textual representation and its actions. An example of a resource that can be used for the latter example is

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

BlueJ (Barnes and Kölling, 2003), where a debugger can be used to execute the program in steps so the variation of the code and variation in values of variables can be observed simultaneously during program execution.

For the last category in Table 6.2 the feature added focuses on objects and classes as models of the real world. To help students discern the dimension related to the modeling feature, the *generalization* pattern can be used. Modeling appears in many areas in the students' lives. Road signs model real world phenomenon like road bumps and let us avoid long, written instructions. Mathematical symbols model complex relations like sums and integrals and simplify the treatment of computations. Modeling in computer programming helps us to treat complex real-world problems. Once the student has discerned the modeling dimension, the *fusion* pattern, where variations in several dimensions are exhibited simultaneously, can be used to help the student discern the relationship between the modeling dimension and the action and textual dimensions. Results from the first investigation point to the importance of letting students follow the whole process of a programming task, including the analysis of a problem in real life, and not only focus on implementing code. 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 need help to modify their models to find 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 feature of objects and classes, and also to discern the relationship between the real world problem, the model in terms of class diagrams, and the implementation of the problem as code.

For further examples on how results from phenomenographic analyses can be implemented in teaching, I refer to Paper IV where a phenomenographic analysis of novice students' understanding of what *computer programming* means is described. Critical features and corresponding dimensions of variation are identified, followed by a discussion on how patterns of variation can be used to open a space of learning for the students. In Paper V on the other hand, we present a phenomenographic analysis of novice students' understanding of what it means to *learn* computer programming. It has shown to be important for students to have a good understanding of what learning the subject means. The outcome space presented in Paper V can be used by educators in similar ways as the present discussion to facilitate for novice students to get a good foundation for their learning.

6.1.5 The results related to previous research

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

For example, Computing Curricula 2001 Section 7.2 (Roberts and Engel, 2001) writes:

> 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 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. Using terminology from variation theory, if the focus of introductory programming is on coding only, the textual dimension of variation is highlighted at the expense of the action and modeling dimensions.

As a second example I will discuss some misconceptions pointed to in the literature (Holland et al., 1997), namely an overemphasizing of the object's data feature at the expense of the behavioural feature and the "object as a kind of variable" misconception. The latter may occur if the examples 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.

Both misconceptions point to the importance of understanding the concepts as they are described in the second categories in Table 6.2. The second category emphasizes that classes describe the behaviour of objects. The second category also explains classes as a description of properties of objects, and most real-world objects have more than one property.

Holland et. al give some advice on how to help students avoid these misconceptions. To increase the chances of avoiding the "object as a kind of variable" misconception the authors suggest that all the classes showed as an introduction should have more than one instance variable and that these variables should be of different type. Another way to avoid over-emphasising the object's data feature, suggested by the authors, is using introductory object examples where the response to a message is substantially altered depending on the state of the object. Holland et al.'s suggestions are in line with variation theory and the discussion in Section 6.1.4. Using variation theory terminology, examples with at least two instance variables of different types is using the generalization pattern, and examples where the response to a message substantially alters depending on the object is using the (pseudo) separation pattern as discussed above.

A third example, also mentioned by Holland et al. and Sanders and Thomas (2007), is the common problem among novice programmers of understanding the difference between class and object. This might become a problem if several examples are presented in which only a single instance of each class is used. Holland et al. suggest that it would help to avoid this misconception

if several instances of each class are always presented. As explained in Section 6.1.2, 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., which is according to the first category in Table 6.2.

In the light of the present study, the recommendations from Holland et al. are explained by and theoretically rooted in variation theory. Variation theory and patterns of variation are thus scientifically based and empirically tested tools to be used by educators to develop their teaching.

As a fourth example, Holmboe (1999) performed a study where students who had just finished an introductory course on object-oriented programming, senior students, and educators, were asked to describe in their own words what object-oriented programming is. He made a qualitative analysis of the answers, and concludes that some types of knowledge are more suitable as a basis for further knowledge construction than others. He writes about the understanding that includes the world outside the computer itself: "A person with holistic knowledge relates the implementation and design of a computer program to the real world being simulated." Holmboe emphasizes the importance that "[...] more students will experience the connection between reality, model and implemented program, and thus reach holistic knowledge of object-orientation sooner in their learning process." The third category in Table 6.2 captures an understanding of classes and objects that includes the world outside the computer itself, the modeling of real-world phenomena, and Section 6.1.4 discusses how educators can facilitate for students to discern this understanding by use of patterns of variation.

One challenge for educators of object-oriented programming, 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 critical features and related dimensions of variation of the concepts the students need to discern in order to reach a rich understanding. Here, variation theory can be used, as demonstrated in the previous discussion. Finally, the patterns of variation are like a tool box for educators to open up dimensions of variation and thus give students opportunities to come to richer understandings of the concepts.

6.2 Dimensions of variation and student learning of practise

Paper VIII discusses students' learning of threshold concepts. The paper points to the important but problematic role of practise in programming students' learning, and how concepts and practise interact in the learning process.

Paper IX develops this line of research further. The paper gives examples of typical novice student programming activities related to the skills of reading, writing, and debugging code, see Table 6.3.

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 emphasise the structured 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 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 6.3: *Common novice programming skills with associated activities.*

The paper discusses how the activities correspond to different levels of proficiency, and furthermore, how the activities relate to previously identified dimensions of variation. These dimensions were identified from the phenomenographic outcome space on novice students' understandings of the concepts object and class as discussed in Section 6.1.3. The identified dimensions of variation are thus related to different conceptual understandings as well as to activities at different levels of proficiency. In this way, the dimensions of variation act as interfaces between qualitatively different conceptual understandings and activities at different levels of proficiency.

There are implications for teaching following from these results. First, novices are often expected to perform many of the activities mentioned in Table 6.3 at an early stage of their education. Some of them are however related to dimensions of variation corresponding to a high level of proficiency. These dimensions are at the same time related to advanced ways of understanding concepts that we know very few of the students have discerned yet. This means, using variation theory terminology, that the students have not yet discerned the dimensions of variation related to the

activities, and we still expect them to manage them. This can to some extent explain why novice students have such big problems learning, and why the activities in the lab do not always lead to the expected learning outcome.

Another result from Paper IX is that to be able to discern a certain feature of a concept, *or* to make an activity meaningful, certain dimensions of variation in the learning space need to be open for the student. Or, to phrase the same thing differently: the learning of concepts *and* activities presupposes that related dimensions of variation are discerned. At the same time, the richer ways to see the concepts, and the activities at the higher level of proficiency, relate to more dimensions of variation and require thus that more dimensions *and* their relations be discerned.

Educators can use the results from Paper IX together with patterns of variation to facilitate students' learning, the conceptual as well as the practical. When dimensions of variation are identified, appropriate patterns of variation can be introduced to the students to facilitate the learning of corresponding concepts *and* practises.

7. Conclusions and future work

Computer programming is a core area in computer science education that involves practical and well as conceptual learning goals. It is however widely reported in the computer science education research literature that novice students have great problems in learning to program. The problems reported apply to both concepts and practise.

The research presented in this thesis contributes to the body of knowledge on students' learning by investigating the relationship between conceptual and practical learning in novice students' learning to program. Previous research in computer science education has focused either on students' learning practise or on concepts. The present research however indicates that students' problems with learning to program partly depend on a complex relationship and mutual dependence between the two.

The most common way to reach practical as well as conceptual learning goals in programming education is to "learn through practising". Students are expected to "learn to do the practise" as well as to learn the concepts through practising. If the students do not master the practise this might hinder further learning, conceptual as well as practical. The present research indicates that the students find practise at least as difficult to learn as concepts. The practise in not merely the unproblematic means of reaching the learning goals.

The research builds on three empirical investigations. The data from the investigations have been analysed from several perspectives. Students' conceptual and practical learning are first investigated separately by means of content analysis, and phenomenography and variation theory.

The analyses of student learning of concepts show that many students have problems to learn central concepts. The analyses show however how phenomenographic results can be used to facilitate for students' learning by use of variation theory. The analysis of students' ability to master the practise shows that students hold a great variety of strategies that they can use when they are stuck in their learning. On the other hand senior computer science students perform poorly when asked to perform a design tasks. Design is a core skill in computer science education. There are obviously problems in students' achievements of the practical learning goal.

In a subsequent analysis inspired by phenomenography and variation theory I show that practise, in terms of programming activities at different levels of proficiency, as well as conceptual understandings at qualitatively different levels, are related to dimensions of variation.

Previous phenomenographic research points to how critical features of concepts are related to dimensions of variation. Previous research also suggests

that practise can be related to dimensions of variation. The most significant finding in the present thesis is that I have demonstrated that practise, in terms of activities at different level of proficiency, and qualitatively different conceptual understandings, have dimensions of variation *in common*. This has been possible since I propose a way to identify dimensions of variation related to practises.

An analytical model is suggested where the dimensions of variation are like interfaces, relating concepts and activities. The implications of the model are several. If the dimensions of variation are at the center of the learning process this implies that when students discern a dimension of variation, related conceptual understandings *and* the meaning embedded in related practises can be discerned.

The model further suggests that activities as well as concepts can relate to more than one dimension. This implies that activities at a higher level of proficiency, as well as qualitatively richer understandings of concepts, relate to more dimensions of variation. The analysis on novice students' conceptual understandings points to dimensions of variation that many of the novice students not seem to have discerned. The analysis of students' activities shows that some of the activities students' often are expected to do and learn early in their education, relate to these dimensions of variation that the former study showed were problematic to discern. This can to some extent explain why the exercises in the computer lab do not always lead to improved learning. The results can furthermore be used by educators to help students' discern dimensions of variation and thus facilitate for the learning, practical as well as conceptual. A concrete example is given on how variation theory and patterns of variation can be applied in programming education.

The results need further investigations. Phenomenography and variation theory (Marton and Booth, 1997; Marton and Tsui, 2004) traditionally discuss ways to identify critical features of phenomena like 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 the student learning by proposing a way to identify dimensions of variation related to practise. Furthermore, the research proposes a model which demonstrates how dimensions of variation are like interfaces between concepts and practise, and between several concepts and several practises. There is a need of further empirical studies on how practise relates to dimensions of variation, and on the relationship between conceptual and practical learning. The analytical model can thus be used. This line of research might be possible by performing Learning Studies (Lo et al., 2004) which focus on educationally critical features of concepts and related practises.

Summary in Swedish

Nybörjarstudenters lärande av begrepp och praktik i programmering

Programmering är ett kärnämne inom datavetenskapliga utbildningar på universitetsnivå. Undervisning i programmering har lärandemål som gäller praktik lika väl som begrepp. Forskning i datavetenskapens didaktik visar emellertid att nybörjarstudenter har stora svårigheter att lära sig programmering. De rapporterade svårigheterna gäller såväl praktik som begreppsförståelse.

Forskningen i den här avhandlingen bidrar till befintlig forskning genom att undersöka relationen mellan begreppsligt och praktiskt lärande med fokus på nybörjarstudenters lärande av objekt-orienterad programmering. Tidigare forskning inom datavetenskapens didaktik har antingen fokuserat på studenters lärande av praktik, eller på lärandet av begrepp. Trots många försök att utveckla undervisningen kvarstår problemen med nybörjarstudenters lärande. Avhandlingen visar emellertid att studenters problem att lära sig programmering delvis beror på ett komplext samspel mellan och ett ömsesidigt beroende av praktik och begrepp i lärandeprocessen.

Det vanligaste sättet att nå de praktiska såväl som de begreppsliga lärandemålen i programmeringsutbildningar är att "lära genom att göra praktik", det vill säga genom att skriva datorprogram. Studenternas lärande av de praktiska såväl som de begreppsliga lärandemålen beror till stor del på om de klarar av att "göra praktik". Avhandlingen pekar på att studenterna erfar praktiken åtminstone lika svår att lära som koncepten. Praktiken är inte bara ett oproblematiskt medel att nå de konceptuella lärandemålen.

Forskningen bygger på tre empiriska studier. Den första studien undersökte nybörjarstudenters förståelse av några centrala begrepp inom objekt-orienterad programmering. 14 civilingenjörsstudenter inom området miljö- och vattenteknik intervjuades. Data från den första studien har främst analyserats med en fenomenografisk och variationsteoretisk forskningsansats.

Den andra studien fokuserade på att identifiera centrala och för studenterna problematiska begrepp, så kallade tröskelbegrepp. 16 sistaårsstudenter med datavetenskaplig inriktning intervjuades. Data från den andra studien har analyserats med en innehållsanalytisk forskningsansats.

Syftet med den tredje studien var att undersöka om studenter i slutet av sin datavetenskapliga utbildning kan designa datorprogram. Data från undersökningarna, designer gjorda av studenterna under kontrollerade former, analyserades med en innehållsanalytisk forskningsansats.

Studenternas lärande av begrepp och praktik analyserades först var för sig. Därefter undersöktes relationen mellan begreppsligt och praktiskt lärande.

Den fenomenografiska analysen av nybörjarstudenternas begreppsförståelse visar kvalitativt skilda sätt på vilka studenterna förstår, eller uppfattar, några centrala begrepp i objekt-orienterad programmering. Resultatet tyder på att många studenter har problem att lära sig de mer avancerade sätten att förstå begreppen, som också är de önskvärda från ett utbildningsperspektiv. Den variationsteoretiska analysen visar emellertid att variationsmönster (eng. *patterns of variation*) kan användas av lärare på resultat från den fenomenografiska analysen för att stödja studenter i deras lärande.

Analysen av hur studenter klarar de praktiska lärandemålen visar att studenterna besitter en stor variation av strategier som de kan använda när de får problem i sina studier. Studien visar också att studenter i slutet av sin datavetenskapliga utbildning presterade sämre än förväntat på designuppgiften. Design är ett kärnområde i datavetenskaplig utbildning som studenterna förväntas lära sig. Resultaten av analysen tyder på att det finns problem med studenters förvärvande av de praktiska lärandemålen.

Analysen av relationen mellan begreppsligt och praktiskt lärande är inspirerad av fenomenografi och variationsteori. Den visar att såväl praktiken, i termer av programmeringsaktiviteter på olika färdighetsnivåer, som kvalitativt skilda förståelser av begrepp, är relaterade till variationsdimensioner (eng. *dimensions of variation*).

Tidigare fenomenografisk forskning pekar på hur kritiska aspekter av begrepp är relaterade till variationsdimensioner. Tidigare forskning föreslår också att praktik kan relateras till variationsdimensioner. Det mest signifikanta resultatet i avhandlingen är att det visas att praktiken, i termer av programmeringsaktiviteter på olika färdighetsnivåer, och kvalitativt skilda begreppsliga förståelser, har *gemensamma* variationsdimensioner. Avhandlingen beskriver ett sätt att relatera variationsdimensioner till programmeringsaktiviteter, vilket har gjort det möjligt att komma fram till resultatet att begrepp och aktiviteter kan ha gemensamma variationsdimensioner.

En analytisk modell föreslås där variationsdimensioner fungerar som gränssnitt mellan begrepp och aktiviteter. Modellen har flera implikationer. Om variationsdimensioner är i centrum av lärandeprocessen, innebär det att när studenter urskiljer en variationsdimension, kan relaterade begreppsliga förståelser *och* meningen i relaterade aktiviteter urskiljas.

Modellen visar dessutom att såväl aktiviteter som begrepp kan relatera till mer än en variationsdimension. En tolkning av det resultatet är att aktiviteter på en högre färdighetsnivå, likaväl som kvalitativt rikare begreppsliga förståelser, relaterar till fler variationsdimensioner. Analysen av studenternas begreppsliga förståelser pekar på att många inte har uppfattat alla variationsdimensioner. Analysen visar ytterligare att vissa aktiviteter som studenterna förväntas kunna på ett tidigt stadium av sin utbildning, relaterar till just de variationsdimensioner som den tidigare nämnda studien pekar

på som svåra att uppfatta. Det nämnda resultatet kan till viss utsträckning förklara varför de praktiska övningarna i programmeringsundervisningen inte alltid leder till ökat lärande, varken av begrepp eller praktik.

Det är viktigt för läraren att kunna identifiera variationsdimensioner så att dessa kan lyftas fram i undervisningen. Avhandlingen ger konkreta exempel på hur variationsteori och variationsmönster kan användas i programmeringsundervisning. Utgående från ett fenomenografiskt utfallsrum visas hur kritiska aspekter av de olika förståelserna beskrivna i utfallsrummet kan identifieras. Varje kritisk aspekt relaterar till en variationsdimension. Därefter diskuteras hur olika variationsmönster kan användas för att lyfta fram variationsdimensioner i undervisningen, vilket kan hjälpa studenter att urskilja de identifierade variationsdimensionerna. Läraren kan på så sätt ge möjlighet till studenterna att lära sig relaterade begrepp *och* praktik på nya sätt.

Acknowledgments

At the end of my PhD studies I look back on a joyful but also demanding journey. Many people have helped and encouraged me during the journey. There are friends and relatives, colleagues and students, who have been supporters and contributed to the thesis, and I would like to thank them all. I will mention some of them below.

First of all I would like to thank Michael Thuné who has been my supervisor during the whole thesis work, and Anders Berglund who has been my supervisor after my licentiate thesis, for your great support during the process of creating this thesis. Your knowledge in the two areas my research spans, your advice, patience, and encouragement to me to try my own ideas, and our exciting discussions and collaboration on papers have been invaluable in my thesis work. I would also like to thank Shirley Booth who introduced me to the phenomenographic research approach during my licentiate work.

I would also like to thank my co-authors of papers in the thesis, the Sweden Group, for the enjoyable and rewarding research we have performed together (in alphabetic order): Jonas Boustedt, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, Kate Sanders, Lynda Thomas, and Carol Zander. I would especially like to thank Robert McCartney and Kate Sanders for their English proof reading the thesis, and Jonas Boustedt and Carol Zander for enjoyable travel to conferences and research meetings, and for long and rewarding discussions.

I also want to thank my research group at the Department of Information Technology at Uppsala University for valuable seminars with discussions, feedback, and encouragement, and all colleagues at the Department of Scientific Computing for a warm and stimulating work environment. Specifically I want to thank Liselott Dominicus for being a friend and traveling companion on the journey to the PhD.

Finally I would like to thank my family, Per, Nils, and Olof, who have made everything worthwhile. In this I also include my mother Anne-Mari Sundin who has encouraged me to start by saying:

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

and has continued to support my work throughout.

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

Bibliography

ACM Curriculum Committee on Computer Science (1968). Curriculum '68: Recommendations for the undergraduate program in computer science. *Communications of the ACM*, 11(3):151–197.

Austing, R. H., Barnes, B. H., and Engel, G. L. (1977). A survey of the literature in computer science education since curriculum '68. *Communications of the ACM*, 20(1):13–21.

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

Bayman, P. and Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of basic programming statements. *Communications of the ACM*, 26(9):677–679.

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 *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 257–261, New York, NY, USA. ACM.

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.

Berglund, A., Daniels, M., and Pears, A. (2006). Qualitative research projects in computing education research: an overview. In *ACE '06: Proceedings of the 8th Austalian conference on Computing education*, pages 25–33. Australian Computer Society.

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.

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.

Clancy, M., Stasko, J., Guzdial, M., Fincher, S., and Dale, N. (2001). Models and Areas for CS Education Research. *Computer Science Education*, 11(4):323–341.

Collier Reed, B., Ingerman, Å., and Berglund, A. (2008). Reflections on trustworthiness in phenomenographic research:ărecognising purpose, context and change in the process of research. *Education as Change*, (in press).

Daly, C. and Waldron, J. (2004). Assessing the assessment of programming ability. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 210–213.

Davies, S. P. (1993). Models and theories of programming strategy. *International journal of Man-Machine Studies*, 39(2):237–267.

Denzin, N. K. and Lincoln, Y. S. (1994). Introduction. Entering the Field of Qualitative Research. In Denzin, N. K. and Lincoln, Y. S., editors, *Handbook of Qualitative Research*, pages 1–17. SAGE Publications.

Denzin, N. K. and Lincoln, Y. S. (2005). Introduction: The discipline and practice of qualitative research. In Denzin, N. K. and Lincoln, Y. S., editors, *The SAGE Handbook of Qualitative Research third edition*, pages 1–32. SAGE Publications.

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

Eckerdal, A. (2009). Ways of Thinking and Practising in Introductory Programming. Technical Report 2009-002, Department of Information Technology, Uppsala University, 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., Ratcliffe, M., Sanders, K., and Zander, C. (2006a). Putting threshold concepts into context in computer science education. *SIGCSE Bulletin*, 38(3):103–107.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., and Zander, C. (2006b). Can graduating students design software systems? In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 403–407.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliff, M., and Zander, C. (2006c). 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.

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. In *ITiCSE-WGR '98: Working Group reports of the 3rd annual SIGCSE/SIGCUE ITiCSE conference on Integrating technology into computer science education*, pages 41–56.

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.

Fincher, S. and Petre, M. (2004). Mapping the territory. In Fincher, S. and Petre, M., editors, *Computer Science Education Research*, pages 1–8. Routledge.

Finlay, L. (2002). "Outing" the Researcher: The Provenance, Process, and Practice of Reflexivity. *Qualitative Health Research*, 12(4):531–545.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., and Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18(2):93–116.

Fleury, A. E. (1999). Student conceptions of object-oriented programming in Java. *The Journal of Computing in Small Colleges*, 15(1):69–78.

Fleury, A. E. (2000). Programming in Java: Student-Constructed Rules. In *Proceedings of the 31st SIGCSE technical symposium on Computer science education*, pages 197–201, Austin, Texas, United States.

Fleury, A. E. (2001). Encapsulation and Reuse as Viewed by Java Students. *ACM SIGCSE Bulletin*, 33(1):189–193.

Fung, P., Brayshaw, M., and du Boulay, B. (1990). Towards a taxonomy of novices' misconceptions about the prolog interpreter. *Instructional Science*, 19(4-5):311–336.

Goldman, K. J. (2004). A concepts-first introduction to computer science. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 432–436. ACM.

Goldweber, M., Clark, M., and Fincher, S. (2004). The relationship between CS education research and the SIGCSE community. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 147–148. ACM.

Graneheim, U. and Lundman, B. (2004). Qualitative content analysis in nursing research: concepts, procedures and measures to achieve trustworthiness. *Nurse Education Today*, 24(2):105–112.

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.

Gugerty, L. and Olson, G. (1986). Debugging by skilled and novice programmers. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 171–174, New York, NY, USA. ACM.

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

Hoepfl, M. C. (1997). Choosing Qualitative Research: A Primer for Technology Education Researchers. *Journal of Technology Education*, 9(1):47–63.

Holland, S., Griffiths, R., and Woodman, M. (1997). Avoiding object misconceptions. In *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, pages 131–134.

Holmboe, C. A. (1999). A 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*, pages 17–20.

Hsieh, H.-F. and Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research*, 15(9):1277–1288.

Kahney, H. (1983). What do novice programmers know about recursion. In *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 235–239, New York, NY, USA. ACM.

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*, pages 7–11.

Kirk, J. and Miller, M. L. (1986). *Reliability and validity in qualitative research*. Sage Publications.

Kölling, M. (1999a). The problem of teaching object-oriented programming, part 1: Languages. *JOURNAL OF OBJECT-ORIENTED PROGRAMMING*, 11(8):8–15.

Kölling, M. (1999b). The problem of teaching object-oriented programming, part 2: Environmentss. *JOURNAL OF OBJECT-ORIENTED PROGRAMMING*, 11(9):6–12.

Kölling, M. and Rosenberg, J. (2001). Guidelines for teaching object orientation with java. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 33–36, New York, NY, USA. ACM.

Krippendorff, K. (2004). *Content Analysis: An Introduction to Its Methodology*. Thousand Oaks, Calif.: Sage.

Lincoln, Y. S. and Guba, E. G. (1985). *Naturalistic Inquiry*. SAGE Publications.

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.

Lo, M. L., Marton, F., Pang, M. F., and Pong, W. Y. (2004). Toward a pedagogy of learning. In Marton, F. and Tsui, A., editors, *Classroom Discourse and the Space of Learning*, pages 189–225. Lawrence Erlbaum Associates, Mahwah, NJ.

Lopez, M., Whalley, J., and Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the forth International Computing Education Research Workshop*, pages 101–111.

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., Runesson, U., and Tsui, A. (2004). The space of learning. In Marton, F. and Tsui, A., editors, *Classroom Discourse and the Space of Learning*, pages 3–40. Lawrence Erlbaum Ass., Mahwah, NJ.

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

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.

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, 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(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.

Mostyn, B. (1985). The Content Analysis of Qualitative Research Data: A Dynamic Approach. In Brenner, M., Brown, J., and Canter, D., editors, *THE RESEARCH INTERVIEW Uses and Approaches*, pages 115–145. ACADEMIC PRESS INC. (LONDON) LTD.

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*, pages 95–103.

Pears, A., Seidman, S., Eney, C., Kinnunen, P., and Malmi, L. (2005). Constructing a core literature for computing education research. *ACM SIGCSE Bulletin*, 37(4):152–161.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *ITiCSE-WGR '07: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 204–223, New York, NY, USA. ACM.

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.

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 37th SIGCSE technical symposium on Computer science education*, pages 560–561.

Powers, K., Ecott, S., and Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 213–217, New York, NY, USA. ACM.

Ragonis, N. and Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts. *Computer Science Education*, 15(3):203–221.

Randolph, J. (2007). *Computer science education research at the crossroads: A methodological review of the computer science education research: 2000-2005*. PhD dissertation: Utah State University http://www.archive.org/details/randolph_dissertation Retrieved November 19, 2008.

Roberts, E. and Engel, G. (2001). Computing Curricula 2001: Final Report of the Joint ACM/IEEE-CS Task Force on Computer Science Education. IEEE Computer Society Press, December 2001, http://www.acm.org/sigcse/cc2001/.

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.

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.

Sanders, K., Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., , Thomas, L., and Zander, C. (2008). Student understanding of object-oriented programming as expressed in concept maps. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 332–336.

Sanders, K., Fincher, S., Bouvier, D., Lewandowski, G., Morrison, B., Murphy, L., Petre, M., Richards, B., Tenenberg, J., Thomas, L., Anderson, R., Anderson, R., Fitzgerald, S., Gutschow, A., Haller, S., Lister, R., McCauley, R., McTaggart, J., Prasad, C., and Scott, T. (2005). A multi-institutional, multinational study of programming concepts using card sort data. *Expert Systems*, 22(3):121–128.

Sanders, K. and Thomas, L. (2007). Checklists for grading object-oriented cs1 programs: concepts and misconceptions. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 166–170, New York, NY, USA. ACM.

Simon (2007). A Classification of Recent Australasian Computing Edcuation Publications. *Computer Science Education*, 17(3):155–169.

Spohrer, J. C. and Soloway, E. (1986). Alternatives to construct-based programming misconceptions. *SIGCHI Bulletin*, 17(4):183–191.

Stamouli, I. and Huggard, M. (2006). Object Oriented Programming and Program Correctness: The Students' Perspective. In *ICER '06: Proceedings of the second International Workshop on Computing Education Research*, pages 109–118, Canterbury, United Kingdom.

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

Tenenberg, J., Fincher, S., Blaha, K., Bouvier, D., Chen, T., Chinn, D., Cooper, S., Eckerdal, A., Johnson, H., McCartney, R., Monge, A., Moström, J., Petre, M., Powers, K., Ratcliffe, M., Robins, A., Sanders, D., Shwartzman, L., Simon, B., Stoker, C., Tew, A., and VanDeGrift, T. (2005). Students designing software: a multinational, multi-institutional study. *Informatics in Education*, 4(1):143–162.

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

Trigwell, K., Prosser, M., and Taylor, P. (1994). Qualitative differences in approaches to teaching first year university science. *Higher Education*, 27(1):75–84.

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

VanDeGrift, T. (2004). Coupling pair programming and writing: Learning about students' perceptions and processes. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 2–6.

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Ajith Kumar, P. K., and Prasad, C. (2006). An austalasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies. In *ACE '06: PProceedings of the 8th Australian conference on Computing education*, pages 243–252.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *SIGCSE Bulletin*, 28(3):17–22.

Zou, L. and Godfrey, M. W. (2008). Understanding interaction differences between newcomer and expert programmers. In *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, pages 26–29, New York, NY, USA. ACM.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 600

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-9551

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2009