UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 734

On the Road to a Software Profession

Students' Experiences of Concepts and Thresholds

JONAS BOUSTEDT

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2010

ISSN 1651-6214
ISBN 978-91-554-7789-9
urn:nbn:se:uu:diva-122304

Dissertation presented at Uppsala University to be publicly examined in room 2446, Polacksbacken, Villavägen 14, Uppsala, Friday, June 4, 2010 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract

Boustedt, J. 2010. On the Road to a Software Profession. Students' Experiences of Concepts and Thresholds. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 734. 61 pp. Uppsala. ISBN 978-91-554-7789-9.

Research has shown that there are gaps in knowledge between newly hired and experienced professionals and that some of these gaps are related to concepts, such as the concepts of object orientation. This problem, and the fact that most computer science majors want to work in the software industry, leads to questions regarding why these gaps exist and how students can be better prepared for their future careers. Against this background, this thesis addresses two theme-based perspectives that focus on students' views of concepts in Computer Science.

The first theme-based perspective investigated the existence of potential Threshold Concepts in Computer Science. Such concepts should be troublesome, transformative, irreversible, and integrative. Qualitative methods have been mainly used and empirical data have been collected through semi-structured interviews, concept maps, and written stories. The results identified two Threshold Concepts, suggested several more, and then described the ways in which these concepts have transformed students.

The second theme-based perspective took a phenomenographic approach to find the variation in how students understand concepts related to the software profession. Data were collected via semi-structured interviews. In one study the interviews were held in connection with role-playing where students took on the role of a newly hired programmer. The results show a variety of ways to experience the addressed phenomena in the student collective, ranging from superficial views that often have a practical nature to more sophisticated understandings that reflect a holistic approach, including a professional point of view.

Educators can use the results to emphasize concepts that are important from students' perspectives. The phenomenographic outcome spaces can help teachers to reflect upon their own ways of seeing contrasted with student conceptions. I have indicated how variation theory can be applied to open more sophisticated ways of seeing, which in this context stresses the professional aspects to help students prepare for becoming professional software developers.

Keywords: Biographies, class diagram, computer science education, computer science education research, computing concepts, concept map, content analysis, higher education, java interface, learning, object orientation, phenomenography, programming, role-play, software development, software profession, threshold concepts, variation theory

Jonas Boustedt, Division of Scientific Computing, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

© Jonas Boustedt 2010

ISSN 1651-6214
ISBN 978-91-554-7789-9
urn:nbn:se:uu:diva-122304 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-122304)

To the “Sweden Group” and the CSER community

List of Papers

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

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

II K. Sanders, J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, L. Thomas, and C. Zander. (2008) Student Understanding of Object-Oriented Programming as Expressed in Concept Maps. In *SIGCSE Bulletin*, 40(1), pp. 332–336.

III J. E. Moström, J. Boustedt, A. Eckerdal, R. McCartney, K. Sanders, L. Thomas, and C. Zander. (2008) Concrete Examples of Abstraction as manifested in Students' Transformative Experiences. In *Proceeding of the fourth international workshop on Computing education research*, pp. 125–136.

IV C. Zander, J. Boustedt, R. McCartney, J. E. Moström, K. Sanders, and L. Thomas. (2009) Student Transformations: Are They Computer Scientists Yet? In *Proceeding of the fifth international workshop on Computing education research*, pp. 129–140.

V J. Boustedt. (2008) A Methodology for Exploring Students' Experiences and Interaction with Large-scale Software through Role-play and Phenomenography. In *Proceeding of the fourth international workshop on Computing education research*, pp. 27–38.

VI J. Boustedt. (2009) Students' understanding of the concept of interface in a situated context. *Computer Science Education*, 19(1), pp. 15–36.

VII J. Boustedt. (2010) A Student Perspective on Software Development and Maintenance. Technical Report 2010-012, Department of Information Technology, Uppsala University, Sweden.

VIII J. Boustedt. (2010) Ways to Understand Class Diagrams. Technical Report 2010-013, Department of Information Technology, Uppsala University, Sweden.

Reprints were made with permission from the publishers.

Author's Contributions

This section comments on the author's contributions to the papers upon which this thesis is based.

I When I joined the research group, known as the "Sweden Group", in March 2006, some data were already collected from instructors using questionnaires and informal interviews. I was engaged in designing a new study on students using semi-structured research interviews. All seven authors worked together on design, data analysis, and writing the paper. My particular contribution was sharing experiences from interviewing and analysing interviews.

II The study was proposed and designed mainly by K. Sanders. All seven authors worked together with data collection, graph coding, analysing and writing the paper. My particular contribution was to make and run software tools to analyse some aspects of the graph data.

III Originally, the idea to use biographies came from C. Zander who, together with L. Thomas, collected most of the data using assignments in English writing classes. All seven authors worked together with data analysis and writing the paper. I mainly focused on Design Patterns in analysing the data and writing the paper.

IV This paper and Paper III were based on the same data set. All six authors worked together with data analysis and writing the paper. I mainly focused on analysing and writing about which types of concepts could be associated to students' transformational experiences.

V This methodology paper describes the research approach and methods used in my first study on students' experiences and understandings of concepts and the practical work related to a situated context of work with a larger software system. I am the sole author of this paper.

VI In this publication, I report the findings from the study described in Paper V. I am the sole author of this paper.

VII This is a study where students' understandings of software development and software maintenance were explored. I designed the study, collected and analysed the data, and wrote the paper.

VIII Students' ways to understand class diagrams and related concepts were investigated. The study used the same interview data and analysis methods as used in Paper VII. The paper was produced in sole authorship.

Contents

1. Introduction

Computer Science Education Research is still a novelty in comparison to educational research in more established academic subjects. Over the past thirty years, research in this young discipline has been primarily engaged with problems related to students' difficulties in learning programming. I have devoted myself to trying to understand more about how students perceive and understand *concepts* in Computer Science and aspects of the *software profession*. This thesis presents my findings and it is my hope that they contribute to the field of Computer Science Education Research.

1.1 Research agenda

In my research on student learning in Computer Science, I have focused on two intertwined themes.

Student perspectives on concepts in Computer Science

The first theme involves finding out why it is so difficult to learn computer science and aspects of object-oriented programming in particular. I have chosen to approach this problem area by identifying various concepts in the subject area that students perceive to be particularly distinctive, for example, by being difficult or enriching. It is also an objective to investigate how students in all stages of university education relate to these concepts and how they connect some of the concepts to each other, see theme 'A' in Figure 1.1.

Inspired by a theory by Meyer and Land [44] which conjectures the existence of *Threshold Concepts* in academic disciplines, a group of researchers including myself, took on the task to investigate whether such concepts exist in Computer Science, and given that they do exist, what they are and how they affect students' learning. In a series of studies, the researchers in this international group have explored this matter to find empirical evidence.

Preparing to be a software professional

The second theme concerns how well students are preparing, or are prepared by what they have learned, for a future career and the demands and expectations that will be placed on them in a professional context.

Earlier studies have pointed out knowledge gaps when comparing novice employees to what experienced software professionals need to know [32], and

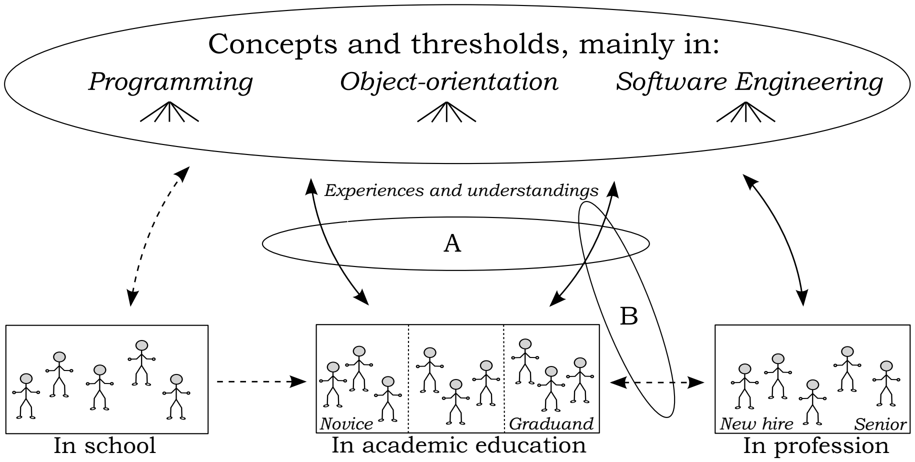

Figure 1.1: An illustration of the two research themes. Theme A is focused on trying to identify concepts that are problematic and transforming seen from the university students' perspective. Theme B is focused on how education prepares university students to become software professionals. This is addressed by investigating how last-year students experience concepts that are related to professional perspectives. Note that my research is not aimed at learning in upper secondary school although some pupils study programming.

that the novice employees' expectations often seem to mismatch the reality of the profession [3]. The question is: *"Why?"*

The indications of substantial knowledge gaps concerning object-oriented concepts, and that there may be knowledge gaps in other aspects, motivate the study of how students majoring in computing are prepared for the future and how they understand various concepts, processes and situations that relate to the expectations that are imposed by the professional context.

I have investigated these matters by studying graduating students' understanding of specific concepts that are important both for understanding the discipline and for understanding the professional perspective. Moreover, I have studied students' understanding of broader phenomena and processes with strong ties to a professional context, see theme 'B' in Figure 1.1.

How the themes are connected

Concepts can be understood in different ways by students – certain concepts may never have been heard of while other concepts are well-known. Most ways to understand are more or less correct, but some ways are more complete than others seen in an educational and professional perspective. However, the students that manage to get a degree in Computer Science or Computer Engineering bring along their understanding of concepts and expectations of the profession into their first employment. Any gaps in necessary knowledge and skills have to be filled when needed, learned by experience or by studies at work. A better understanding of which concepts students experience as troublesome and worthwhile, and how they understand these concepts, will help educators in training better computer scientists as well as better preparing students for becoming software professionals. As indicated by research [32], software design and object-orientation are areas with a potential for improving formal learning of concepts to better prepare students for the expectations of future employers.

One delicate problem for educators and education researchers is the balance between the scientific subject-related and the professional point of view, and whether the two perspectives are in conflict. In my opinion, it is common that these two perspectives enrich, motivate and reinforce each other when it comes to the understanding of concepts and practice in computer science.

1.2 The main research questions

The underlying key issue in Computer Science Education Research is: *"How can we improve learning and instruction in Computer Science?"* In my research, I have chosen to search for concepts in Computer Science that students experienced as particularly difficult and important and that have changed the students' understanding of Computer Science. I have also examined the variability in how students understand concepts that represent essential knowledge for those who want to work as software professionals. An overarching theme is whether and in what ways we can improve education, to better prepare students to become software professionals.

The overall research questions in this thesis are:

- *Why are programming and Computer Science so difficult to learn – are there concepts that both hinder and enable students to understand the big picture?* The question is made operational by finding which concepts in Computer Science are the most meaningful to students in the sense of being both hard-earned and rewarding, and in which ways they affect students learning.
- *What can we do to enhance the possibilities for students to become well-prepared software professionals?* The question is made operational by searching for the different ways students understand various concepts that software professionals need to master.

1.3 Outline

This thesis is a “comprehensive summary” of my research. For readers with a background in Computer Science, this is a well-known format, whereas readers with a background in Education Research are used to the monograph format. Unlike monographs in which research is presented as a single coherent text, a comprehensive summary is structured by writing a text that presents and summarizes research publications included in the thesis. Typically, this text is relatively short and those who want to know more about the details of research can find them in the included publications.

In this chapter, I described the background to my research interests and the overall research questions, and in Chapter 2, I will describe previous research related to my questions. Chapter 3 describes the theories and approaches I have used in different ways in my research. Chapter 4 describes the various methods used to collect empirical data, and Chapter 5 describes the methods used to analyse the data. The main results are reported in Chapter 6, and finally the conclusions of my research are summarized in Chapter 7. The publications included in the thesis are placed at the end of this comprehensive summary.

2. Related work

In recent years, literature providing general overviews of Computer Science Education Research has emerged. Clancy et al. [11], Holmboe [24], Pears and Daniels [49] as well as Fincher and Petre [18] describe good examples, models, areas and research agendas for such research. Berglund, Daniels and Pears [5] describe examples of qualitative research projects in the area.

Randolph [52] investigated Computer Science Education Research on meta-level which resulted in a methodological review based on journal publications and conference proceedings. He concludes that up to now, many hypotheses have been generated; however, it is now time to find more evidence and establish stable results.

2.1 Learning the concepts necessary to program

The question of what it means to learn programming and why it is difficult for beginners has been a main theme in research on computer science education for nearly three decades. Several studies indicate that first year students have poor programming skills. DuBoulay [13] stressed the need for understanding the mechanisms of an executing program and in this respect pointed out the comprehension of the "notional machine" as one of the keys for learning programming. DuBoulay as well as Bonar and Soloway [6] reported that students were confused by the discrepancy between programming languages and natural languages. More recent results reveal that difficulties in learning to program are still a major issue. McCracken et al. [42] asked if students can write code, Lister et al. [34] investigated whether students can read and debug code, and Eckerdal et al. [16] inquired students' abilities to design code. In all cases the answer was – *No*! Software design is an activity that requires a more comprehensive view than just using a programmer focus, because it deals with real world phenomena and requirements [43].

Robins et al. [53] have compiled a literature overview on studies of learning programming. They were mainly interested in how beginners learn to program, and they took a cognitive perspective. They noticed that the research has focused on understandings and development of programs, mental models, and knowledge and skills that are required to be able to program. They claimed that it takes 10 years for a novice to become an expert, and they classified five developmental levels: novice, advanced beginner, competent, skilled, and expert. Five overlapping domains are involved in learning, namely: orientation, the concept of machine, notation, structures and practical skills.

Pears et al. [50] have selected and compiled relevant literature on introductory programming. Their paper gives good advice for instructors faced with issues regarding curricula design, pedagogical approaches, choice of programming language, and teaching tools. They concluded that there is no evidence to support any particular pedagogical approach. Instead, their study focused on describing various proven approaches. This helps educators to make well-informed decisions.

Booth pioneered in taking a phenomenographic approach to tackle pedagogical issues within Computer Science Education. In her classic work in the area [7], she investigated how students learn and approach programming through a study where the students' descriptions of their conception of several related phenomena and situations were analysed. The students learned a functional programming language, and it is interesting to reflect on similarities and differences compared to learning object-oriented programming, which is problematic for many students and thereby is focused in my research.

Bruce et al. [10], used phenomenography to investigate how students go about learning to program – what they called "the act of learning to program." The outcome of their research pointed out five distinctive ways in which the students acted: following instructions, learning to code, understanding and integrating concepts, learning what it takes to solve problems, and finally, learning what it takes to be a part of the programming community. Then they discussed the possible implications the results have for teaching.

Similarly, Eckerdal [14][15] used a phenomenographic approach to study how a group of novice students, involved in a Java programming course, in different ways experienced learning to program and the object-oriented concepts *class* and *object*. The results were later used by Thuné and Eckerdal [57] in a discussion on how Variation Theory (Section 3.4) can be applied in teaching to help students be aware of the more advanced ways to understand the concepts and approaches that had emerged from the earlier study.

Hadjerrouit [22] claims that passive learning, where knowledge is transmitted from the teacher to the learner, is not adequate for most students when it comes to the object-oriented paradigm because the concepts are far more abstract than in procedural programming. Instead, he advocates a constructivist approach to learning object-oriented design and programming and claims that learning must be an active process of construction. In a more recent publication [23], he brings this approach further and describes a general model for how constructivism can be applied to teaching within software engineering.

My conclusion is that learners' understanding of concepts is a central issue; they can be viewed in many different ways and there are different ways to learn them. One of my research aims is to understand the variation in ways of seeing concepts. This knowledge can be used to help students see concepts in advanced ways.

Threshold Concepts in computing

Inspired by the notion of Threshold Concepts (see Chapter 3.1), Flanagan and Smith [20] divide programming thresholds in two kinds: (1) localized Threshold Concepts, e.g., Java Interfaces, and (2) the programming language itself as an overwhelming threshold. Their focus is on "operationally challenged students" for whom the programming *language* is troublesome. These students are generally motivated, highly qualified and successful in other subjects. The authors lend terminology from linguistics and identify "markedness" of logically related contrasting attribute pairs in Java, e.g., true/false and public/private. Asymmetric pairs may cause problems for the students, especially if one of the words is only implied by the language, such as in the pair static and "non-static". Another problem they see is that students read code as if it was natural language and as they do not understand the code, they desperately need notes or templates to be able to mimic it. The authors point out debugging as another problem for students being overwhelmed by long error listings filled with unfamiliar words such as exception, symbol and thread. They conclude that the language needs to be addressed much more.

Shinners-Kennedy [56] points out that everyday notion conceal important Threshold Concepts within Computer Science. In the search for the concepts believed to truly transform computer science students, we risk to miss fundamental ones because they are simple and obvious. For instance, the notions "empty list" and "end of list" are the most troublesome parts for students when they solve list problems, not the algorithms. "State" is another example of a concealed every day life concept that can be very troublesome, because in everyday life: "... we are not conscious of the fact that our next action is determined by the current state or that our actions that change the state are the critical ones." However, in programming the understanding of "state" is central, e.g., how it affects running programs: "Novice programmers are notoriously bad at what computer scientists describe as program debugging, despite the fact that they have extensive experience of the activity in other contexts." (cf. [34]) To make students aware of this concept, Shinners-Kennedy advocates a state based way to introduce several other related concepts. [56]

However, the definition of Threshold Concepts is not unchallenged. Rowbottom [54] claims that the definition of Threshold Concepts is too vague to be practical to use for empiric investigations. His point is that it is not possible to establish Threshold Concepts by studying students' abilities – even if Threshold Concepts exist – and that the thresholds are individual.

In our research, we have taken up the challenge to use the notion of Threshold Concepts as a means for learning more about the relation between students and concepts in computing, and we were excited to investigate the possibility of finding such concepts in empirical data using the definition given by Meyer and Land [44]. We believe this is a suitable approach towards understanding more about concepts that play an important role for learning in computing in various ways.

2.2 Educating for a professional career in industry

One of the main issues in my research deals with the question of how well computing educations prepare the students for a software profession such as developers and programmers, and this question takes its starting point in the presumption that a considerable portion of the students that follow programmes in computer science or computer engineering strives for profession in the software industry. Being software professional requires a comprehensive view, including knowledge about computer systems, programming, databases, project methodology, test methods, and good customer relations. The object-oriented paradigm and its related development methods deals with most of the topics mentioned above.

In an attempt to establish the relevance of formal education for software professionals, Lethbridge [32] investigated the skills of active professional software developers, how much of it originated from formal academic training and how much were acquired after employment. He studied 168 professional software developers in 24 countries. Among topics that the participants valued highly, it turned out that knowledge in programming languages by large came from formal education. However, there were a number of other highly valued skills where the majority of learning occurred after formal education. Object-oriented concepts, software design and patterns, architecture, design of graphical interfaces, maintenance, refactoring, configuration, human computer interaction, people skills, project management, release management and requirements analysis are examples of such knowledge areas where the gap between the formal competence and the needed competence was substantial [32]. I take this as a challenge and I believe it is possible for education researchers to help reduce these gaps!

Begel and Simon [3] focus mainly on how *newcomers at work* can be better prepared *as new software developers*. They point out that newcomers in the software industry had “significant mismatches between their job expectations and reality” [3] which can lead to low self-efficacy and role ambiguity, but on the other hand: “Newcomers with accurate expectations have higher self-efficacy and lower role ambiguity.” [3][19] Hence, they suggest a number of improvements to computing education. They emphasize social aspects, such as confusion of roles and power hierarchies. Besides suggestions such as pair programming, legitimate peripheral participation and mentoring, they propose “new hire videos” that could tune in the students’ or new hires’ expectations to a more realistic picture of what happens at work:

> In these videos, new developers in industry could talk about their experiences, reflections on their university education, expectations, and daily tasks. While most software developers’ jobs would not be considered “sexy” they do involve quite a bit more communication and social activity than pure functional knowledge of programming, design and debugging. [3]

Tvedt et al. [59] suggest a Computer Science Curriculum that in their view is better adapted to the needs of the industry. They point out that contemporary educational systems produce students with good technical skills, but unfortunately, the students lack the required practical software engineering of the profession. Their solution to this problem is their own proposed educational model, Software Factory. The students will learn more and consolidate more of their knowledge by applying their new skills in an authentic development environment. It would accordingly be of advantage to the students, the teacher staff, the academic institution, and the industry. Their model has been adapted and implemented.

An unconventional variant is project-based educations, or at least courses that introduce realistic or even authentic projects where the students themselves choose which knowledge are required to fulfil the commissions. For an example, see Jaccheri [26].

In contrast, Parnas [48] claims that educational programmes within computer science should not focus on software engineering because software engineering and computer science are different. His point is that programmes that do focus on software engineering need to follow the structure of traditional engineering educations. I believe it is necessary to emphasize software engineering in all computing educations, but I agree that much is to be learned from other engineering traditions.

The research described above addresses the theme in which my second research question is formulated. Some of the authors point right at the issues related to the preparedness for – and the transition into a professional life; their results support and motivate my research and how it has been designed.

A dialogue between university and industry is needed

Jaccheri and Morasca [27] point out the importance of a dialogue between the industry and the educational institutions and identify possible connections between industry and education: (1) universities can arrange continuation training for employees, (2) former students facilitates direct communications channels between the companies and the universities, (3) companies are interested of sharing the results of empirical research within the education and the industry, (4) companies can act as more or less authentic customers in student projects, finally, (5) professional experts act as teachers when they share their experiences with students in guest lectures.

Vaughn [60] reports results and experiences from an educational model where students work in authentic projects towards authentic customers in an industrial environment. Their experiences are mainly good and both the students and the customers are happy with this way of working. The deliverables that was the basis for examination was a conceptual model of operations, a specification of system requirements, a design document, a test plan, system documentation, and a formal delivery to the customer.

These experiences of attempts to improve learning by connecting teaching to a relevant context are interesting examples of practical ways to make student more aware of professional aspects. Other researchers have investigated the possibility for students to learn computer science as apprentices.

Can education be based on apprenticeship?

Lave and Wenger [31] started from the idea to try preserving apprenticeship – the traditional and ancient way of learning. They tried to investigate and explain its relation to the concept situated learning. From this perspective, they created a sociological and cultural epistemological theory based upon the presumption that learning takes place in social forms.

In the beginning, the learners are allowed to take limited part of the culture, which is called *Legitimate Peripheral Participation* (LPP). Gradually, the commitment gets deeper and more complex.

Ben-Ari [4] examined situated learning in the context of Computer Science. He concludes that it in its proper sense is not applicable for the entire chain of learning that must precede the high-technological knowledge that Computer Science Education strives to reach. Generalization and models must be utilized in order to make the education effective. On the other hand, it is possible to make use of and be inspired by this learning style when the content of the education is designed and the literature is chosen. The teachers should be well aware of the different *Community of Practice* (CoP) that the education aims for and they should design courses that reflect authentic situations taken from these CoPs. Ben-Ari is therefore sceptical of the effectiveness of an entire education formed as an apprenticeship. Yet, for suitable courses, he appreciates the idea of creating authentic environments and situations, e.g., he has designed a course book based authentic documents.

Kölling and Barnes [29] suggest an integrated model for teaching that combines apprenticeship with problem based learning and case studies. They describe how to do this in a first programming course in Java. First, the students are acquainted with a software system, which is a game designed by experts. The students explore the software interactively by running it and studying the code while they describe the software to peer students. Then they discuss design of alternate versions of the game and improvements to the existing software. The discussion soon moves from details in the code into code quality and maintenance and the students develop the skill of being able to evaluate code critically. Then the students work with exercises that gradually extend the software in different ways. Finally, the students make their own versions of the game as an assignment. During this activity, tutors discuss the solutions with the students with focus on aspects such as maintenance and extendibility.

Similar ideas inspired me when designing the study described in Paper V and VI. In a role-play, the informants worked with large software and were later interviewed. Also, after the study, the software was reused in programming classes to help students get a more comprehensive view of what software can look like. Other researchers [3] suggest that students would benefit from experiences of working with legacy code. In my view, this has connections to learning in apprenticeship.

Software maintenance as an important but often overlooked activity

Kajko-Mattson et al. [28] have pursued research on education within software engineering and its relevance for the industry. In particular, they focus on software maintenance and conclude that system maintenance is a job for the beginners, whilst the experienced take care of system development. They claim that maintenance has low status and quote Gunderman:

> Maintenance has been viewed as a second class activity, with an admixture of on-the-job training for beginners and low-status assignments for the outcast and the fallen. [21]

On the contrary, they argue that software maintenance is important and requires high competence in the form of skills and formal training. To achieve the needed competence they suggest an education in large-scale software.

> A highly skilled maintainer is the most important organizational asset pivotal for achieving quality software, strategic for improving maintenance and development processes, essential for remaining competitive and critical for business survival. This requires that universities properly prepare students to enter the maintenance workforce and that maintenance organizations actively build and maintain their human knowledge and skill base. [28]

As also indicated by Begel and Simon [3], software maintenance seems to be a potential task for new hires and that this is something that they are not expecting nor preparing for during education.

Implications for my research

Previous research points out that novice students are poor at writing, reading, and debugging programs. They have insufficient understanding of object-oriented concepts and they have problems with designing programs. These students are struggling with the overall understanding of the syntax and semantics used in programming languages, and some are confused by what is similar and what is not similar to natural languages.

Newly hired professionals experience a mismatch between their expectations of work and their experience of the reality. Their tasks often deals with learning new tools, reading and writing documentations and specifications, and fixing bugs and implementing features to existing software.

More experienced professionals describe in retrospect that there were major knowledge gaps which they had to fill after being employed. Programming languages as such, were not experienced to be an issue, but they experienced

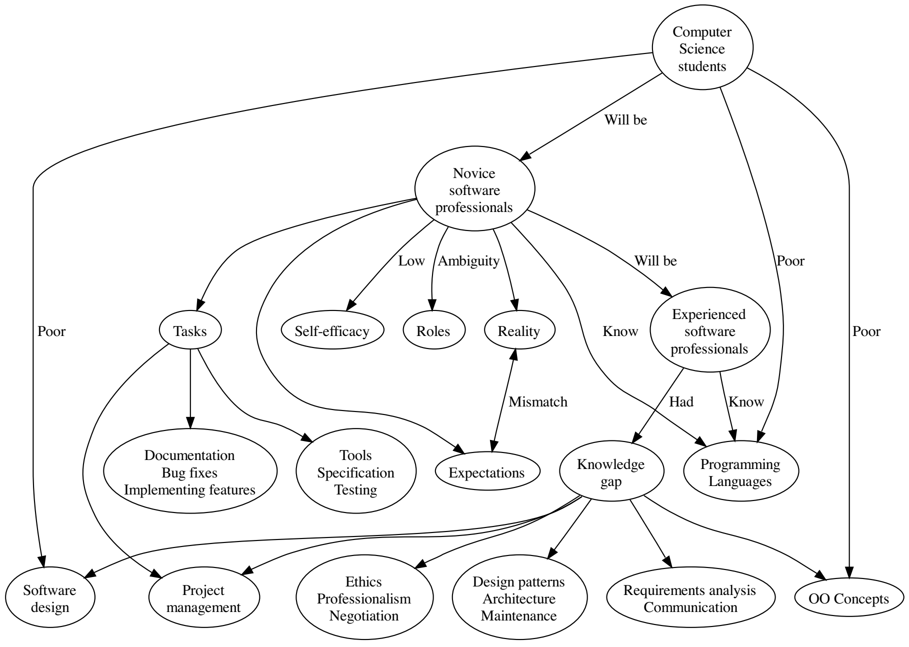
Computer Science students
Will be
Novice software professionals
Low
Ambiguity
Will be
Poor
Poor
Poor
Tasks
Self-efficacy
Roles
Reality
Know
Experienced software professionals
Mismatch
Had
Know
Documentation Bug fixes Implementing features
Tools Specification Testing
Expectations
Knowledge gap
Programming Languages
Software design
Project management
Ethics Professionalism Negotiation
Design patterns Architecture Maintenance
Requirements analysis Communication
OO Concepts

Figure 2.1: A summary of interesting findings regarding students learning of Computer Science and the relations to the software profession.

that they had learned much more about object-oriented concepts, software design, maintenance, et cetera, at work.

These results stress the importance of identifying and addressing concepts in computing that are problematic to students, especially the concepts that has also been indicated by the research on software professionals. My research intention is to contribute with more knowledge of first year and final year students' understanding of concepts that are important for the discipline, both in academic and professional sense.

My research has two themes for investigation: (1) students' experiences of concepts that they themselves in different ways see as important, memorable and sometimes troublesome, (2) the variation of the ways in which students in the second half of their education view concepts connected to professional conditions and known knowledge gaps [32] in this context. Object-oriented concepts are central in both themes.

This chapter presented many interesting findings from Computer Science Education Research. Figure 2.1 is an attempt to illustrate and summarize what I believe is most interesting and inspiring for my research.

3. Theory and approach

This chapter gives an overview of the major theories and research traditions that has inspired the research presented in this thesis.

Threshold Concept theory is a fairly recent theory about valuable concepts within disciplines which act as portals to new learning dimensions, and how knowledge about such Threshold Concepts can be used to improve instruction.

Concept mapping is a way to structure concepts and thoughts of how concepts are related. It can be used as a means for learning and also as a tool for understanding how people experience concepts and relations in some field.

Phenomenography is a way of seeing learning and it is also a qualitative research approach to learning about learning from the learners' perspective.

Variation Theory explains that variations in what the learner perceives is needed for learning. For example, the concept of colour cannot be discerned if everything in the viewer's world has always been blue.

Finally, models for obtaining trustworthiness in qualitative research are discussed.

3.1 Threshold concept theory

Meyer and Land [44] mean that some concepts are experienced as dark mysteries that seem quite impossible to understand for the uninitiated to a discipline. Compared with more straightforward concepts, these troublesome concepts often play a major role in learning the subject. Large efforts are often required to understand them, and for a while the learner may be stuck in a conceptual limbo before he or she finally finds the gate that leads out of darkness. Understanding of such concepts has transformed the learner forever into seeing the world with new eyes; like pieces in a jigsaw puzzle, they integrate other bits in the knowledge puzzle into a whole. This type of concept can often be found among those learning topics that constitute the frontiers of a discipline which distinguishes it from other sciences.

Fascinated by these "portals" that both prevent and enable learning, Meyer and Land coined the term *Threshold Concept*, and began to do research on this theme. In order to clarify more precisely what they meant by the new term, they proposed a definition based on its distinguishing features. Threshold Concepts are: (1) *transformative*, (2) probably *irreversible*, (3) *integrative*, (4) possibly *bounded* to a discipline area, and (5) potentially *troublesome* [44].

Furthermore, Meyer and Land claim that it is easier to identify Threshold Concepts in some subjects, such as Physics, and more difficult in others, such as History. They argue that teachers' knowledge of the Threshold Concepts that exist in a discipline plays an important role in the evaluation and design of effective learning environments, and their purpose is to create a debate about this. By virtue of their strong transforming properties Threshold Concepts are often difficult to learn. There is an obvious risk that those who do not really understand these concepts will be detained in a state where they are forced to imitate and remember by heart without getting familiar with the concept. They are then stuck into what Meyer and Land denote *liminal space* [45].

Land et al. [30] encourage educators in various subjects to reflect on their subjects by using threshold concepts as an analytical tool to better understand learning in their subject and thus to improve education.

> Threshold Concepts can be used to define potentially powerful transformative points in the student's learning experience. In this sense they may be viewed as the 'jewels in the curriculum' inasmuch as they can serve to identify crucial points in the framework of engagement that teachers may wish to construct in order to provide opportunities for students to gain important conceptual understandings and hence gain richer and more complex insights into aspects of the subjects they are studying. [30]

Shift in perspective leads to change of language (which may be natural, formal or symbolic), discourse and personality [45]. This means that there is potential to detect whether a threshold has been crossed by talking to learners and study what they are talking about and how they use language.

3.2 Concept mapping

Concept maps can be used as a tool to graphically illustrate the perceptions of concepts and how concepts are interrelated – a means for structuring and representing knowledge visually. Concept mapping has been used for evaluation, learning, teaching and research, for example in engineering [58].

A central notion in this context is *concept*, and what is meant by a concept may vary in different contexts; however, Novak and Cañas, two ardent advocates of concept maps, define the term in the following way:

> *Concept* – a perceived regularity in events or objects, or records of events or objects, designated by a label. [47]

A concept map consists of rectangles or ellipses containing a word that identifies the concept, and hence these symbols represent the concepts. The concepts are connected with lines that implicate that the concepts are related

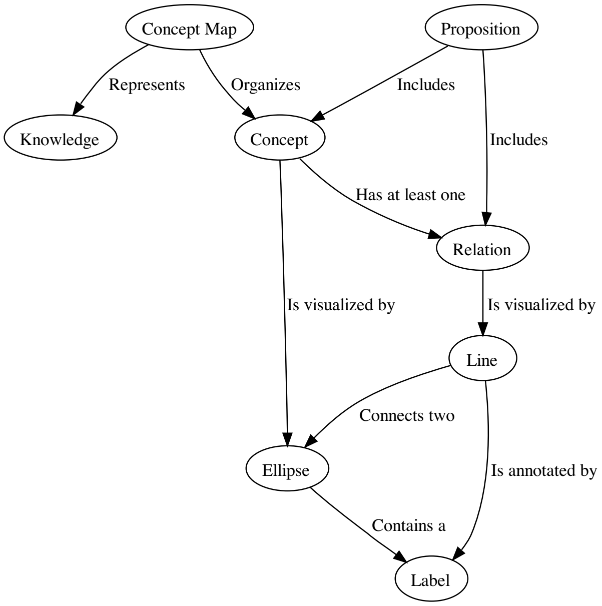

Figure 3.1: A meta concept map. This diagram organizes knowledge on concept maps. The ellipses and their labels act as the concepts and the lines represent the relations between the concepts. Text labels explain the nature of the relation. A proposition is a statement that can be read from the graph, e.g., "A concept map represents knowledge" or "A concept is visualized by an ellipse."

and a textual label on the line explains the semantics, i.e., how they are related. In addition, relations can be directed as indicated by arrows. Thus a network of associated knowledge is built, often hierarchical by nature with general and broad terms in the top and more specific concepts further down in the chart. Figure 3.1 demonstrates a simple concept map that is an attempt to structure knowledge on concept maps.

Concept maps can be used as tools for learning but also for the evaluation of knowledge. One way to investigate knowledge which is represented by a concept map is by looking at the "sentences", or *propositions*, which can be seen in the network:

> *Propositions* are statements about some object or event in the universe, either naturally occurring or constructed. Propositions contain two or more concepts connected using linking words or phrases to form a meaningful statement. Sometimes these are called semantic units, or units of meaning. [47]

For example, it is possible to determine whether the propositions are accurate, and thus concept maps can be used to smoothly collect data and identify common misconceptions among pupils and students [17, 47]. There are proposals to teaching models based on concept mapping. Some of these ideas suggest that a number of concepts related to a particular learning object are given to learners who should organize them into a concept map. However, in less directed learning situations, concepts need not be given in advance [47].

3.3 Phenomenography

Marton and Booth point out that people do things differently, and they learn to do these things in different ways; some do it worse and some do it better [37]. Phenomenography originated in educational questions of how learning comes about and how it is possible to improve the learning process. Amongst other things, Marton and Säljö were interested in deep and surface approaches to learning and gave contributions to that field of research, e.g., [39, 40].

It gradually evolved and matured into a research tradition that concerns how different aspects of the world appear to some group of people. Essentially, the studies within this approach are explorative and use empirical data, and they all take a second order perspective on some phenomena. That is to say, the phenomenographer does not study the phenomena as what they are (the first order perspective), but the variation of what they are as experienced and expressed by people (the second order perspective). Marton, one of the pioneers of Phenomenography, gives the following definition of this research specialization:

> Phenomenography is a research method adapted for mapping the qualitatively different ways in which people experience, conceptualize, perceive, and understand various aspects of, and phenomena in, the world around them. [35]

Consequently, the object of study is the relation between a certain phenomenon and a group of people and the variations of the relation. It is neither the phenomenon nor the people it tries to explain; it is the group's experience of the phenomenon, see Figure 3.2.

The ontology of Phenomenography is non-dualistic, which means that it does not separate the observer from the observed (object and subject). Marton [36] explains it in the following way:

> There is only one world, a really existing world, which is experienced and understood in different ways by human beings. It is simultaneously objective and subjective. An experience is a relationship between object and subject, encompassing both. The experience is as much an aspect of the object as it is of the subject. [36]

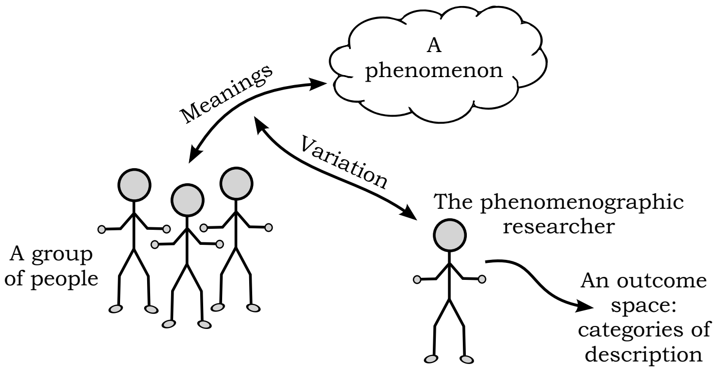
A phenomenon
Meanings
Variation
A group of people
The phenomenographic researcher
An outcome space: categories of description

Figure 3.2: The object of study for a phenomenographic researcher is the variation in the relation between a certain phenomenon and a group of people. The primary result of the analysis is a limited set of qualitatively distinct categories of description. The categories and the relations between them constitute the outcome space.

In this non-dualistic world, the set of different ways to experience an object is what actually constitutes it. Moreover, because the experiences all relate to this constitution, they are all logically related. A prominent feature of Phenomenography, compared to other qualitative research traditions, is thereby, the way in which the results are structured.

Experiences from earlier studies had shown that different people described phenomena in only a few different ways, and that led to a fundamental epistemological assumption, namely that there are only a limited set of qualitatively distinct ways to experience and describe a phenomenon. Each qualitatively distinct way to experience forms a category of description. In addition, there is always a set of logical relations between the categories, and the logical structure in combination with the categories of description constitutes the outcome space. In this way, the outcome space contains a rich set of information of how the phenomenon is experienced and how these experiences relate to each other.

Moreover, there is no explicit connection to the experiences of any individual person in the outcome space. Each category describes a particular way to experience a certain phenomenon, observed in the collective, and is thereby constituted by merged fragments of meaning found in the individual's description of the phenomenon. The collective outlook is a quality that distinguishes Phenomenography from qualitative research in general, which is often described as taking the individual's perspective [12].

As in its origin, the most common application for the research approach is still to study different aspects of learning and teaching. However, Phenomenography is not restricted to that area only. Bowden [9] divides the research approach into two forms: the applied (or developmental) form, and the pure form, separated from institutional learning environments.

> Phenomenographic research methods of data collection and analysis can be used to study a range of issues, including approaches to learning, approaches to teaching, understanding of scientific phenomena learned in school, or understanding of general issues in society unrelated to educational systems. [9]

Marton and Booth emphasize that all of the frequently used terms in phenomenographic publications, such as “conceptions”, “conceptualizations”, “ways of understanding”, “ways of comprehending”, are all synonyms for “ways of experiencing”. One should not understand them as referring to the internal mechanisms in the human brain. The phenomenographic researchers always allude to the experiential sense of the words, all in line with the non-dualistic approach [37].

The research process

Bowden [9] outlines the phenomenographic research process as having four stages: plan, data collection, analysis and interpretation. In all of these stages, the researcher must maintain focused on the purpose of the study. This is important to consider for obtaining trustworthy results. As in all research, it starts with a plan that defines the purpose and the strategies. Naturally, what drives the research is an underlying question that the research activity tries to answer. Students’ difficulties in coping with physics gave Bowden a good reason to try to make sense of the students’ understanding of important concepts in physics.

The input data for a phenomenographic investigation is essentially people’s statements of experiences of a phenomenon. The predominant method for collecting this data is through semi-structured interviews with people, and the researcher must select the persons carefully and consider why they are a good choice. In semi-structured interviews, the interviewer poses open-ended questions that address the problem area or asks the subjects to talk about what the phenomenon X means to them.

Even though the semi-structured interviews should be planned on beforehand, they can take different directions and follow the spontaneous thoughts that might appear differently from case to case. The next phase is the analysis of the data, which often starts by transcribing the recorded interviews. The texts are then sought for different meanings and the contexts they appear in. Sometimes phenomenographers decontextualize the fragments of meaning, and sometimes not. In either way, the meanings constitute a pool, from which the categories are condensed.

The categories should relate to each other logically. If not, the researcher should reconsider the data again. Section 5.1 elaborates more on the analysis process. Finally, the results should be interpreted according to the purpose of the study. In applied, developmental Phenomenography, the interpretation is a natural consequence of the posed research question. If the result tells how students experience phenomena in an educational context, the teachers can use the results to reflect upon their pedagogy and instruction. They can adapt their way of how they present new concepts, or they can get a better understanding of why students fail to do certain tasks.

A pure phenomenographic study, on the other hand, might have only the purpose to describe the experience of a phenomenon, without any further implications. In all cases the results of the study must be seen in the light of its purpose, and if a researcher wants to use the results in a different context, this issue must be taken in consideration.

3.4 Variation theory

In Paper V through VIII, a phenomenographic perspective is taken, which makes the assumption that there is a limited number of qualitatively different ways to experience a learning object in a certain group of people. Moreover, phenomenographers mean that an important prerequisite for learning is the ability to discern critical aspects of the learning object. Variation theory shares the basic concepts of learning with Phenomenography and provides a theory for how to give conditions for learners to identify critical aspects and thereby get a richer understanding.

Above all, what should be learned must exist in a context that is meaningful to the learner; that the proper relevance structure is provided [37, p.140, p.155]. Then instruction can be enhanced by helping students to discern the critical aspects of a particular learning object. However, a mere listing of facts is not enough to ensure a rich understanding of complex phenomena; the learner must be aware of the different aspects involved and how they interact. This can be achieved by introducing carefully selected variations in what the learner takes in through his or her senses [37, p.145, p.152].

The idea is to highlight relevant features of a learning object, for example by altering the “value” of some aspects while others are kept unchanged. A *dimension of variation* is spanned by all the possible values for some associated property or aspect of the learning object. Empirical research on teaching has been able to discern four different patterns of variation that use different combinations of invariance and variability: *Contrast*, *Separation*, *Generalization* and *Fusion* [41][38]. In addition, the variations will help the learner to break the natural attitude, which is required to start a reflection [37, p.148].

Variations can reach over long periods and must not necessarily be sudden. One well known and practiced variation in computer science is the learning of programming (in depth) by learning many different programming languages and programming paradigms. It can be compared to what happens when a young person learns a second language. The learner becomes aware of properties of languages that he or she never had to think about before and starts to see how languages are structured and their differences and similarities.

By analysing the phenomenographic outcome spaces it is possible to identify critical aspects of learning a specific learning object, and then try to find the corresponding dimensions of variation that allows to see the critical aspects. Thuné and Eckerdal [57] give suggestions for computer science teaching based on Phenomenography and Variation Theory and report successful results from a pilot study that used the suggested variations and then assessed the learning experiences. Paper VI, VII, and VIII use phenomenographic outcome spaces and Variation Theory to give some proposals for explicit variations in teaching.

4. Collecting data

My colleagues and I chose to study learning in Computer Science from the student's perspective. Using a number of methods we collected various types of empirical data, which were analysed using different qualitative research approaches. The data provides information on how students themselves describe aspects of the subject area of computer science; additional data are collected solutions to assignments which students have undertaken.

In study I through IV, data were gathered from students in the UK, Sweden, and the United States. The Swedish material was translated into English before its data were analysed.

In study V through VIII, data were gathered from students at Swedish universities. The interviews were conducted in Swedish and were also transcribed and analysed in this language. Research findings were subsequently reported in English.

4.1 Interviewing informants

A number of studies used semi-structured research interviews as data collection method (Paper I, V, VI, VII, and VIII). The interviews were recorded and transcribed into written form. Semi-structured interviews are often based on a few prepared themes and questions. In this interview technique, it is vital for the interviewer to be sensitive to the interviewee and to come up with follow-up questions in response to answers from the interviewee, and it is not possible to make a detailed plan for this [37, pp.129–132]. The interviewer must be prepared to rearrange the order of the questions and bring in new follow-up questions to catch the student's spontaneous reflections. Due to the dynamic nature of these interviews it is important to ensure that the interview covers all of the prepared questions and themes.

The length of the interviews in the various studies has varied. In the studies described in Paper V through VIII, the interviews took between 45 and 90 minutes to complete. The collected material is extensive and contains rich data on the informants' views on the phenomena that were addressed in the studies.

4.2 Doing role-play and interviews

Paper V describes a methodology used in Paper VI to gather data on how students handle a development assignment of professional character and how they perceive related object-oriented concepts. The informants were participating in an experiment in the form of a role-play in which they were acting as a new hired programmer at a company where they suddenly had to independently complete a software development project. After completing the work, each informant was interviewed about how he or she had experienced the situation, how the mission was conceived, and what kind of system they had worked on and how they went about it. In addition, informants were interviewed on a number of concepts that were central in this context. In addition, the entire directory tree for the development environment that had been used was saved, which means that it was possible to examine what and in what order files had been added and changed, which scripts that the informants had executed, et cetera. All events on the computer screen during their work were also recorded on tape.

4.3 Collecting concept maps

In the study reported in Paper II, students and other informants were asked to perform a task in which they would explain their view of object orientation and how different concepts in object-oriented programming linked, through designing concept maps (see Section 3.2). Data were collected primarily during programming lessons, but also by recruiting informants otherwise. The written instructions gave an example of how a person had developed his or her thoughts on the concepts of "home" and "kitchen" by drawing a concept map describing the concepts of "home" and how they could be associated with one another. It was clear from the instruction that the concepts are drawn as ellipses and that the relationships between concepts are drawn with arrows that have labels explaining how these concepts are interrelated. In addition, the instruction clarified that the example was not complete and it was not the only way to describe a home.

Students were instructed to start with the concepts of "class" and "instance" and build a concept map that would summarize their understanding of object orientation and object-oriented concepts. In total, we collected concept maps from 107 informants at six institutions in three countries. Among these there were 71 beginners, 12 in the middle of their studies, 15 graduating students and 9 teachers. All informants had studied object-oriented programming.

4.4 Collecting biographies

In the following, I use the terms "narrative" and "biography" synonymously, and I have taken the liberty to use and interpret the terms without discussing their connection to theories and research traditions in detail.

Ahlberg [1] used both written and oral narratives when she was investigating the possibilities to find "view-turns" – experiences of changing ways of experiencing, among 126 final year students during practice in health service. The students who agreed to participate in the first part of her study wrote their narratives in connection with a lecture and they were given at least 20 minutes to accomplish the task. They were given a blank sheet of paper and an instruction asking them to describe a situation in their clinical practice where their view of this situation changed from seeing it as a problem to seeing it as a success. Then they were also asked to reflect on what was of vital importance for the change. In order to get richer narratives in the main part of her study, she collected narratives using semi-structured interviews. In several ways, her study relates to our research and methods in Paper I, III and IV. She has a similar research question, she uses qualitative methods, and she uses narratives (biographies) and semi-structured interviews. Finally she takes a phenomenographic approach as I do in Paper V through VIII.

Narratives or biographies have also been used in studies related to Computer Science Education Research. When Schulte and Knobelsdorf [55] explored how individuals recall and interpret their computing experiences in retrospect, they used biographical research and their work inspired us when we designed our study.

We have obtained data by collecting students' own stories of how they have experienced learning in computer science (Paper III and IV). Data were collected by asking students to write and submit "transformation biographies" that should describe a situation in their life where some aspect of computing had changed them or their way to view the computing subject. Inspired by the work of Schulte and Knobelsdorf, we gave the informants two made-up stories as a way to focus the structure of the student biographies. As most of the biographies were collected as home assignments in English writing classes, the students were given quite some time to reflect. We believed this should allow the students to think about their experiences and remember important learning episodes. In addition, the formal writing style was emphasized in the classes and it was hoped that this requirement would make it easier to read and compare biographies with each other. Data were collected from 86 computing major students in the second half of their programs. They came from 5 institutions in 3 different countries. The majority were collected in the US and the UK. The relatively few Swedish biographies were translated into English prior to analysis.

5. Data analysis

In the quest for empirical evidence, the various collected data has been analysed using different approaches, mainly of qualitative nature. These approaches are described in the present chapter, and in the end trustworthiness in qualitative research is discussed.

5.1 Phenomenographic analysis

In phenomenographic analysis, the researcher refines the primary source of data by transcribing recorded interviews into a textual form where the participants' quotes are anonymous. However, it is still possible to separate individuals by using pseudo names. The next step is to search the texts for different expressions of meaning that relate to a certain phenomenon.

> Phenomenographic analysis - whether it is seen as construction or discovery - focuses on the relationship between the interviewee and the phenomenon as the transcripts reveal it. [61]

These manifestations of meaning can be identified in several places and different forms in the text. Meanings are found where the interviewee explicitly describes her experience of the phenomenon as such. However, implicit descriptions can also reveal meanings, as in descriptions of how she uses the phenomenon, or which purposes, advantages or drawbacks this phenomenon brings about.

The meanings are expressed by quotes that form a large collection of further refined data. The quotes are usually decontextualized from the text, but the reference to their context should be kept, to maintain the possibility to re-interpret their meaning. The purpose of decontextualisation is to be able to find qualitatively different meanings, experiences and understandings of the focused phenomena, on a collective level. Some meanings stand out from others, while some have something in common with other ways to experience.

The fragments of meaning are in this way condensed into clusters of meaning that are abstracted and outlined in categories of description. It is important to understand that the categories do not express any particular individual's understanding; rather they are the result of an analytical categorization of the meanings found on the collective level. In the process of forming categories, one should search for different dimensions in such a way that each category opens a new dimension of understanding the phenomenon and its meaning, or new relations between dimensions. This avoids categories that are instances or variations within the same dimension.

The phenomenographic outcome space is distinguished by the categories of description and their mutual logical relations, usually the hierarchic inclusiveness which implies that the meaning of the categories include each other in the sense that a certain understanding also includes or implies a similar, more elementary understanding. As Phenomenography originated in studies that in one way or another aimed to understand or improve formal learning, it is reasonable to range the outcome space in a hierarchy where the quality of each category is valued by some measure of compliance to the educational goals.

> Thus, we seek an identifiably hierarchical structure of increasing complexity, inclusivity, or specificity in the categories, according to which the quality of each one can be weighted against that of the others. [37, p.126]

Consider the structured and interrelated outcome space in contrast to sociological research traditions where it is usual to make categories without any requirement of internal logical relations. It is important to emphasize the clear and inevitable relation between the result and the subject field that includes or views the phenomenon. This is an argument for taking a phenomenographic perspective in educational research within a specific subject field.

On the other hand, it is vital to make clear that the analysis is not a matter of sorting the subjects' conceptions into a predefined structure. One of the fundamental epistemological assumptions within Phenomenography is the relations between the categories of description. The various ways in which a phenomenon can be experienced are logically connected to each other through the phenomenon itself and the structure of the logical relations is typically hierarchically inclusive.

Another property of the outcome space is the collective level of descriptions constituted in the categories. It is not the case that all individuals or a specific individual have a certain structure of their way to experience. Rather, the analyst tries to constitute the categories on a collective level and if successful, the well-founded categories can be structured and interrelated. This is what the phenomenographic researchers are striving to achieve.

Marton and Booth [37] describe three principal criteria for the expected properties of an outcome space constituted of categories of description. The first criterion is that each category should have a distinct and unique relation to the phenomenon according to a distinguished way of experiencing it. This is motivated by the fact that Phenomenography is a pedagogical research specialization, focused on learning, with the goal to obtain a clear picture of qualitatively distinct ways to experience phenomena that have a connection to learning.

The second criterion is that the categories must have a logical relation to each other that often is hierarchical and often is inclusive as well. From a pedagogical perspective there is a norm that defines which ways of understanding (experiencing) a certain phenomenon is preferable before others. The pedagogical goal is often that the learner should be able to experience phenomena in a more extensive, complex or specialized way and therefore a hierarchical structure of the categories is sought that corresponds to this goal.

The third criterion is that the system of categories should be as compact as possible. This implies that there should not be more categories than necessary to express the critical experiences and the differences between them. [37]

A definition of inclusive categories

As discussed in the previous section, the hierarchic structure of the outcome space can be explained by inclusiveness of conceptions. Inclusiveness in this context means that a certain way to experience a phenomenon also includes another way of experiencing it.

However, the description of inclusiveness is a bit vague and we need to consider what it means to say that one way of experiencing also includes another way. I have proposed the following definition of inclusiveness for my studies. Given that there is a category, A, that codes a particular way to experience and describe a phenomena. Then category A is included in another category, B, if their relation fulfils the following conditions:

- There is a non-contradictory relation between category A and B, and
- The relation is of the type B consists of A, or B is an augment of A, or
- Something in B assumes A.

During the data analysis, we used this definition to study and establish the categories' internal relations and plausibility related to the other categories in the outcome space.

5.2 Content analysis

Similarly to the phenomenographic analysis process, content analysis is a qualitative approach used to understand and categorize meanings in written text. Fragments of meaning are coded and then the different codes are synthesized into separate categories that represent different qualities of meaning. A major difference from phenomenographic analysis is, however, that the categories that emerge from content analysis do not necessarily need to be related to each other.

Content analysis can be used to analyse data with two different approaches, either an inductive, data-driven – *conventional content analysis*, or with a deductive, theory-driven – *directed content analysis* [25].

In the studies reported in Paper I, III and IV, we have analysed transcripts of interviews and writing biographies based on the theory of Threshold Concepts. We have mainly used a deductive approach where we have assumed that Threshold Concepts related to learning of various subjects in computer science exist. We have been looking for evidence in the collected texts by searching for traces of the most important key characteristics that define threshold concepts: transformative, integrative, irreversible, and that they are troublesome. On the other hand, we have used inductive analysis to make interesting discoveries in our data, for example, to find and explore concepts and other aspects that came up in connection with transformations, such as how students reflects and identify with the computing field.

5.3 Concept map analysis

We decided to analyse the concept maps after coding them to graphs. This meant that each concept map was transformed into a graph with nodes and edges. Then we analysed the graphs in a number of different ways. We calculated numerical characteristics, such as maximum depth and distance between certain nodes. Naturally, we investigated how many different concepts that the informants used and how these concepts were related. This was addressed, e.g., by analysing and counting the propositions (see Section 3.2), which we called "sentences", i.e., the combination of the (directed) triplets node, edge, node. The analysis was a mixture of quantitative and qualitative methods.

5.4 Trustworthiness

As in all research, the qualitative researcher wants to be heard and believed by other researchers and receivers. The fundamental condition to achieve this goal is to uphold trustworthiness and to deliver credible results. In qualitative research, it is crucial to show that the chosen research methods reflect the goals for the research in a suitable manner, and to show how to use the results.

Booth [7] discusses these matters through accounting for her own long experience of programming and her familiarity and good relations with the students who participated in her study. She describes the exhaustive and open-ended interviews and declares that the transcripts are open for other researchers to read. In this research, there are no absolute truths and therefore, she claims, the researcher must argue convincingly for the chosen methods, the results, and the interpretations. This can take place in seminars, presentations and by peer-reviewed articles within the research tradition.

She explains that, due to the collective level of the results, the interviewees seldom confirm credibility; the individual's experience is not traceable, and the researcher's interpretations go further than the individual's understanding at the time of the interview [7, pp.64-69, 90-92].

The phenomenographic analysis has a subjective nature since it reflects the researcher's way to discover and experience the meanings within the text material. For instance, Walsh [61] discusses openly and critically concerning the logical relations between categories and whether the categories are discovered or constructed and what the difference is between these approaches within the phenomenographic analysis.

In a study on applied Phenomenography, Åkerlind [2] finds a common conduct in the analytical process when it comes to keeping an open mind, suppress own preconceptions, focus on the whole, the search for variations in meanings and relations between them and the iterative process using restructuring and tentative categories. However, Åkerlind also identifies areas within the analysis where there are variations in the practice. Some researchers use de-contextualized fragments of meaning and others do not. The collected data are handled in different manners and the principle of letting the logical structure follow the data as close as possible is sometimes compromised by the desire to reflect the researcher's professional classifications. The collaboration with other researchers varies from individual analysis to collaborative analysis [2].

Lincoln and Guba discuss trustworthiness within qualitative research and argue that this research, in contrast to positivist traditions, is inevitably associated with subjective values [33, pp.37-38]. Hence, in order for the researcher to be trustworthy, in the sense that the audience thinks it is worthwhile to take part of the results, it is of great importance to communicate how the researcher is reasoning and to account for both the data and the researcher's interpretations. They point out that trustworthiness cannot be judged by the same measures as in the positivist science tradition. Instead, they suggest four alternate terms that replace the traditional terminology.

> The four terms 'credibility,' 'transferability,' 'dependability' and 'confirmability' are, then, the naturalist's equivalents for the conventional terms 'internal validity,' 'external validity,' 'reliability' and 'objectivity'. [33]

The credible researcher should persist long enough to ensure that a sufficient and unbiased amount of data is gathered. The critical aspects should be studied in depth using different angles, various data sources, methods, researchers and theories (triangulation). The raw data must be available for reexaminations and the informants should comment on the results.

The descriptions should be so rich and thick that someone who is interested in making a transfer to another context should be able to decide if that is possible or not. The data sources should be selected to maximize variations.

The dependability of the results can be increased if the research group is split and each sub-group deals with data independently and compare the results (stepwise replication), or an auditor could examine the data, the process and the produced results and then see if the conclusions are similar.

It should be possible to conduct a confirmability audit trail (log) by reviewing the raw data, the analysis and synthesis, and the documentation of the process and other documents. This ensures that the results are products of the conducted investigation and are not products of the researcher's preconceptions [33].

Mulholland and Wallace [46] suggest a method to present results from narrative studies in a legitimate and trustworthy manner, by dividing the presentation in three stories. The first story shows the strength and credibility of the study. It contains a story told by the subject of the research – the data. The second story is told as the researcher's interpretations of the first story. The third story adds a theoretical model to the first two stories. The researchers give suggestions for how the experiences of the inquiry can be useful, for example, how they can improve educational matters for the participant, the researcher and the reader. Finally, the researchers account for how the study has influenced themselves and their view on teaching. Moreover, the first two stories can be re-read using the new perspectives gained from the third story.

We can conclude that a very important asset in obtaining trustworthiness in qualitative research is a rich set of data that can be shared with the reader in various ways, together with the researcher's interpretations and conclusions.

Trustworthiness in my research

In the research on Threshold Concepts, the analysis has been done in true collaboration among the researchers in the research group. Sometimes individual interpretations and ideas have been subject for discussions where it has been important that all agree on a common interpretation. Our intention has always been to present as much evidence and data as possible, and it is possible to contact us and discuss our interpretations by e-mail or otherwise. We have also used different methods for obtaining data and for analysis. During this research our results have been presented regularly at different conferences and we have been open for discussions.

In the phenomenographic research, I have been supported and reviewed by two other researchers with an interest in education, and sometimes my first interpretations of data have been reconsidered after discussions and good advice. In my reports, I present quotes richly together with my interpretations, which enables the reader to build an opinion of the data and how I have reasoned during the analysis. I have collected data from informants that represent the type of students which I address in my research questions and when the participants were recruited I aimed for a variation in age, gender and experience to avoid a one-sided view of the phenomena in question.

6. The main results

My research is all about learning concepts and the various ways in which concepts and phenomena are experienced. All research and results are based on empirical investigations and mainly on qualitative analysis. One theme tried to answer the questions about if there are concepts in computer science that transforms the students, what these concepts are, and in which ways they transform students. The other theme concerns how prepared students are for a future profession in the software industry, by learning how students see concepts, tools and phenomena that are important for a profession in the software industry, and how they respond to situations that are similar to what they can expect at work. The following sections summarize the most important results from the research in these themes. Brief overviews of the results are presented in Table 6.1 and Table 6.2.

6.1 Learning and understanding concepts

This section presents the results that emerged from the research on questions raised in my first research theme. Table 6.1 presents a summary of what is addressed in the papers belonging to this theme and which kind of results they show.

Are there Threshold Concepts in Computer Science – troublesome?

Early in our research on troublesome concepts, we could conclude that there are certain concepts in Computer Science that, at least partly, meet the criteria that define Threshold Concepts. In Paper I, the first in a series of studies reporting on Threshold Concepts in Computer Science, we could point out two main concepts after analysing interviews with students. The first was labelled as “pointers/memory”, which is a rather specific concept, and the second was “object-oriented programming”, which is a broader concept. This relates to the terms “localized threshold” and “overwhelming threshold”, used by Flanagan and Smith [20].

In which ways is the concept “object orientation” integrative?

Due to the many underlying concepts that constitute the overarching “super concept” object orientation, we decided to further investigate the ways in which students perceive some of the concepts associated to object orientation.

Table 6.1: *Summary of the results on Threshold Concepts.*

Aims	**Paper**	**Data**	**Analysis**	**Result**
Orientation	I	Questionnaires & interviews[a]	Mixed[b]	Potential thresholds, e.g., OO & abstraction
Find TCs	I	Interviews[c]	Content[d]	Indications of TCs in CS, e.g., OO & pointers
OO views	II	Concept maps[e]	Mixed	Concepts in OO are not integrated by students
Abstraction?	III	Biographies[f]	Content	Some abstract concepts are TCs, not "abstraction"
Find TCs	IV	Biographies	Content	Long detailed list of concepts divided in themes
Change	IV	Biographies	Content	Student identities: turning professional

[a]Questionnaires and informal interviews with instructors. [b]Mixed methods described in paper. [c]Semi-structured interviews with graduating students. [d]Content analysis. [e]Concept maps drawn by novice students. [f]Biographies written by non-novice students.

As reported in Paper II, we studied how students understand object orientation and object-oriented programming. We investigated which concepts the students related to these topics and how they related these concepts to each other. The data was collected by asking students to draw *concept maps*. They were instructed to start by inserting the terms *class* and *instance* into the map and then they should add any related concepts that they could think of, and describe the relationship between all the concepts, using arrows and labels. The results of this study show that the students describe a static perspective on object orientation. There is nothing in the data that suggests any form of interaction between objects in an executing program.

Only in a few cases, links to real objects in the outside world occurred, indicating that the idea of creating a model of reality was not in the students' foci. Neither could we find relations to tangible things such as users, hardware and memory.

Although the object-oriented concept of "inheritance" was related to "class" in a number of cases, it was only in a few cases that the informants related to other central object-oriented concepts such as "encapsulation", "abstraction" and "polymorphism". However, it turned out that the students connect both data and behaviour to the class concept, which stands in contrast to previous results indicating that students often only see the class as a data container.

The general impression was that most of the novice students in the study had not crossed the threshold yet; they had not fully integrated the necessary constituents into an understanding of object orientation as a whole.

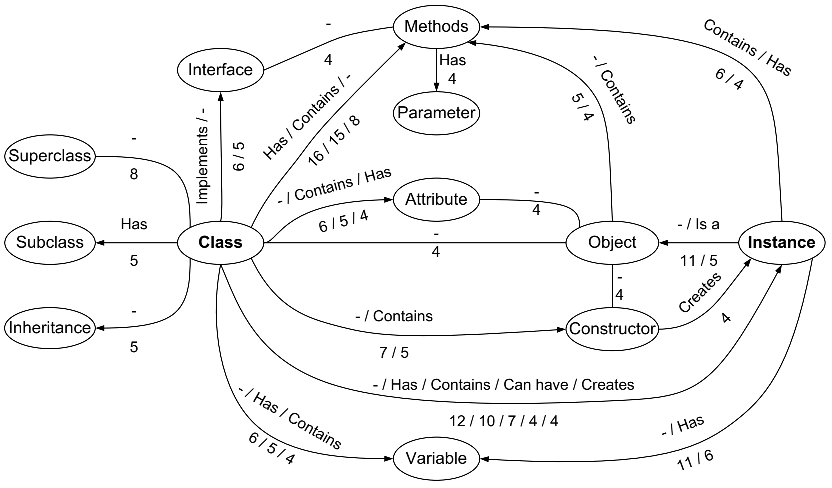

Figure 6.1: This aggregated concept map represents the *collective view* of object oriented concepts as expressed by the informants' concept maps. The informants were first and second year students in computer science. The aggregated map is built by the most frequent *propositions*, including all with a frequency of four occurrences. The concepts *class* and *instance* were given at the beginning.

Figure 6.1 summarizes many of the tabular results in Paper II, now represented by a synthesized concept map that includes all "propositions" ("sentences") that were represented in more than four cases in the data. For instance we can see in the figure that *class* is related to *inheritance* in only five cases, and that the relation between classes and methods in 16 cases is *has*, in 15 cases is *contains*, and in 8 cases have no label at all.

The results indicate that object-oriented programming teachers should take some time to clearly explain the differences between class and instance/object, and give many examples of how the objects in our environment can be modelled with the class as a design, or blueprint of its instances, which are the models' representations of the real objects. We also noted that it should be a very useful exercise for students to make concept maps. It is something they could do several times during a course.

The transformative nature of concepts

After studying how students perceive the connections in one of the overall Threshold Concept, object-oriented programming, which we had previously identified, we wanted to go further and examine students' own experiences of things that had led to some form of *transformation* in education (or earlier). As described in Paper III, we collected data from students through written reports about something that changed their way of looking at or experiencing Computer Science. The analysis showed that many of these stories in one way or another could be linked to understanding or learning to use various forms of abstraction. Previously, analysis of questionnaires and informal interviews with teachers indicated that *abstraction* could be a likely candidate for a Threshold Concept at a global level (Paper I). In Paper III, we examined things that change the students' perspectives on Computer Science and in which particular ways these things are related to abstraction. The empirical results did provide that 47 out of 86 students' stories discussed various forms of abstraction, but only in one case abstraction was mentioned explicitly. We identified the following concepts: *modularity* (previously they wrote long linear programs – now the programs were split in parts, abstracted into distinct components), *data abstraction* (now, the data can be combined to form various types of aggregations), *OO-concepts* (inheritance and polymorphism, code reuse, design patterns), and *complexity* (algorithm analysis).

The biographies show that students often connect their learning (transformation) to concrete examples of problems they used to have and that they have made generalizations from these experiences. Other students have learned some abstract concept, but did not fully understand or appreciate it before they had implemented it. The transformations have in most cases led to a change of behaviour and they seem to be connected with an understanding of *why* rather that what. In spite of the variety of different topics in computing, most of the transformations occurred when the students were involved in designing software and making programs.

The conclusion is that we have no evidence to claim that abstraction in itself is a Threshold Concept, even if it is a key concept and a necessary ability. However, some of the identified concepts that we related to abstraction may be thresholds.

In Paper IV, we wanted to further study the collected biographies to investigate in which ways the students have been transformed, if they have started to think, act and identify themselves more as a Computer Scientist, if they reflect on their own knowledge, and which of the concepts are those that have transformed them.

The empirical results indicate that thresholds are very individual. The analysis resulted in a large number of different concepts, at least 75 different examples. It was quite difficult, but we could find a thematic categorization: programming, software design and engineering, object orientation, efficiency, technology, and personal development. We could confirm that the overall picture is that the students, who wrote the biographies really had begun to think, act, and identify themselves as computer scientists.

Table 6.2: *Summary of the results on students' views of concepts related to profession.*

Phenomena	**Paper**	**Data**	**Analysis**	**Result**
Java Interface	VI	Interviews[a]	Phenomen.[b]	4 P-categories[c]
Plugin concept	[8][d]	Interviews	Phenomen.	2 P-categories
A software system	VI	Interviews	Phenomen.	3 P-categories
Problem solving	VI	Interviews[e]	Content an.[f]	3 C-categories[g]
Software development	VII	Interviews	Phenomen.	4 P-categories
Software maintenance	VII	Interviews	Phenomen.	4 P-categories
Class diagram use	VIII	Interviews	Phenomen.	3 P-categories
Class diagram	VIII	Interviews	Phenomen.	3 P-categories
Class diagram symbols	VIII	Interviews	Phenomen.	4 P-categories

[a]Semi-structured interviews. [b]Phenomenographic analysis. [c]Phenomenographic inclusive categories of description. [d]Excluded in Paper VI. [e]Interviews and "traces" left in the computers' file system. [f]Content analysis of interviews and analysis of the file system. [g]Content analysis categories.

6.2 Understanding concepts related to the profession

Paper V, VI, VII, and VIII used a phenomenographic approach to explore the ways in which students see a number of concepts and phenomena that are related to the realities for software professionals. The concepts *Java interface*, *plugin*, and *software system* were addressed in a situated context, while the concepts *software development*, *software maintenance*, and *class diagram* were not. Table 6.2 gives an overview of the addressed concepts and the resulting phenomenographic outcome spaces that emerged from the phenomenographic analysis.

In Paper VI, VII, and VIII, the outcome spaces are extensively described using quotes and interpretations. Each outcome space is also briefly described in tabular form that enables the reader to get an overview. All outcome spaces are briefly described in the following text, and one outcome space from each study is presented in tabular form.

Students' understanding of software concepts in a situated context

The results give a picture of the students' different ways of seeing these concepts which in itself is interesting, but above all they can give useful implications for teaching. In Paper V and VI for example, the collective of informants who role-played around working with "large scale" software as "rookie developers", saw Java interfaces in a variety of ways (see Table 6.3). These different ways of seeing have varying relevance for the context of the software. The least relevant way of seeing is the interface as a "to-do list". The more advanced ways of understanding interfaces connect to the advanced understandings of the concept of plugins; the dynamic features of plugins can

Table 6.3: *Summary of the outcome space for students' views of the Java Interface.*

Cat.	**How the concept Interface is described**
1	The Interface is a text, in the form of a to-do list, that tells the programmer what to do; what operations he or she should write. It is an uncompleted program, skeleton code, or a template, to start from when a new class should be written.
2	The Interface is certainly defined by a text; however, the text constitutes an abstract "thing" that can be bound to a class by referring to the interface's name. The class is thereby obliged to have implementations for all the operations specified by the interface. In this way, the interface becomes a forcing contract. The programmer must implement the interface, and the interface has a name.
3	The interface is a data type for reference variables and thereby indirectly for objects. The interface, defined by text, has a name for the data type it represents. The data type can be used to create variables that can handle those objects that fit the content declaration. This is an expression of a meaningful relation between interface, class and object.
4	The interface is an open connection to new and unknown objects. The purpose of interface type handles is that they represent an open connection towards arbitrary objects that implement the same interface. Hence, the same handle can connect to several objects, defined by different classes. According to the descriptions, it is possible to replace objects and use different object types without having to change the rest of the software. Using interfaces allow objects to communicate with each other even though they are "strangers."

be obtained using interfaces and this requires an advanced understanding of interfaces.

Faced with the task, the students acted in three qualitatively different ways. One response was to read the documentation over and over again, but never doing anything concrete about the code. This was labelled as "Hands off". Another way, "Waterfall", was to read the documentation and after a while start to write program code in a new source file. However, they started from scratch and they did not compile the code. The third and most successful way used, "Prototype", was to identify and copy a source file with content similar to what they were supposed to develop. Then they inserted the new piece of software as a prototype which they could test and further develop iteratively. These results resemble what Perkins et al. categorized as "stoppers" and "movers" [51].

How students experience software development and software maintenance

Paper VII reports results from a study on how students experience software development and software maintenance. These broad concepts were chosen

Table 6.4: *Summary of the outcome space for students' views of software development.*

Cat.	How *software development* is described
1	Software development is described, mostly from a subjective perspective, as finding a solution to a problem or creating or building a computer program that solves a problem, that meets a need or realizes an idea.
2	Software development is described as finding out which functions and parts should be included in a program to meet the need and how they should be designed. Different design methods are described to achieve the goals.
3	Software development is described from a professional perspective as designing for the future; it is important that the software can handle future changes, be reused and that the design is documented so it can be understood.
4	Software development is described from a professional perspective as designing software and understanding what the customers and end users need, what can be achieved, the time frame, the economic aspects, and which methods to use.

with the aim to study how students experience "the big picture" and possible connections to the software *profession*. The informants' ways of talking about software development was divided in four distinct categories of description, see Table 6.4. A straightforward and relatively introvert way to see the concept is that software development is problem solving. The next category describes software development in terms of aspects of design of code, user interfaces and design methods. The third category includes a more advanced way to describe software development which brings in the future perspectives and the focus is on how to make the software reusable and maintainable. In Category four, the most advanced way of seeing software development is more extrovert and brings in the understanding of the importance of knowing and considering the needs from the end-users, the customers and the business requirements.

The basic understanding of software maintenance among the informants in the study is seeing maintenance as "debugging". Next level includes the understanding that maintenance is to add features to a piece of software required by a customer. Next category adds the understanding that software needs to be adapted to changes in the world around the software. The most advanced way to describe maintenance involves the understanding that maintenance is a natural and continuous part of the job as software professional.

How students understand class diagrams

The ability to represent design solutions using graphical notations is an important skill in professional contexts. It requires an understanding of object-oriented concepts as well as how they are manifested in a modelling language. In Paper VIII, it was investigated how students experience *class diagrams* in

Table 6.5: *Summary of the outcome space for students' views of class diagrams use.*

Cat.	How the use and purpose of class diagrams are described
1	Class diagrams are used as a documentation of existing program code which is good for someone who wants to learn about the software and perhaps is going to make changes to the program.
2	Class diagrams are used as a way to develop software designs. It is a tool that can be used to model a solution to a problem and therefore it also documents existing or non-existing program code.
3	Class diagrams are used as a way to design software and are used as a means for a dialogue with team members in a dynamic design process that will end up in program code. They are perfect to use as a tool to develop, discuss and test models on a whiteboard together with the team.

the notation used in the *Unified Modelling Language* (UML). The informants were presented with a set of class diagrams and in the interview these class diagrams were discussed in different ways. The analysis gave the results as summarized below.

Three different categories of description emerged for the purpose and use of class diagrams, see Table 6.5. The most elementary way to describe it was using class diagrams as documentation of code. A more advanced way to see was using class diagrams as a tool for software design. The most advanced way of seeing was class diagrams as a means for dialogue between, e.g., software developers.

Also the ways of seeing class diagrams per se was categorized in three categories. The straightforward way is to describe class diagrams as a diagram that shows classes and connections. A more sophisticated way is to see class diagrams as a diagram that shows in which ways classes are related. In the most advanced category, a class diagram is described as a diagram that shows how classes are related in hierarchies and structures.

The relation types symbolized by diamonds were described as relations (only), as something which means that one class "has" another, as something that means that one class contains or consists of other classes, and finally that there are differences between the white and black diamonds that indicate aggregation and composition.

Only few informants talked about the diamond symbols and related concepts in an advanced way. This can be contrasted with the observation that most students could identify the inheritance symbol (also indicated by Paper II), and that education seems to have succeeded in that particular sense.

Software professionals need to work with design models; however, some of the graduating students told me that they did not know much about UML. This may indicate that more emphasis should be put on learning the fundamentals of UML in programming and design classes.

6.3 On applying the results in teaching

One of the most important objectives for my research is to apply the findings in education. These aspects have been addressed in the enclosed papers and in this section I will try to give a brief summary.

Implications for teaching from the findings on Threshold Concepts

Pointers and memory are often problematic for students who learn older programming languages. The lack of pointers in Java, where objects are always "by reference", is often regarded to be an advantage of learning and using Java because of simplicity and less errors. Dereferencing is always done in one step and there is only one such operator.

However, the mental effort required to understand pointers may open up certain ways of seeing. Without pointers, the "notional machine" [13] may be more vague as well as the notion of memory. I do not suggest that Java should be abandoned; however, while eliminating troublesome concepts, the "portals" to a more comprehensive understanding of computers and programming may never be crossed. We should consider the risk of doing the students a disservice if the conceptual machine is totally abstracted away in education. Feasible assignments may be a substitute, e.g., to construct an object-oriented model of a computer.

The results also indicate that object orientation is a Threshold Concept, and previous research points out that understanding of object-oriented concepts needs to be strengthened after graduation [32]. These concepts need to be motivated by good examples closely connected to realistic software issues. A hypothesis of mine is that the mechanisms involved in object-orientated programming are easier to motivate and understand in depth if they are explained in a situated context, where the advantages and features appear naturally.

Creating connections to realistic situations

I assume that students, teachers, and the industry, have an interest in preparing students for a professional career. This includes good understanding of software development, in a wider perspective than what it takes to pass in a beginner's course in object-oriented programming. An education that targets people who want to work with system development and programming, or perhaps with administration of such activities as a manager, needs to give the students profound insights in aspects of software development.

To achieve this, what should be learned must exist in a context that is meaningful to the learner, that the learning situation has a proper relevance structure [37]. For a student who wants to be a software developer in the future, every new experience of working with software will contribute to learning. New situations give incentives for the individual to widen perspectives and reflect on previous experiences.

The participants in the study described in Paper V showed a constructive attitude to the situation during the role-play experiment and most of them said that they had learned from participating. This experiment has shown that it is possible to let students work with large and advanced software, that they can get into it, achieve tasks in a limited time frame, and learn. This indicates that it may be worthwhile to let students spend more time on doing the same kind of things as the professionals do, and actually elaborate on authentic software from the industry, or other communities.

However, learning using the "real thing" may require good basic skills and self-confidence. Not all students have the same prerequisites. If this is used in teaching, the instructor must carefully ensure that the students have a constructive attitude, and that they have the required basic knowledge. Neglecting to do so could discourage the students and affect their self-esteem negatively. A way to neutralize this could be to let the students work in pairs, as also suggested by [3].

My conclusion is that we should discuss these matters with local companies and try to find good examples of authentic software and tasks that would be encouraging and instructive for students. In the best of worlds, representatives from companies could give guest lectures and tell the real story about being a software professional. In Paper VII, the importance this may have for students is indicated by one of the informants.

Opening possibilities to discern new ways of seeing

Sometimes certain ways to understand tend to cement in the learner and then we must challenge these ways of seeing. Hence, we should consider how we introduce, motivate and discuss new concepts, and how we construct exercises. In Paper VI, VII and VIII I have tried to give examples on how Variation Theory (see Section 3.4) can be applied to the phenomenographic findings. In the resulting outcome spaces there are certain ways of seeing the concepts in question that are more valuable than the more concrete and surface understandings. These advanced understandings are the ones that we want to help the students to recognize. Looking carefully into which dimensions of variation that are needed to enable the desired understanding will give a foundation to build variations upon. Depending on the nature of the specific way to understand the concept, different patterns of variations can be used.

In summary, a Computer Science teacher informed by the different ways students may experience involved concepts and processes, can use this knowledge and create strategies for helping students with shallow ways of seeing to be aware of more profound views. Teachers' meditation upon students ways of seeing may also start reflections on their own perspectives on the subject and perhaps also open up new ways of seeing. In combination with knowledge of potential threshold concepts in the subject and variation theory, the teacher will have powerful tools.

7. Conclusions

Why are programming and Computer Science so difficult to learn – are there concepts that both hinder and enable students to understand the big picture?
Many concepts in computer science are truly complicated. Our studies on students' learning experiences have provided us with a number of examples of concepts that in the students' view were troublesome, hard to integrate and transforming. Some concepts were very specific and others had a more overarching character. Perhaps the most important lesson learned from this research is that we should not take for granted that all concepts introduced in teaching are easily adopted by the students or that their learning progresses in the way we may expect.

The students talked mostly about concepts related to programming, object orientation and software design, and in the narratives we saw that students often experienced problems when they were working with programming projects. Programming assignments and problem solving are natural parts of the computer science education tradition and our results stress their importance for conceptual understanding.

An implication is that more empirical research should be done on the ways in which students reach an impasse during such activities, how we can develop effective methods to detect when it happens and how we can help students to discover their way over the thresholds without giving them solutions. Using the definition given by Mayer and Land [44], we can now say that Threshold Concepts exist in Computer Science and that some of these concepts are closely involved in the process of transforming the students into thinking and acting like computer scientists. Teachers can use the notion of Threshold Concepts as a powerful tool to discuss and reflect on learning and teaching.

Traditionally, a syllabus reflects an expert view of which concepts should be included and emphasized in a course in Computer Science. Our empirical research on threshold concepts among students reflects the learners' view of which concepts they experience as learning challenges in different ways. A natural and interesting continuation of the research on threshold concepts and conceptions would be to closer study the different ways in which students experience the concepts we have identified as potential thresholds. A phenomenographic approach would be a natural way to investigate this. In combination with the expert view, such research could provide results that can be used for design of course syllabuses, and variation theory can be applied to help learners overcome the thresholds.

What can we do to enhance the possibilities for students to become well-prepared software professionals?

I have used phenomenography to identify different ways to understand a number of concepts which are important to master in order to be well-prepared for the software profession. My results reflect that the more advanced ways of seeing connect to each other. The uniting link is the expressed overall view and the connection to professional perspectives. The "high quality" descriptions all express an integrated understanding of important principles.

While the more advanced categories converge, the less advanced categories diverge and get more specific. An imaginary student, hypothetically equipped with only a concrete perspective, would not have the apprehensive view that ties the concepts together. Such fragmented understanding does not comprise a professional point of view! This student would be forced to catch up on these matters later, and this may of course be an obstacle for success at work.

As it is likely that similar ways of seeing would appear among students in comparable contexts, the findings are useful to teachers. I hope that those who read this thesis reflect on *their own* ways of seeing the concepts addressed in my research and which ways of seeing that are the most valuable in their opinion. Perhaps the studies presented here will inspire open-minded teachers to learn more about how *their students* understand these concepts and how the students experience their learning. Such knowledge will enable teachers to help students to discover advanced ways of understanding.

Knowing that most computer science majors are becoming software professionals, I hope that educators will consider the professional perspective. Teachers without experience of the software profession may be interested in company visits and inviting guest lecturers to their classes. For certain, students need the purely intellectual challenges, but they also need the opportunity to reflect upon the academic content in relation to its professional aspects.

I argue that professional perspectives should be a recurrent theme in teaching, as this would help students to better understand many computing concepts. Learning should take place in relevant contexts and it is not sufficient to just tell students how to think about things. It cannot be expected that all students understand intellectually challenging features which are only implicitly addressed in assignments and lectures. On the contrary, teachers should explicitly help students to see the advanced aspects of what they are teaching. To reach this goal, a teacher has to provoke their current views. I have suggested how to take advantage of my phenomenographic results using variation theory, which provides effective patterns for how teachers can help students to discover new ways of seeing by introducing variations.

In this way we can better match the students' expectations to the reality of the business. This may actually improve the efficiency of education regarding the society as a whole. Using this approach, we will increase the possibilities of providing students with the self-confidence and power needed to be well-prepared for being successful at work as software professionals!

Svensk sammanfattning

På väg mot en profession som utvecklare av mjukvara

Tidigare studier har visat att det är vanligt att universitetsstudenter har problem med att lära sig ämnet datavetenskap. Framförallt har forskningen intresserat sig för studenters svårigheter med att lära sig programmering vilket är ett ämnesområde som förutsätter god förståelse av en mängd begrepp samt praktiska färdigheter. Det har bland annat visat sig att studenter har problem med att skriva, läsa, korrigera och designa programkod. Framförallt har studierna intresserat sig för studenters förmågor och problem under den tidigare delen av sin utbildning.

De flesta som studerar på utbildningsprogram med datavetenskap som *huvudämne* är inriktade mot att arbeta professionellt med programvara när de är färdiga med sin utbildning. Forskning på akademiskt utbildade professionella programmerare har påvisat att de som nyanställda hade kunskapsluckor som de var tvungna att komplettera på egen hand. Det är alltså viktigt att undersöka hur de studenter som studerar dataämnen kan förberedas bättre inför sin framtida profession, för att underlätta övergången till yrkeslivet.

För att öka förståelsen om lärande i datavetenskap och hur man bättre kan förbereda studenter för de krav som ställs i det professionella yrkeslivet, har två olika ansatser använts för att studera studenters relation till olika datavetenskapliga begrepp.

I det första temat har en internationellt sammansatt forskargrupp undersökt vilka begrepp och företeelser inom datavetenskap som studenter uppfattar som särskilt utmärkande. Som teoretisk inramning använde gruppen en relativt oprövad teori om lärande och undervisning som framför en hypotes om förekomsten av så kallade tröskelbegrepp inom olika discipliner. Dessa karakteriseras av att de kan upplevas som problematiska, transformerande, oåterkalleliga, integrerande för den som ska lära sig och ibland markerar de ut ämnets gränser gentemot andra discipliner. Med denna definition som verktyg söktes empiriska belägg för existensen av sådana begrepp inom ämnet datavetenskap. Huvudsakligen har kvalitativa metoder använts och data samlades in genom enkäter, informella och semistrukturerade intervjuer, begreppsdiagram, samt skrivna berättelser om studenters erfarenheter av lärande inom datavetenskap.

Det andra temat knyter nära an till studierna om tröskelbegrepp, men här har istället ett fenomenografiskt angreppssätt använts för att försöka förstå *variationen* i hur studenter uppfattar begrepp och andra företeelser. Fenomenografi är en kvalitativ forskningsansats som vill förstå på vilka kvalitativt skilda sätt ett fenomen beskrivs av en grupp av individer, och syftet är bland annat att kunna använda resultaten i undervisningssammanhang. Till skillnad från det första temat har den här forskningen inriktat sig på att undersöka på vilka olika sätt som studenter ser på olika yrkesmässigt relaterade fenomen för att bilda en uppfattning om hur förberedda studenterna är för yrkeslivets krav. De begrepp som valdes ut relaterar till utveckling och underhåll av programvara, mjukvarusystem, objektorientering, klassdiagram, samt Java-interface. Flera av dessa begrepp tillhör de områden där den tidigare forskningen hade påvisat kunskapsluckor hos nyanställda programmerare. För att samla in data om studenters olika uppfattningar har framförallt semistrukturerade forskningsintervjuer använts. I en av studierna utvecklades denna datainsamlingsmetod genom att använda ett rollspel som skulle påminna deltagarna om en professionell situation. Genom att spela rollen av en nyanställd programmerare fick informanterna i uppdrag att lösa ett problem som innebar att de skulle vidareutveckla en omfattande programvara.

Genom att använda idén om tröskelbegrepp har studierna lyckats påvisa att det finns begrepp inom datavetenskap som ur studenters perspektiv är särskilt problematiska och betydelsefulla. Efter flera olika analyser av vad studenter själva berättar har ett antal sådana begrepp kunnat identifieras. Objektorientering och objektorienterad programmering var ofta återkommande begrepp. I en av undersökningarna studerades i vilken mån studenter kunde identifiera och integrera de underbegrepp som ingår i dessa övergripande områden det visade sig att en helhetsförståelse saknades. Ur data från studenter i slutet av utbildningen kunde det däremot påvisas att många av studenterna faktiskt förändras mot att börja tänka och agera mer som datavetare. Ett sätt att använda resultaten är att ta mer hänsyn till vilka begrepp som är särskilt viktiga ur studentens perspektiv. Exempelvis kan resultaten användas till att formulera lärandemål och i samband med att lärarresurser och undervisningstid ska fördelas.

De fenomenografiska resultaten visar att studenterna erfor de fenomen som diskuterades i intervjuerna på ett antal kvalitativt distinkta sätt. Bland de kollektiva uppfattningar som framkom framträdde en variation, från ytliga synsätt som ofta hade en konkret karaktär, till mer avancerade förståelser som avspeglar både ett djup och ett helhetsperspektiv vilket även inbegriper yrkesmässiga aspekter. Resultaten kan användas av lärare som vill förstå hur det stoff som de lär ut kan uppfattas av sina studenter vilket leder till en kritisk reflektion över det egna sättet att se. De här insikterna kan också utnyttjas av lärare som vill hjälpa sina studenter att nå de djupare förståelsenivåerna. Jag har visat exempel på hur man kan åstadkomma detta genom att använda variationsteori, som genom att variera olika aspekter av ett fenomen hjälper den lärande att varsebli detta på nya, mer avancerade sätt. I det sammanhanget är det viktigt att överväga de professionella aspekterna av begreppen för att hjälpa studenterna att bli bättre förberedda inför sin framtida roll som professionella utvecklare av programvara.

Acknowledgements

My PhD project has been ongoing for several years and many people have become involved in my work in different ways. There are some people who I would like to acknowledge in particular.

First, let me mention my supervisors Michael Thuné, Shirley Booth and Roy Nilsson. Michael has been my main mentor from the very beginning and has taken an interest in my research with indefatigable energy and great patience. Shirley was the one who introduced me to phenomenography and her expertise in the field was invaluable. She supervised me during the early years as a PhD student. Roy has been my local mentor at University of Gävle. He has coached me since 2005 and his expertise in didactics and his advice has been most valuable to me.

Anita Hussénius, head of department at University of Gävle, gave important support in my balancing act between research and teaching. She showed a genuine interest in my welfare when I occasionally had moments of lack of faith in my own ability. Some of my work colleagues have kindly supported me by taking over parts of my teaching so that I could focus on completing my thesis. It was especially Magnus Hjelmblom, Fredrik Bökman and Bengt Östberg who helped in that regard.

Anna Eckerdal has been my closest PhD student friend. She is almost like a big sister to me; always one step ahead and constantly creating new connections to people. She introduced me to the "Sweden Group". In this group we have worked together with Carol Zander, Jan Erik Moström, Kate Sanders, Lynda Thomas, Mark Ratcliffe and Robert McCartney. These friends have certainly been a great source of inspiration, and they have opened my eyes to the strength and joy in working with others in research, even across national borders. I sincerely hope that we will continue with the exciting research and that we will have many fun moments together in the future.

If Anna is my sister in research, then the persons involved in Uppsala Computing Education Research Group (UpCERG) has been my research family; a group that makes you feel at home.

My own family has always inspired me to keep the spirit going. They have been very understanding and flexible in situations such as the Friday evening on-line meetings that kept me away from the weekend dinner table.

The thesis and the papers are in English. For me, as a non-native English writer, it is sometimes hard to get it right. Dan and Sharon Lazenby, Douglas Howie and Alan Shima have made valuable contributions by proofreading the thesis. Kate Sanders, Carol Zander and Annica Gullberg helped me with proofreading Paper VI.

Finally, I would like to address the persons who accepted to be informants in experiments and interviews. Without the students' contributions, there would be no results at all.

Uppsala University and the Swedish Research Council have funded the greater part of my research. University of Gävle, Nordea and Sandvik AB have helped with funding to carry out my interviews and share experiences with other researchers at conferences both in Sweden and abroad.

Thank you all for your efforts, your kindness, and your patience!

Bibliography

[1] K. Ahlberg. *View-Turns: University students' Narratives of Qualitative Changes in Ways of Experiencing Meaning of situations during Educational placement*. PhD thesis, Acta Universitatis Gothoburgensis 206, University of Gothenburg, Sweden, 2004.

[2] G. S. Åkerlind. Variation and commonality in phenomenographic research methods. *Higher Education Research & Development*, 24(4):321–334, 2005.

[3] A. Begel and B. Simon. Novice Software Developers, All Over Again. In *ICER '08: Proceeding of the Fourth international Workshop on Computing Education Research*, pages 3–14, New York, NY, USA, 2008. ACM.

[4] M. Ben-Ari. Situated Learning in Computer Science Education. *Computer Science Education*, 14(2):85–100, 2004.

[5] A. Berglund, M. Daniels, and A. Pears. Qualitative Research Projects in Computing Education Research: An Overview. *Australian Computer Science Communications*, 28(5):25–34, 2006.

[6] J. Bonar and E. Soloway. Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers. *Human-Computer Interaction*, 1(2):133–161, 1985.

[7] S. Booth. *Learning to program: A phenomenographic perspective*. PhD thesis, Acta Universitatis Gothoburgensis 89, University of Gothenburg, Sweden, 1992.

[8] J. Boustedt. *Students working with a Large Software System: Experiences and Understandings*. Licentiate thesis, Department of Information Technology, Uppsala University, Sweden, 2007.

[9] J. Bowden. The nature of phenomenographic research. In J. Bowden and E. Walsh, editors, *Phenomenography*, Qualitative research methods series, pages 1–12. RMIT University Press, Melbourne, 1st edition, 2000.

[10] C. Bruce, L. Buckingham, J. Hynd, C. McMahon, M. Roggenkamp, and I. Stoodley. Ways of Experiencing the Act of Learning to Program: A Phenomenographic Study of Introductionary Programming Students at University. *Journal of Information Technology Education*, 3:143–160, 2004.

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

[12] N. Denzin and Y. S. Lincoln, editors. *Handbook of qualitative research.* Sage Publications, London, 1994.

[13] B. DuBoulay. Some difficulties of learning to program. In E. Soloway and J. C. Sphorer, editors, *Studying the novice programmer*, pages 283–300. Lawrence Erlbaum, New Jersey, 1989.

[14] A. Eckerdal. *Novice Students' Learning of Object-Oriented Programming*. Licentiate thesis, Department of Information Technology, Uppsala University, Sweden, 2006.

[15] A. Eckerdal. *Novice Programming Students' Learning of Concepts and Practise*. PhD thesis, Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 600, Uppsala University, Sweden, 2009.

[16] A. Eckerdal, R. McCartney, J. E. Moström, M. Ratcliffe, and C. Zander. Categorizing student software designs: Methods, results, and implications. *Computer Science Education*, 16(3):197–209, 2006.

[17] J. Edwards and K. Fraser. Concept maps as reflectors of conceptual understanding. *Research in Science Education*, 13(1):19–26, December 1983.

[18] S. Fincher and M. Petre. *Computer Science Education Research.* Taylor and Francis Group, London, 2004.

[19] C. Fisher. Social support and adjustment to work: A longitudinal study. *Journal of Management*, 11:39–53, 1985.

[20] M. T. Flanagan and J. Smith. From Playing to Understanding: The Transformative Potential of Discourse Versus Syntax in Learning to Program. In R. Land, J. H. F. Meyer, and J. Smith, editors, *Threshold Concepts Within the Disciplines*, chapter 7, pages 91–103. Sense Publishers, Rotterdam, 2008.

[21] R. E. Gunderman. A glimpse into a program maintenance. In G. Parikh, editor, *Techniques of program and system maintenance*. QED Information Sciences, Inc, Wellesley, MA, 1988.

[22] S. Hadjerrouit. A constructivist approach to object-oriented design and programming. *ACM SIGCSE Bulletin*, 30(3):171–174, 1999.

[23] S. Hadjerrouit. Constructivism as Guiding Philosophy for Software Engineering Education. *ACM SIGCSE Bulletin*, 37(4):45–49, 2005.

[24] C. Holmboe, L. McIver, and G. Carlisle. Research agenda for Computer Science Education. In G. Kadoda, editor, *Proceedings of the 13th Workshop of the Psychology of Programming Interest Group*, pages 207–223, Bournemouth, UK, 2001.

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

[26] L. Jaccheri. Software quality and software process improvement course based on interaction with the local software industry. *Computer Applications in Engineering Education*, 9(4):265–272, 2001.

[27] L. Jaccheri and S. Morasca. On the importance of dialogue with industry about software engineering education. In J. B. Thompson and H. M. Edwards, editors, *SSEE '06: Proceedings of the 2006 international workshop on Summit on software engineering education*, pages 5–8, New York, NY, USA, 2006. ACM.

[28] M. Kajko-Mattsson, S. Forssander, G. Andersson, and U. Olsson. Developing CM3: Maintainers' Education and Training at ABB. *Computer Science Education*, 12(1–2):57–89, 2002.

[29] M. Kölling and D. J. Barnes. Enhancing Apprentice-Based Learning of Java. *ACM SIGCSE Bulletin*, 36(1):286–290, 2004.

[30] R. Land, G. Cousin, J. H. F. Meyer, and P. Davies. Threshold Concepts and Troublesome Knowledge (3): implications for course design and evaluation. In C. Rust, editor, *Improving Student Learning Diversity and Inclusivity*. Oxford Centre for Staff and Learning Development, Oxford, 2005.

[31] J. Lave and E. Wenger. *Situated Learning: Legitimate peripheral participation*. Cambridge University Press, Cambridge, 1991.

[32] T. C. Lethbridge. What knowledge is important to a software professional? *Computer*, 33(5):44–50, May 2000.

[33] Y. S. Lincoln and E. G. Guba. *Naturalistic inquiry*. Sage, Newbury Park, CA, 1985.

[34] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 119–150, New York, NY, USA, 2004. ACM.

[35] F. Marton. Phenomenography – a research approach to investigating different understandings of reality. *Journal of Thought*, 21(3):28–49, 1986.

[36] F. Marton. The structure of awareness. In J. Bowden and E. Walsh, editors, *Phenomenography*, Qualitative research methods series, pages 70–79. RMIT University Press, Melbourne, 1st edition, 2000.

[37] F. Marton and S. Booth. *Learning and Awareness*. Lawrence Erlbaum Associates, Inc, Mahwah, New Jersey, 1997.

[38] F. Marton and M. F. Pang. On Some Necessary Conditions of Learning. *The Journal of Learning Sciences*, 15(2):193–220, 2006.

[39] F. Marton and R. Säljö. On Qualitative Differences in Learning – 1: Outcome and Process. *British Journal of Educational Psychology*, 46:4–11, 1976.

[40] F. Marton and R. Säljö. On Qualitative Differences in Learning – 2: Outcome as a function of the learner's conception of the task. *British Journal of Educational Psychology*, 46:115–127, 1976.

[41] F. Marton and A. B. M. Tsui. *Classroom discourse and the Space of Learning*. Lawrence Erlbaum Associates, Inc, Mahwah, New Jersey, 2004.

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

[43] W. M. McCracken. Research on learning to design software. In S. Fincher and M. Petre, editors, *Computer Science Education Research*. Taylor and Francis Group, London, 2004.

[44] J. H. F. Meyer and R. Land. Threshold Concepts and Troublesome Knowledge (1): Linkages to ways of thinking and practising within the disciplines. In C. Rust, editor, *Improving Student Learning: Improving Student Learning Theory and Practice – Ten Years On*. Oxford Centre for Staff and Learning Development, Oxford, 2003.

[45] J. H. F. Meyer and R. Land. Threshold Concepts and Troublesome Knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. *Higher Education*, 49:373–388, 2005.

[46] J. Mulholland and J. Wallace. Strength, Sharing and Service: restorying and the legitimation of research texts. *British Educational Research Journal*, 29(1):5–23, 2003.

[47] J. D. Novak and A. J. Cañas. The Theory Underlying Concept Maps and How to Construct and Use Them. Technical Report IHMC CmapTools 2006-01 Rev 01-2008, Florida Institute for Human and Machine Cognition, 2008. URL: http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf.

[48] D. L. Parnas. Software Engineering programmes are not computer science programmes. *Annals of Software Engineering*, 6(1–4):19–37, 1998.

[49] A. Pears and M. Daniels. Structuring CSEd Research Studies: Connecting the Pieces. *ACM Conference on Innovation and Technology into Computer Science Education*, 35(3):149–153, 2003.

[50] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin*, 39(4):204–223, 2007.

[51] D. N. Perkins, C. Hancock, R. Hobbs, F. Martin, and R. Simmons. Conditions of learning in novice programmers. In S. E. and S. J. C., editors, *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale NJ, 1989.

[52] J. Randolph. *Computer Science Education Research at the Crossroads: A Methodological Review of Computer Science Education Research*. VDM Verlag Dr. Müller, Saarbrücken, Germany, 2008.

[53] A. Robins, J. Rountree, and N. Rountree. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2):137–172, 2003.

[54] D. P. Rowbottom. Demystifying Threshold Concepts. *Journal of Philosophy of Education*, 41(2):263–270, 2007.

[55] C. Schulte and M. Knobelsdorf. Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. In *ICER '07: Proceedings of the third international workshop on Computing education research*, pages 27–38, New York, NY, USA, 2007. ACM.

[56] D. Shinners-Kennedy. The Everydayness of Threshold Concepts: State as an Example from Computer Science. In R. Land, J. H. F. Meyer, and J. Smith, editors, *Threshold Concepts Within the Disciplines*, chapter 9, pages 119–128. Sense Publishers, Rotterdam, 2008.

[57] M. Thuné and A. Eckerdal. Variation Theory Applied to Students' Conceptions of Computer Programming. *Europeean Journal of Engineering Education*, 34(4):339–347, 2009.

[58] J. Turns, C. J. Atman, and R. Adams. Concept maps for engineering education: a cognitively motivated tool supporting varied assessment functions. *Education, IEEE Transactions on*, 43(2):164–173, May 2000.

[59] J. D. Tvedt, R. Tesoriero, and K. A. Gary. The Software Factory: An Undergraduate Computer Science Curriculum. *Computer Science Education*, 12(1–2):91–117, 2002.

[60] J. R. B. Vaughn. Teaching Industrial Practices in an Undergraduate Software Engineering Course. *Computer Science Education*, 11(1):21–32, 2001.

[61] E. Walsh. Phenomenographic analysis of interview transcripts. In J. Bowden and E. Walsh, editors, *Phenomenography*, Qualitative research methods series, pages 13–23. RMIT University Press, Melbourne, 1st edition, 2000.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 734

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-122304

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2010