Learning to Program, Learning to Teach Programming: Pre- and In-service Teachers' Experiences of an Object-oriented Language

by

IRENE GOVENDER

LEARNING TO PROGRAM, LEARNING TO TEACH PROGRAMMING: PRE- AND IN-SERVICE TEACHERS' EXPERIENCES OF AN OBJECT-ORIENTED LANGUAGE

by

IRENE GOVENDER

submitted in accordance with the requirements for the degree of

DOCTOR OF PHILOSPHY IN MATHEMATICS, SCIENCE AND TECHNOLOGY EDUCATION

in the subject

COMPUTER SCIENCE EDUCATION

at the

UNIVERSITY OF SOUTH AFRICA

PROMOTER: PROF D J GRAYSON

JOINT PROMOTER: PROF L M VENTER

NOVEMBER 2006

To my late mum, Rosy

and

my dad, Sigamoney Kallian,

who passed away in the final stages of this work

ABSTRACT

The quest for a better way to learn and teach programming, in particular object-oriented programming, is a challenge that continues to intrigue computer science educators. Even after decades of research in learning to program, educators still search for the optimal instructional approach that will solve the ‘learning to program effectively’ problem among introductory programming students.

The aim of this study was to gain insight into, and to suggest possible explanations for, the “qualitatively different ways” in which students experience learning to program using an object-oriented programming language, and to recommend teaching and learning strategies as a result of the outcomes of the research. In order to achieve these aims, a combination of phenomenographic research methods and elements of activity theory have been employed to gain an in depth understanding of pre- and in-service teachers’ learning experiences. The categories of description for the phenomenon, learning to program and the influence of the learning context have been analysed and described in detail.

It is argued that understanding learning to program using Java, in order to teach programming involves more than understanding learning to program as it is normally taught in university programming courses. In addition to object-oriented concepts such as message passing, inheritance, polymorphism, delegation and overriding, it entails understanding how learning to program is reflected in the goals of instruction and in different instructional practices. Knowledge of learning to program must also be linked to knowledge of students' thinking, so that teachers have conceptions of typical trajectories of student learning, and can use this knowledge to recognize landmarks of understanding in individuals.

The findings suggest relationships among students’ affective appraisals of the value of learning to program, their conceptions of learning to program, their approaches to learning it, their evaluations of their performance in tests and examinations and outcomes of their actions. The relationships emerged from student descriptions of their actions and the way in which different aspects of their learning and outcomes related to one another were qualitatively described and in some cases, quantified. In particular, the tensions between prior programming knowledge of a procedural language and current learning of an object-oriented language have emerged in the study. This has implications for teaching, as this study was set against the backdrop of the change in programming language in high schools, from a procedural to an object-oriented language.

ACKNOWLEDGEMENTS

My sincerest thanks and appreciation go to:

- My promoter Prof Diane Grayson for her patience and guidance in leading me through this project. Her encouragement, support and keen sense of criticism have enabled me to see this thesis through to its completion. She encouraged me to think more deeply and creatively.
- Prof John Barrow for his valuable insights during the initial stages of the study and his willingness to help even when he was in the process of obtaining medical boarding.
- My co-supervisor Prof Venter for his insightful, clear comments and just for agreeing to co-supervise my work during the last stages of the study amidst his busy schedule.
- My late parents for placing a high premium on academic achievement and their encouragement throughout my life.
- My husband Desmond (Des), and sons, Stanton and Joash for all their love, encouragement, moral support and understanding when I was not able to be with them.

CONTENTS

FIGURES

TABLES

GLOSSARY OF TERMINOLOGY

Throughout the thesis several terms are used with a particular meaning in mind; these meanings need to be explicated. The order in which the terms are described is logical, rather than alphabetical.

In-service: Qualified teachers who teach at the secondary school level and were studying (concurrently) the new programming language.

Pre-service: Students who were studying at the university to become professionally qualified as teachers, and who chose to major in computer science education

Phenomenography: A technique used to study variation, specifically variation between qualitatively different ways in which people see, experience, and understand the same phenomena.

Experience: Knowledge resulting from actual interaction with facts and events; this knowledge is reflected in statements about the world, actions and artifacts produced. Hence the experience is a description of the internal relationship between a person and phenenomena.

Ways of experiencing: How a phenomenon is understood, conceptualized, thought about or discerned by people.

Categories of description: A set of ways in which a phenomenon is experienced at a collective level; the individual characteristics and contextual factors are stripped away.

Outcome space: The union of a set of categories of description, forming an abstract space in which individuals move—more or less freely—back and forth.

Programming: The act of writing a computer program using a programming language to solve a problem.

Programming language: A specialized language for writing programs.

Procedural programming: An approach based upon the concept of procedure (known as subroutine) calls. Procedures simply comprise a series of computational steps to be carried out. The way to complete a task is to find a procedure that fulfils the requirements of the task and then invoke the procedure. If no such procedure exists, the programmer has to write one.

Objected-oriented programming (OOP): An approach that focuses design on the data (=objects) and on the interfaces to it. The way to complete a task is to find, or create, an object of a class whose behaviour includes carrying out the task and sending the object a message. If no such class exists, the programmer has to write one.

Relevance structure: Relates to what is called for to make sense of things, and to the criteria by which some parts of the phenomenon under study are seen as more or less relevant.

Activity theory: The elements of activity theory, as they are employed in this study, serve as a framework for describing, analyzing and explaining human activities (such as learning) as integrated parts of an environment. Within an activity theoretical approach, learning can be understood as being integrated in a larger system that considers the socially-based nature of human activity.

Enrolment: Pertains to the variety of factors forming part of establishing an “identity”, that of being (considering oneself) a programmer.

1 INTRODUCTION

The quest for a better way to learn and teach programming, in particular object-oriented programming, is a challenge that continues to intrigue computer science educators. Even after decades of research in learning, educators still search for the optimal instructional approach that will solve the 'learning to program effectively' problem. Hence it is imperative to understand what makes learning to program difficult for most introductory students, and how best we, as educators, can alleviate this difficulty. Since learning and teaching could be regarded as two sides of the same coin, knowledge of students' thinking and various conceptions of learning to program can inform teachers' instructional approaches. The study is, therefore based on empirical evidence to answer the question: 'how do pre-service and in-service teachers in educational environments experience learning to program?'

1.1 Background to the Study

A new curriculum for computer studies, which was renamed information technology, was approved as one of the national subjects in the National Curriculum Statement (NCS) (Department of Education, 2003). For the past decade most schools in South Africa, in particular those in the province of Kwa-Zulu Natal (KZN), have taught computer studies, which included a procedural programming language, namely Pascal. In the new curriculum, which was implemented in 2006, a new language is advocated, namely Java or Delphi. The language being advocated is not just another procedural language, but an object-oriented language. The change of programming language has implications for

teacher-trainees (pre-service teachers) and even more so for in-service teachers. Most in-service teachers learnt, and were trained, to teach a procedural language, such as Pascal.

By the year 2006 all computer studies (information technology) teachers were expected to be proficient in the object-oriented language in order to meet the curriculum needs of 2006 and beyond. In order to meet this goal, these in-service teachers enrolled for a course in object-oriented programming, using Java in 2004. The proposed new curriculum also has implications for the undergraduate pre-service teachers' computer science curriculum.

In the past, programming performance amongst students has been a matter for concern. This was reflected in the senior certificate examinations for computer studies during the period 1996 to 2002. The examining panel for computer studies, of which I was a part from 1996 to 2002, was able to obtain an insider's perspective of students' problem areas in the examinations. The computer studies examination consisted of two papers; theory and a practical. The sum of the marks obtained in each paper resulted in the final mark obtained by each student. These final marks were generally high and were considered a fair indication of the students' performances. However, the high achievements in the final result were often not reflective of the actual performance in the practical aspect (programming) of the examination. Students scored higher in the theory paper, which overshadowed the generally poor performance in the programming aspect of the exam. Therefore, misperceptions of students' programming ability still exist.

Whilst research in other fields of discipline education has been well established, computer science education research, at least in South Africa has been slow to come to the fore. Using some of the methods and results of research from other disciplines can help to understand aspects of computer science education.

In physics education research there is a shift in focus in physics instruction (Redish and Steinberg, 1999) from considering "*what are we teaching and how can we deliver it*" to "*what do students learn and how do we make sense of what they do*"(my italics). This shift in emphasis paved the way to revealing the insight that in order to address students'

learning difficulties in physics, emphasis should be placed on students' experiences of physics learning. Similarly, the insight gained from students' experiences of learning to program should help to address their difficulties in learning to program. Knowledge of "how students respond to teaching, how they tackle the everyday demands of learning and studying, or what kinds of difficulties…they encounter" (Hounsell, 1984, p. 189) can bring us closer to an understanding of what it means to learn in higher education. Investigations into different aspects of the teaching-learning process should, therefore, focus on both "what" and "how" students are expected to learn from lectures, tutorials or tests in the learning environment, and aim at maximizing what students learn in the long term.

1.2 The significance of the study

The study is strategically positioned in time; South Africa's national curriculum (Department of Education, 2003) proposes significant shifts in the ways that teachers conceive and carry out their work. Teachers are asked to focus not only on what learners learn, but how they learn – the process of learning as well as the content of learning. The National Curriculum Statement (NCS) for information technology is being implemented in 2006 in high schools in South Africa. It is significant in that the experiences of pre- and in-service teachers of the transition from a procedural language to an object-oriented language can play a major part in our understanding of the difficulties and successes in learning the new language. The learning experience of both the pre-service and in-service teachers would help to determine the important pedagogical issues in programming in particular, using an object-oriented programming language.

This study is also significant because it contributes to developing the teaching and learning of computer programming, as well as to enhancing the professional development of educators. It would appear, from my examination of the current and completed research that pedagogical issues related to programming in teacher training and high schools, and teacher trainees in tertiary institutions, is an under-researched area in KZN. This study therefore, adds to the body of research on programming pedagogies. Apart from the understanding gained and the resultant pedagogical issues, the combination of methods

chosen for the study has not been widely used in computer science education. This combination yields some useful methodological findings.

1.3 Rationale of the study

There is general agreement in the literature that learning to program is not an easy task (Kölling, 1999, Jenkins, 2002). Having to program in a new style if one has prior programming experience creates tensions between learning to program and learning to teach programming. Teachers' perceptions and behaviour are formed by their own experiences, both their past experiences and current views. The phenomenon, learning to program using a new programming paradigm within the teaching environment in which many in-service teachers found themselves, provided the basis for the study. The Centre for the Improvement of Mathematics, Science and Technology Education (CIMSTE) at UNISA offered an introductory programming course for in-service teachers. Based on the discussions with staff members in CIMSTE and my own reflections on the literature, the following elements offered themselves as key areas of investigation:

- experience of learning objected-oriented programming, and how it seems to be different for the pre-service and in-service teachers;
- the impact, if any, that previous programming knowledge has on learning to program in an object-oriented language;
- pre- and in-service teachers perceptions of the relationship between learning to program and problem solving
- implications for teaching.

Some of these concerns are cited in Ventura and Ramamurthy (2004). In exploring these issues related to learning to program, and more specifically learning to program in an object-oriented language, it should enable the teachers and lecturers to be confronted with, and to understand better, what students *really* derive from programming lectures/classes. As Ramsden points out,

> a relational perspective does not look for elegant general laws of learning, but for guiding hypotheses about typical conceptions and approaches that will help teachers convey particular subject matter in certain educational circumstances. (Ramsden, 1988, p. 28)

It is important to emphasise the conceptual understanding of students, which requires teachers to become aware of the characteristic ways in which students construct their knowledge and of common student difficulties (Haberman, Lev & Langley, 2003).

In addition, a pilot study was conducted as a foreground to the investigation. This pilot study gave me some insight into the chief characteristic of programming, namely problem solving. Problem solving was not simply a "concept"; it was what programming students *did* in practice. But of course, problem solving was also how students learnt programming; or at least, how they were meant to learn programming. Wotz (as cited in Mitchell, 2001) claims that "programming, regardless of paradigm, is about solving problems with the computer". This study explores the relation between learning to program, the aim of computer programming, and its main instrument, problem solving.

As a point of departure from other studies into student problem solving with undergraduate students and other learners (Casey, 1997; Choi & Repman, 1993; Reed & Palumbo, 1991; Saj-Nicole & Soloway, 1986), phenomenographic methods were used to develop and answer the three research questions in this investigation. This research draws extensively upon students' experiences at a collective level in a way not done before in South Africa. This information will contribute to the understanding of the nature of learning to program using an object-oriented language which undergraduate pre-service and in-service teachers experience.

The research questions addressed in this study are listed below. In each case, the question refers to learning to program using an object-oriented language.

Research question 1: What are the qualitatively different ways in which pre-service and in-service teachers learn to program?

Research question 2: What are the qualitatively different ways in which these teachers go about solving introductory computer programming problems?

Research question 3: How does the learning context influence the approach adopted during learning to program?

In this study, a qualitative approach was adopted. Simultaneously, techniques and strategies from the quantitative tradition were 'borrowed' in the interest of developing a comparative analysis between pre-service and in-service teachers, and prior programming knowledge and performance assessments in examinations.

To summarize, the main aims of the research are to:

(i) gain insight into the "qualitatively different ways" in which pre-service and in-service teachers experience learning to program using an object-oriented programming language;

(ii) suggest possible explanations for the qualitatively different ways in which pre- and in-service teachers experience learning to program as a result of the outcomes of the research; and

(iii) recommend teaching and learning strategies which may be incorporated into the teacher training programming curriculum in order to overcome some of the conceptual hurdles that students encounter, especially with regard to object-oriented programming.

1.4 Outline of the study

The study consists of nine chapters. Chapter 1 discusses the background to the study, the rationale of the study, and the significance, and aim of, the study.

An overview of programming is given in chapter 2. The key differences between procedural programming and objected-oriented programming (OOP), differences that are essential to the understanding of the experience of learning to program, and the tensions

that exist between the two programming paradigms are highlighted. This chapter also reviews the literature with regard to learning and teaching programming. Some of the major approaches to programming, known student difficulties with aspects of programming and problem solving research in programming are reviewed.

Chapter 3 defines the framework for my investigation. The framework elaborates the theoretical perspective underpinning the study; that is, naturalistic inquiry leading to phenomenography. Adopting a naturalistic inquiry research framework implies focusing on naturally occurring events during the research. Using a phenomenographic perspective to explore students' experiences implies focusing not simply on *what* students learn but on *how* students learn. A further dimension of phenomenography that is important to this study is its emphasis on the variation of students' experience and the resultant outcome space. Some attention is also given to activity theory in order to understand the influence that learning contexts can have on learning itself.

Chapter 4 sets out the research methods used in the exploration of the three research questions. The rationale for the interview, journal writing and observation methods used, as well as the selection of research participants, is described. The results of the pilot study are also discussed.

Chapter 5 presents the analysis of the variation in the way in which students learn to program using the phenomenographic approach. The different categories of descriptions (or conceptions) of learning to program are derived, and the dimensions of the variation throughout the categories, are explored.

Chapter 6 presents the analysis of the variation in ways in which students program, i.e. solve computer programming problems. As a result of the analysis, different approaches to programming are presented.

Chapter 7 presents the results of research question 3. A description of how students use the different settings (within specific learning contexts) of learning to program and problem

solving to which they have been exposed, for example the lecture, the test, and previous knowledge is provided. The chapter then focuses on the different ways in which the students integrate the influence of the settings into their intentions and conceptions of learning to program. Teachers' perceptions and views, which add to the context, are also considered. Teachers are an important part of the organizational network surrounding learning. Their activities contribute to the shaping of the learner and the setting and, in turn, reflect this "arena" of action (Lave, 1988). Hence part of understanding students' learning to program relates to interpreting the perceptions and actions of their teachers.

Chapter 8 highlights important and converging aspects of the study and anomalies are identified. Relationships between different aspects, such as level of previous knowledge and performance in examinations, are also examined here, using quantitative analysis.

Finally chapter 9 is concerned with synthesizing the results pertaining to all three research questions. The findings are summarized and discussed. The implications for teaching programming and learning to program are considered. Some recommendations are made for further study and limitations are discussed.

2 PROGRAMMING: A REVIEW AND DISCUSSION

There has been considerable research into computer science education, in particular programming, in the last 20 years. This interest in the teaching and learning of programming is not only a part of the general growth spurt in computer education, but also a field of research in its own right.

Programming is a challenging task and programming courses are generally regarded as difficult by many introductory students (Buck & Stucki, 2001; Mahmoud, Dobosiewicz & Swayne, 2004). Programming courses often have high dropout rates or, if completion of the introductory courses is achieved, there is a strong possibility that many students are still not able to program. Most educators involved in teaching programming agree that many students struggle in this field.

> Results from a recent project by McCracken *et al*. (2001) are *compelling*, (my emphasis) because of the number of authors from differing educational institutions and cultures. The 10 authors teach introductory programming across 8 universities, in 5 countries. Each author tested his/her own students on a common set of programming tasks. The students performed much more poorly than the authors had expected. The students did not simply fail to complete the set task; most students did not even get close to solving the task. (Lister & Leaney, 2003)

To this end much topical literature concerns the teaching and learning of programming at different levels of education. Some key concerns or questions in the literature are:

- What are the properties of expert programmers?
- What resources and processes are involved in creating or understanding a program?

- Why do a large percentage of learners find programming hard to learn?
- What are the cognitive requirements of the task?

During the period between the 1970s and late 1980s there has been extensive literature related to these questions and regarding programming as a cognitive process (Bishop-Clark, 1995; Brooks, 1977; Dijkstra, 1989; du Boulay, 1989; Linn & Dalbey, 1989). More recent literature highlights studies of object-oriented programming (OOP) and its relationship to the procedural approach to programming.

Studies of programming can be divided into two main categories, those with a software engineering perspective, and those with a psychological/educational perspective (Robins, Rountree & Rountree, 2003). Studies with a software engineering perspective focus on experienced or professional programmers. These programmers typically work in teams and often develop software projects. However, my interest is in the initial development of individual programming skills. This does not mean that an understanding of basic software engineering principles will play no part in the study or be given no attention. However, in this review learning to program is addressed from an educational perspective.

As already stated in chapter 1, the national curriculum for the discipline of computer studies at school spells out a change in the programming language from one of procedural, to one of object-orientation. It is, therefore, necessary for an understanding of both in-service and pre-service teachers' experiences of learning to program to include some comparison and to elaborate on the different programming paradigms and related languages. The next section (2.1) provides an overview of programming in general and programming languages relevant to this study. Drawing from the different sources of literature, several trends and topics can be identified, as indicated in sections 2.2-2.7. Section 2.8 discusses the difficulty of programming by considering factors such as aptitude, learning style and motivation. In section 2.9 the issues relating to the teaching and the pedagogies of programming are reviewed. Three didactic models are reviewed and a fourth model is suggested. Section 2.10 discusses the kind of thinking that is characteristic of programming and Bloom's taxonomy is also given attention with regard to programming,

while section 2.11 reviews the role mental models play in the comprehension of programs. Section 2.12 concludes the chapter.

2.1 Programming review

A program is a set of instructions that a computer can execute to perform a task. This is a simple enough idea, but for the computer to process the instructions, they must be written in a form that the computer can use. Therefore, programs have to be written in programming languages. There are two basic aspects of programming: data and instructions. To work with data, one needs to understand variables and types of data (in the case of procedural programming), and objects encompassing attributes and actions (in the case of OOP); to work with instructions, one need to understand control structures and subroutines (in the case of procedural programming) and interacting objects and methods (in case of OOP). The rules that determine what is allowed and not allowed are called the syntax of the language. Syntax rules stipulate the basic vocabulary of the language, and how programs can be constructed using techniques like loops, branches, and subroutines (methods).

So, to be a successful programmer, one has to develop a detailed knowledge of the syntax of the programming language that one is using. However, syntax is only part of the story. It's not sufficient to write a program that will run: a program that will run *and* produce the correct result is needed! In short a semantically correct program is required. A semantically correct program is one that does what you want it to, i.e. the meaning has to be correct. The meaning of the program is referred to as its semantics.

For the procedural programmer, the basic building blocks of programs are variables, expressions, assignment statements, subroutine call statements and control structures. For the OO programmer, the basic building blocks are objects that consist of data and functionality. Stringing the relevant building blocks together in order to obtain a semantically correct program to perform a specific task, is what makes programming a challenge and what many consider to be difficult (Jenkins, 2002; Thomas, Ratcliffe &

Thomasson, 2004). A distinction between "programming in the small" and "programming in the large" is made in the literature. Eck (2004) refers to programming in the small (sometimes called coding) as that of filling in the details of a design, i.e. the step-by-step instructions for performing fairly small-scale tasks. The design of the overall structure of a program (complex programs) that makes use of many classes (in the case of OOP) and many procedures (in the case of procedural programming) is referred to as "programming in the large". One needs both experience and intuition to solve *even* the simple, small-scale tasks. To develop a complex program, an even greater effort is required.

For a long time, both teachers and students in high schools have been using a procedural language and procedural approach to programming. They have done this with some success. Java, an internet-savvy and OO language, has gained popularity in tertiary institutions and very recently has been introduced in high schools. The attractiveness of the internet has contributed in part to the change of language (object-oriented language). This change has incited a lot of discussion and debate with regard to implementation of the language. Some of the issues of debate in and amongst higher education institutions involve when and how object-oriented programming should be taught in the introductory programming courses (Zhu & Zhou, 2003; Kölling, 1999; Kölling & Rosenberg, 2001) and whether or not the object-oriented approach should be taught in a language-free scenario (Fincher, 1999). These concerns seem to make appropriate the need to highlight the difference between the two approaches to programming.

Comparison of procedural and object-oriented languages

There are many programming languages. Languages are categorized into procedural (or structured, also often referred to as imperative), logic, functional, and object-oriented languages. The popular languages traditionally taught in schools and universities were procedural languages, mainly Pascal[1] and FORTRAN. However, in recent years, object-oriented languages such as Java have become one of the influential programming languages in implementing the OO paradigm. *Object-oriented programming is*

[1] Other procedural languages (such as Basic, C, PL1) were also used particularly before Pascal was developed.

characterized by programming with objects, messages, and inheritance within hierarchies of classes. The terms need to be clarified. An *object* is a program construction that has data (that is, information) associated with it and can perform certain actions. When the program is run, the objects interact with one another in order to accomplish whatever the program is designed to do. The actions performed by objects are called *methods*. A *class* is a type or kind of object. All objects in the same class have the same kinds of data and the same methods associated with that class (Savitch, 2001). Object-oriented programming shifts the emphasis from data, as passive elements defined by relations or acted on by functions and procedures, to active elements interacting with their environment. In contrast to procedural programming, the emphasis shifts from describing control flow to describing interacting objects.

Programs must be designed. Few people can simply sit down at the computer and compose a program of any complexity. The discipline called software engineering is concerned mainly with the construction of correct, working, well-written programs.

The traditional primary software engineering design methodology was structured programming. The term *structured programming* was coined to describe a style of programming that emphasizes hierarchical program structures, in which each command has one entry point and one exit point. The goal[1] of structured programming is to provide control structures that make it easier to reason about procedural programs. The structured programming approach to program design was based on the following principle: divide and conquer. More specifically, to solve a large problem, break the problem into several pieces and work on each piece separately; each piece is then treated as a problem which can itself be broken down into smaller problems. Eventually you will work your way down to problems that can be solved directly, without further decomposition. Next, the sub-solutions must be reassembled to generate the solution to the problem. This step probably involves creating an algorithm[2] that controls the sequence of events. This approach is called top-down programming (Eck, 2004; Morgan, 1991).

[1] The "GO TO" statement, which goes against structured programming principle, still exists in procedural languages. Hence this goal was not achieved in its full generality.

[2] An algorithm is an unambiguous, step-by-step procedure that terminates after a finite number of steps; an

However, the OOP[1] approach to software design is to start by identifying the objects involved in a problem and their interaction. The program that results is a collection of objects, each with its own data (instance variables) and its own set of responsibilities (implemented through its methods, which are similar to *subroutines* in procedural programming[2]), that communicate through messages. In OOP, an object is a self-contained entity that has an internal state (the data it contains) and that can respond to messages (calls to its methods) (Eck, 2004).

Eck (2004) maintains that one of the primary problems with strict, top-down programming is that it makes it difficult to re-use work done for other projects. By starting with a particular problem and subdividing it into appropriate pieces, top-down programming tends to produce a design that is unique to that problem. It is not likely that you will be able to take a large chunk of programming from another program and fit it into your project, without extensive modification to the code. Another approach to design is to start "at the bottom," with problems that one already knows how to solve (and for which one might already have re-usable software components). From there, one can work upwards towards a solution to the overall problem. This approach is sometimes referred to as bottom-up design. Usually the approach to the traditional design involves a combination of top-down and bottom-up design. To a limited extent re-usable code is used and emphasized in the bottom-up design (Morgan, 1991).

Bergin and Winder (2000), among others, believe that OO is a paradigm, different from procedural programming, which requires a change in mental model (a paradigm shift) in the practitioners. It has emerged, from the literature reviewed, that programming for introductory students is challenging and can be difficult. What it may be pertinent to ask now is, will programming in the new paradigm be just as, or less, difficult as it is in the old paradigm, or does the shift in paradigm pose additional difficulties?

---

algorithm is not the same as a program.

[1] OOP refers to object-oriented programming

[2] A crucial difference between methods and subroutines is that methods are polymorphic.

The myth that “object-orientation and procedural concepts are mutually exclusive” is refuted by Lewis (2000). He argues that an object-oriented approach does not throw out the concepts that are admired in a procedural approach; rather it augments and strengthens them.

The key features of OOP

According to Barker (2005), object-oriented programming uses class and objects while following certain design principles, which are:

- *Encapsulation*, which is a process of describing a class or object by giving only enough information to allow a programmer to use the class or object.
- *Inheritance*, which is a way of organizing classes by grouping classes with properties in common so that their common properties need only, be defined once for all the classes. One class can inherit part or all of its data fields and behavior from another class. The class that does the inheriting is said to be a subclass of the class from which it inherits behaviour and data fields. Objects inherit data fields and methods (behaviour) from their ancestors. This is related to the problem of reusing software since class is reusable. For example, one might define a class for vehicles (refer to super class A as in figure 2.1) that has instance variables to record the vehicle’s number of wheels and the number of seats available (e.g. 1-seater, 2-seater etc). A second class may be defined for automobiles (refer to subclass B) so that the automobile class may inherit the instance variables and methods of the class for vehicles and have more features specific to the subclass B.

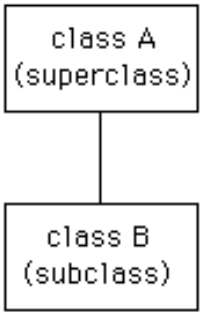


Figure 2.1 A representation of inheritance

- *Polymorphism*, in which objects of different subclasses can respond to the same message (one method name) in different ways.

A diagrammatic representation of the two approaches to programming is illustrated in the figure below.

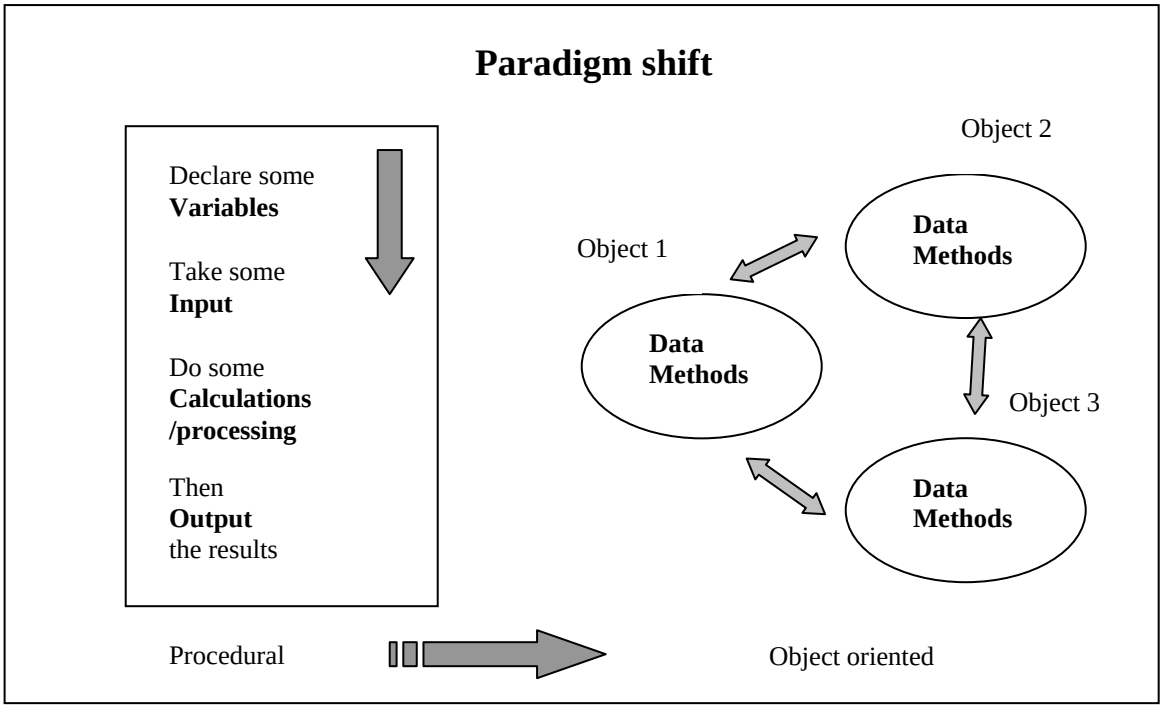


Figure 2.2 Procedural and Object-oriented Paradigm

A number of recent studies (Bergin & Reilly, 2005; Cantwell–Wilson & Shrock, 2001) explore issues relating to the OOP paradigm. The argument put forward for embracing the OO approach is twofold. In the first instance, it is argued that objects are natural features of problem domains and can be represented as entities in the programming domain. Secondly, the mapping between the domains is simple and should, therefore, support and facilitate OOP design. However, the literature reviewed shows that identifying objects is not an easy process for novices, and the mapping between domains is not straight forward. While the literature on expert programmers is supportive of the naturalness and ease of OO design, it also shows that expert OO programmers use both OO and procedural views of the programming domain, and switch between them as and when necessary (Détienne, 1990). However, this study is particularly concerned with novice OO programmers.[1]

In this section, the words *problem* and *solve* have been mentioned a number of times in order to explain the different approaches to programming. In the interviews with students it seemed to be a typical perception that programming is about solving problems. It is, therefore, crucial to discuss and review the literature on problem solving within the context of programming.

2.2 Problem Solving and Programming

In learning a programming language, it is almost obligatory that the goal is to solve problems using the programming language. As Mitchell (2001) points out "the programming faculty owes their first allegiance to teaching students to solve problems with algorithms". This study is not concerned with the discipline of Information Systems (IS), in which most students are often taught programming so that they can understand issues related to implementing a computer system, rather than for they themselves to be able to program. In short, students in the IS discipline, "instead spend time creatively, identifying business opportunities and problems and devising approaches and solutions" (ECU, 2002). Their goal and emphasis is different to that of mainstream programming. This study

[1] Note that novice OO programmers may have had experience in procedural programming and are, therefore, not necessarily the same as completely novice programmers.

highlights the issues concerning solving problems using the programming language and determines how students learn to program in an object-oriented language.

McCoy (1990) argues that learning to program using any programming tool requires skills, such as general strategy planning and logical thinking. Several studies (Palumbo, 1990a; Liao & Bright, 1991; Rucinski, 1991; Choi & Repman, 1993; Thomas & Sylvester, 1996) support a strong relationship between the processes of problem solving and computer programming. Thomas and Sylvester (1996) and Deek (1999) concur that programming involves problem solving which requires a series of steps that are similar to Polya's method for solving problems in mathematics (Polya, 1957). In a study done by Wells (1981), he highlights the processes of programming which involve the use of heuristics, sub goals, looking back techniques (re-evaluation), trial and error and regular patterns of analysis and synthesis. According to Schoenfeld (1985), these processes are inherent to mathematical problem solving. The solution to a problem is referred to as an algorithm. Algorithms are basic to all computer programming. Mathematical skills are also needed in the study of computer algorithms (Knuth, 1997). This is a possible reason why most universities admit students to computer science provided they have at least a grade 12 level mathematical background.

Schoenfeld (1985) suggests that "the ability to solve problems" was taken as an operational definition of understanding. One understands how to think mathematically when one is resourceful, flexible, and efficient in one's ability to deal with new problems in mathematics. His studies also indicate that typical instruction and testing provide little opportunity for students to demonstrate the breadth and depth of their misconceptions. This view is still held by many scholars. Based on conversations with other teachers and workshops held with regard to teaching, it would appear that generally the average student has little or no awareness of, or ability to use, mathematical heuristics. In most testing situations, students are asked to solve problems similar to those they have been trained to solve. As a result, the context keeps them in the right arena, even when they are unable to solve the problems. There is anecdotal evidence to suggest that many students may have passed a programming course, but are unable to program at the conclusion of their

introductory courses (McCracken *et al.*, 2001; Lister & Leaney, 2003 and Thomas, Ratcliffe & Thomasson, 2004).

Computer programming with procedural and object-oriented languages, such as Pascal and Java respectively, can present a steep learning curve for many students. This can deter students from taking an active role in modeling their cognitive ability. Linn (1985) identifies a chain of learning (cognitive) accomplishments that should develop during the course of programming instruction. She identifies three learning accomplishments, namely:

1. *Learning the language features.* The initial learning involves the features and syntax of the language. The features are non-decomposable elements such as "if…else" or "switch" or "try…except" in Java. Students typically demonstrate their knowledge of the language features by understanding already coded programs. For example, they might change a program using a different language construct to achieve the same result. A typical example would be to rewrite a segment of code that contains a "while ..." loop, using the "do…while" loop.
2. *Design Skills.* The next stage of their learning consists of design skills, or techniques for combining language features to form a program that solves a problem. Design skills may include re-useable code that has been written previously and procedural skills. The re-useable code fragments are like templates that perform complex functions, such as sorting names alphabetically, summing a set of numbers or finding the average of a set of marks. The ability to use templates developed by others to solve more complex problems is a feature of design skills. A large repertoire of templates enables the programmer to solve many problems without creating new code. Combining different features of the language and templates also forms part of the design skill.
3. *Problem solving skills.* The third stage in their learning is the development of problem solving skills. A program to be written is initiated in the form of a problem (learning task). The task demands some degree of interpretation on the student's part before a model, in the form of a program, is produced.

This chain of accomplishments is what could be characterized as a deep approach to learning in introductory programming (Robins *et al.*, 2003).
Programming involves more than one distinct process. Given the chain of accomplishments and the goals of deep learning, an observation that recurs both anecdotally and in the literature, is that the average student does not make much progress in an introductory programming course. Linn (1985) observes that few students get beyond the language features link of the chain.

The above chain of learning accomplishments was identified when procedural programming was the popular programming style. A pertinent question now is, is the above chain of cognitive accomplishments appropriate for the object-oriented paradigm? This study suggests, to some extent, how these accomplishments are met (see chapter 5 for analysis of students' experiences in learning to program).

There is general agreement in the literature that problem solving is one of the higher order thinking skills that needs to be developed in our students. According to Thomas and Sylvester (1996), programming is naturally problem solving and, therefore, programming entails higher order thinking skills. Hence, it is desirable that programming should be taught with the notion of promoting problem solving skills. Many introductory programming students have said "I just don't know how to start with this task…" To determine individuals' abilities to solve problems, researchers correlated the individuals' performance in problem solving, with how well they remembered the information and how they organized it in order to complete each of the steps leading to the solution. In this way, researchers identified "good" task analysis skills in problem solving (Good & Smith, 1987). Studies of "expert" programmers (in particular Guindon, 1990) and novice programmers (Winslow, 1996) have led to distinct characterizations of both.

2.3 Experts versus Novices

It is generally agreed (Winslow, 1996) that it takes a long time to turn a novice into an expert programmer. One cannot become an expert without experience. However, one can

have experience without becoming an expert. Some people just put in their time and never develop themselves. Dreyfus and Dreyfus (1986) proposed a breakdown of this continuum into five stages: novice, advanced beginner, competent, proficient and expert. In general most of the learners fall within the first four categories. From the perspective of educators, and for the purpose of this study, I am most interested in the question of how novices[1] learn to program, given the teachers' and students' background and the learning context. There are many studies of expert programmers (those who fall mainly in the competent, proficient and expert categories). In a survey of programming understanding of experts (von Mayrhauser & Vans, 1994) and of novices (Winslow, 1996), there are distinct characterization differences in the way experts and novices program. These characterizations are summarized in the table below.

Table 2.1 Characteristics of expert and novice programmers

Experts	Novices
• Have efficiently organized and specialized knowledge schemas • Organize their knowledge according to functional characteristics, such as the nature of the underlying algorithm • Use both general problem solving strategies (such as divide-and-conquer) and specialized strategies • Use specialized schemas and top-down, breadth-first approach efficiently to decompose and understand programs • Are flexible in their approach to program comprehension • Spend more time on planning and testing code • Have the tendency drastically to reformulate programs when existing programs look questionable (Linn & Dalbey, 1989).	• Limited to surface and superficially organized knowledge • Lack detailed mental models • Fail to apply relevant knowledge • Approach programming "line by line" rather than using meaningful program "chunks" or structures • Use general problem solving strategies (rather than problem specific or programming specific strategies) • Spend less time planning and testing code. • Tend to attempt small "local" fixes in the light of debugging

[1] For the purpose of this distinction between novice and expert programmers, I will refer to novices as encompassing the first two stages of the continuum.

Many of the characteristics of expert programmers are also characteristics of experts in general, as explored in other fields such as chess or mathematics (Robins *et al.*, 2003). It would be a desirable goal to have introductory students emulating characteristics of expert programmers. A key question may, therefore, be asked: *how can introductory students be taught so that they can emulate at least some characteristics of expert programmers?* The discussion in chapter 9 gives some idea of how this can be achieved.

2.4 Knowledge versus strategies

Davies (1993) distinguishes between programming knowledge (of a declarative nature, e.g., being able to explain how a "while…do" loop works) and programming strategies (the way knowledge is used and applied, e.g., using a "while …do" appropriately in a program). Knowledge about computers, programming language(s) and programming tools and resources are a necessary foundation for a programmer. Most introductory programming textbooks present knowledge about a particular language. This knowledge is elaborated on with examples and exercises. As Davies (1993), among various other authors, points out, knowledge is only part of the picture:

> Much of the literature concerned with understanding the nature of programming skill has focused explicitly on the declarative aspects of programmers' knowledge. This literature has sought to describe the nature of stereotypical programming knowledge structures and their organization. However, one major limitation of many of these knowledge-based theories is that they often fail to consider the way in which knowledge is used or applied. Another strand of literature is less well represented. This literature deals with the strategic aspects of programming skill and is directed towards an analysis of these strategies commonly employed by programmers in the generation and comprehension of programs. (Davies, 1993, p.237)

Widowski and Eyferth (1986) compared novice and expert[1] programmers as they worked to understand programs that were typically structured and those that were unusually structured. They found that experts tended to read typical programs in long, infrequent runs[2] and characterized this style as employing a top-down, conceptually-driven strategy. They also found that experts tended to read oddly structured programs in short frequent

[1] See table 2.1 for characteristics of novice and expert programmers
[2] Run was defined as a sequential pass over a section of code.

runs and they characterized this style as a bottom-up heuristic. Novices on the other hand, tended to read both typically and unusually (oddly) structured programs in the same way.

Davies advocates that research should not only characterize the strategies employed by different kinds of programmer, but also concentrate on why these strategies surface. In other words one should explore “the relationship between the development of structured representations of programming knowledge and the adoption of specific forms of strategy” (Davies, 1993, p. 238).

2.5 Comprehension versus generation

Another important trend in the literature is the distinction between studies that explore program comprehension (in which students are given the code of the program, and they have to explain or demonstrate their understanding of how it works), and those studies that focus on program generation (in which students have to create a part of, or a whole, program to perform a task or solve a problem).

Brooks (1977) proposed a model of program comprehension. The model is based on different knowledge domains. This model suggests that writing a program involves constructing mappings from the problem in a domain into the text of a program. In the cargo-routing problem (referred to in Brooks), the objects are cargoes that have destinations. To reach these destinations, there are means of transportation that carry these cargoes with cost and time constraints. Numbers must then be assigned to the cost and time elements and identifiers are assigned to cargoes and destinations. This results in a new knowledge domain. In this domain, the objects have become numbers. In order to use the numbers in the program, an algorithm must be selected. Hence another domain which results in mathematical objects, such as trees or matrices, on which to operate, results. Translation of the algorithm into a programming language creates yet another domain, with data structure implementations (using the procedural paradigm) and primitive operations of the language. Finally, the execution of the program results in yet another domain, one in

which the objects are the contents of memory locations and the operations are those of the hardware.

Brooks described program comprehension as a "top-down" and "hypothesis-driven" process. He suggested that experienced (expert) programmers form hypotheses based on high-level domain and programming knowledge. These hypotheses are verified by searching the program for special features, such as specific structures or functions. Students may vary with respect to their domain knowledge, programming knowledge, and comprehension strategies. Brooks claimed that this model is, therefore, able to account for the variation in comprehension performance, which may arise from factors such as the nature of the problem domain, variations in the program text, the effects of different comprehension tasks (such as modifying and debugging) and the effects of individual differences (e.g. different ability levels, background knowledge, prior computer competence).

Rist (1995) proposed a model of program generation. In this model, knowledge is represented using nodes in internal memory or external memory. A node encompasses an "action" that may range from a line of code, to chunks, such as loops, to routines. Nodes have something referred to as "ports"; these are :use, :make, :obey and :control. These ports allow the nodes to be linked with respect to control flow and data flow. A program is built by starting with a search cue (such as find, average, sort) and retrieving from memory any matching code. These nodes may contain cues, and these cues, within the newly linked node, are then expanded and linked in the same way. Rist's model (cited in Robins *et al.*, 2003) has been implemented in an artificial intelligence (AI) system called Zippy, that allows a user to specify the design strategies to be followed, the order in which the internal and external memories are searched and the content of each memory. In Rist's (1995) article he used the model to illustrate the generation of a Pascal program from English description. (The example given is: "Write a program that shows the number of sunny days (days in which no rain fell), the longest sunny spell, and the average rainfall for a month. Read in the amount of rainfall each day, measured in centimeters."[1])

[1] For a detailed analysis and trace through the model, refer to the article directly.

There are more studies of comprehension of programs than there are of generation of programs (Robins, *et al.*, 2003). This might be because comprehension studies are generally more narrowly focused and constrained, and it is, therefore, easier to interpret and describe the subjects'[1] behaviour. However, it is clear that program comprehension and program generation are related, because during generation the development, debugging (and, in the long term, maintenance) of code involves reviewing and understanding it. One expects these abilities to be highly correlated; however, there are more issues to consider before a direct correspondence can be made.

> Studies have shown that there is very little correspondence between the ability to write a program and the ability to read one. (Winslow, 1996, p. 21)

This is an issue that is still of concern in current studies. *This issue is addressed to a certain extent in Chapter 8.*

2.6 Misconceptions in general programming

It seems that misconceptions will always occur even if instructors read the relevant research related to pedagogical issues with regard to programming. Perceptions, interpretations and experiences of the teaching and learning to program differ. In fact, one can argue that repairing misconceptions is fundamental to learning. Issues that have been highlighted in studies completed some time ago continue to surface in the present. I will review some of the misconceptions or problematic areas in the literature.

Several studies that focused on novices' understanding, and use, of specific kinds of language feature are presented in Soloway and Spohrer (1989). In exploring the concept of a variable, Samurcay (1989) found that initialization is a complex cognitive operation. Reading in data (from user, i.e. external input) is better understood than assignment of a variable (see also du Boulay, 1989). Updating and testing variables were better understood than initialization. In a study of bugs in simple Pascal programs (which read some data and perform some processing) Spohrer, Soloway and Pope (1989) found that bugs associated

[1] Subject refers to the participants under study

with loops and conditionals were much more common than those associated with input, output, initialization, update, syntax/block structure, and overall planning. For example du Boulay (1989) found that, a common looping structure, the "for" loop, is problematic because students often fail to understand that the loop control variable is being updated automatically by the compiler. Arrays are another problematic area for novices. du Boulay noted that students confused array subscripts with the values stored.
Moving onto a more powerful construct, *recursion*, Kahney (1989) showed that users have a wide range of (mostly incorrect) approximate models of recursion. During the same time frame, Kessler and Anderson (1989) found that novices were more successful at writing recursive functions[*] after learning about iterative functions[*], but not vice versa.

Issues relating to flow of control were found to be more difficult than other kinds of processing, for example, finding the average of a list of numbers. When learning object-oriented programming, misconceptions with respect to object references were found to be one of the fundamental problems (Holland, Griffiths & Woodman, 1997). They, therefore, suggest that the" cleanest way to defuse this misconception is to teach reference as a first class concept…"

While these language-feature specific problems are significant, there are more general misconceptions that might be masked. "The notion of the system making sense of the program according to its own very rigid rules is a crucial idea for learners to grasp" (du Boulay, 1989, p. 387). In this respect, the notion of attributing human-like characteristics to the computer ("it was trying to…", "it thought you meant…") can be misleading and confusing.

Furthermore, it has emerged that problems with basic planning and design cause many other problems. For example, in a study of an introductory Pascal programming course, Spohrer and Soloway (1986) discuss two "common perceptions" of bugs:

[*] the corresponding term in OOP is method

> Our empirical study leads us to argue that (1) yes, a few bug types account for a large percentage of program bugs, and (2) no, misconceptions about language constructs do not seem to be as widespread or as troublesome as is generally believed. Rather, many bugs arise as a result of *plan composition problems* – difficulties in putting the pieces of the program together […] – and not as a result of *construct-based problems*, which are misconceptions about language constructs. (Spohrer & Soloway, 1986)

2.7 Programming: Issues

In each of the above trends of research, a problematic notion of programming is emerging. The key point made in section 2.4 is that "knowledge is part of the picture" and strategic programming skill is essential to complete the picture. In section 2.5, it is implied that reading and understanding a program is not necessarily an indication of one's ability to create (or write) a program. An important point made in section 2.6 is that misconceptions with regard to putting the pieces of code together are more widespread than the specific language feature misconceptions. These key issues tell us something very important: in creating or generating a program, problem solving is a taken for-granted skill and, as a result, the lack of problem solving ability may account for the poor achievement in programming and, therefore may maintain the notion that it is difficult.

Programs are usually written for a purpose – with respect to a particular task, problem or specification. Any attempt to write an appropriate program must surely be preceded by an understanding /mental model of the problem domain. The underlying claim is important in the literature reviewed – basic program planning (problem solving), rather than specific language features, is the main source of difficulty. Based on my experience, I concur with this underlying claim, although I consider each to be equally important. Problem solving is a complex process. One must deal with (1) whatever aspects of computer language problem solvers understand or misunderstand, and might bring to bear on a problem; (2) techniques they have (or lack) for making progress when experiencing difficulties; (3) the way in which they use, or fail to use, the information at their disposal; and (4) their computer world view, which determines the ways that the knowledge in the first three aspects is

used. Hence, problem solving needs to be considered carefully with regard to the learning of this skill.

Programming problems arise from a varied range of problem domains and understanding the problem domain is crucial (Adelson & Soloway, 1985). An understanding of the problem to be solved, and its solution space, is critical to problem solving. Most studies differentiate between a task (a goal with a known solution) and a problem (a goal with no familiar solution). What is a problem to a beginner may be a task to someone more advanced. Problem solving ability is then an important indicator of one's level of expertise. Problem solving is an important characteristic of experts in other disciplines as well. It is, therefore, relevant to ask: h*ow can educators develop or improve problem solving skills among introductory programming students?*

2.8 The Difficulty of Programming

Many studies have reiterated the complexity and difficulty of learning to program. The studies have indicated many factors to be contributory to this learning difficulty. Some of these factors will be discussed below.

The question of aptitude: It is often argued that the students who find programming difficult are simply and solely those who do not have a flair for programming. In other words, some students have no aptitude for programming, which is analogous to the cliché, "do not have a head for figures". The skills often cited are problem solving and mathematical ability. Programming aptitude testing has coexisted with the programming profession and education for decades. In order to show a stronger relationship between these tests and actual job performance of programmers, the progress of aptitude testing continued to add new features to the test battery (Winrow, 1999). There exist various programming aptitude tests (e.g. PAAT). Like all tests, the aptitude tests are not perfect. However, the basic conclusion of student performance studies is that aptitude, which is the potential to learn a task, "is not necessarily a good indicator of performance on a task per se" (Tukiainen & Mönkkönen, 2002). In the absence of any conclusive way of measuring aptitude for programming, and if it is possible that "aptitude" for programming does not

exist, it is then argued by (Jenkins, 2002) that one must turn to the cognitive view of learning.

Cognitive factors: Two cognitive factors may be considered as possibilities that might make learning to program difficult: learning style and motivation.

Learning style: Students have different preferences of learning styles. Some may prefer to learn in a solitary environment. Others may prefer a more dynamic learning environment which is conducive to having discussions with peers. Some students may demand a particular learning approach. However, without any guidance, the students tend to adopt the learning style they prefer or which has served them best in the past. The widely known classification of learning styles divides learning into "deep" and "surface" approaches (Biggs, 1999). Gaining an understanding of a topic is characteristic of a deep approach, while a surface approach is characteristic of memorizing for reproducing (see table 3.1 for characteristics of deep and surface learning). Programming is not a body of knowledge, it is a skill. It is argued that deep learning is vital for programming, providing understanding that can be applied in new problem areas. However, it could equally be argued that programming is essentially a "pattern matching" process, in which common problems are identified, and known working solutions are applied. This approach seems to lean more towards a form of surface learning. Programming does require both forms of learning. Surface learning can be useful for remembering the details of syntax, or other issues such as operator precedence, but deep learning (understanding) is necessary if true competence is to be developed.

Motivation: It has been shown that students approach computing degrees with a variety of motivations (Jenkins, 2001). Some may have a real interest in programming (intrinsic motivation); some may see their programming degree as a means to a high-paying career (extrinsic motivation) while others may simply be trying to please their family (social motivation). Motivation, in whatever form, does appear to be a factor that influences their learning (Jenkins, 2002).

The c*omplexity of programming*: An experienced programmer draws on many skills and much experience. Some of the obvious skills are problem solving ability, some idea of mathematics underlying the process, competent and effective use of the computer and correctly testing and debugging a program. Jenkins (2002) discusses the factors that

contribute to the difficulty of learning to program; programming consists of multiple skills, multiple processes, and being an educational novelty adds to the complexity. Dijkstra (1989) suggests that programming is what he terms “radical novelty” in which the tried and tested learning system no longer works. He argues that learning is, therefore, a slow and gradual process of transforming the “novel into the familiar”. Programming is “problem-solving intensive” (Perkins, Schwartz & Simmons, 1988) and “precision intensive.” The simple success that can be achieved by a novice programmer requires a high level of precision; a much higher level than most other academic subjects. A smallest imprecision (such as a misplaced or missing semi colon) or ambiguity can render a program worthless; the difference between success and failure. Learning to program is, indeed, difficult for many, but it should not be as difficult as it is presently if teaching approaches accommodate the difficulties. In order to accommodate these difficulties, teachers should understand and be aware of students’ experiences in learning. Knowledge of programming must also be linked to knowledge of students' thinking, so that teachers have conceptions of typical trajectories of student learning and can use this knowledge to recognize landmarks of understanding in individuals. To this end this research attempts to undertake what is needed: *an in depth study of students’ experiences of learning to program.*

2.9 Pedagogy of programming

While pedagogical research has provided theories of learning and teaching that are useful in informatics education, each subject or discipline requires specific teaching practices (Stodolsky, 1988), and these have to be based on the subject itself. Concerning the teaching of programming, studies with LOGO (Papert, 1980) provided knowledge of how children inductively learn the imperative kind of programming which LOGO supports. Although these studies are less relevant for adult students being introduced to programming, as in this research, I mention it because of the widespread interest it aroused in the past. In a study of conceptual structures of novice programmers (Petre *et al.*, 2004), there is evidence that the current methods of teaching Java and C++ have overwhelmed the students. In tertiary education, developing software support for teaching, e.g. specialized programming languages (Brusilovsky *et al*., 1997) has been an area of interest. Many teaching

innovations have been suggested, but not conclusively evaluated (Carbone & Kaasbøll, 1998).

The three basic strategies for an initial approach to teach programming are: imperative-first, functional-first, and objects-first. The first two approaches have been used for a fairly long period of time, whereas the third one appears to have attracted interest in the last few years. The functional-first approach initially places emphasis on functions leaving the presentation of state for later, whereas in the imperative-first approach the emphasis is first given to the state and then the concept of functions is presented.

According to the ACM curriculum report,

> Objects-first emphasizes the principles of object-oriented programming and design from the very beginning. The objects-first approach begins immediately with the notion of objects and inheritance and then goes on to introduce more traditional structures. (Chang *et al*., 2001, p. 30)

This means that, from the very beginning, both the state and functions must be presented. As the authors of the ACM curriculum report acknowledge, the objects-first strategy creates added difficulties for both the teaching and learning of programming. The typical methodology for teaching of programming began with small and simple programs, to be followed by more complex and larger-sized programs (in keeping with Bloom's (1971) taxonomy). This approach gave novice programmers time to assimilate and gradually to build up new knowledge relevant to the development of the programs. However, when using the objects-first approach, students are required to work with objects from the very beginning. This means that from the start they will have to be taught about objects, classes, methods, constructors, inheritance and, at the same time they will have to be taught the concepts of types, variables and values. They will also have to learn the syntax of the language which, for many novice programmers, comprises one of the biggest sources of difficulties.

The challenge of the objects-first approach has recently led to several studies that investigate this approach (see Cooper, Dann & Pausch, 2003). Some of the pedagogic models of teaching object-oriented programming will be reviewed in the next section.

2.9.1 Pedagogic Models

Kaasbøll (1998) discusses three pedagogic models for programming in the object-oriented style.

- **Semiotic ladder**

This model is based on the language-like features of computer tools: programming languages, modeling languages, formatting, formula, and search instructions. These tools are combined in general software products. The teaching and learning sequence starts out from syntax, and progresses to semantics and pragmatics of the language-like tools. This is based on the principle that syntactical knowledge is needed to express everything and, therefore, should precede the learning of meaning of the language constructs. When the meaning is acquired, the students can start to learn how to use the language for specific purposes, referred to as the pragmatics. Figure 2.3 below illustrates this model of teaching.

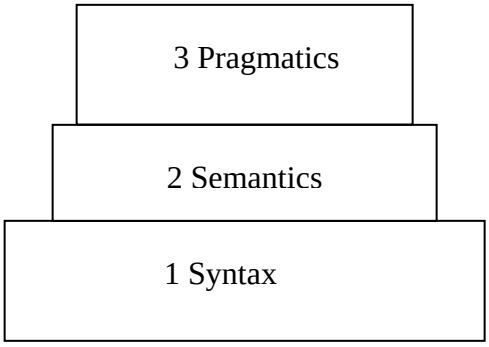


Figure 2.3 Knowledge of programming languages

- **Cognitive objectives taxonomy**

A teaching strategy that bears a resemblance to Bloom's (1971) taxonomy of cognitive objectives has been used by researchers Kirkerud and Reinfelds (cited in Kaasbøll, 1998). Figure 2.4 illustrates this taxonomy. The sequence of steps used in this strategy comprises, firstly, using an application program (running a program) and secondly, reading a program, followed by changing or modifying the program and then creating (generating) a program.

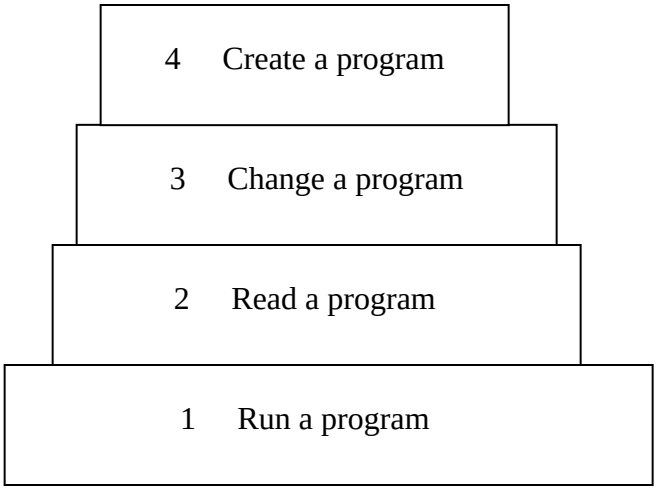


Figure 2.4 Cognitive objectives taxonomy

- **Problem Solving**

Rogalski and Samurcay (1990) are strong proponents of learning to program through the medium of problem solving. In this way, the students should widen their experience and repertoire of common practices, the basis for which is the knowledge structure of the field of programming. This approach stresses the input and outcome of the learning process in terms of knowledge and personal experience. This approach, therefore, seems to lean more towards a model of learning, rather than to a teaching model. The diagrammatic representation of this approach is reproduced from Rogalski and Samurcay (1990) in Figure 2.5.

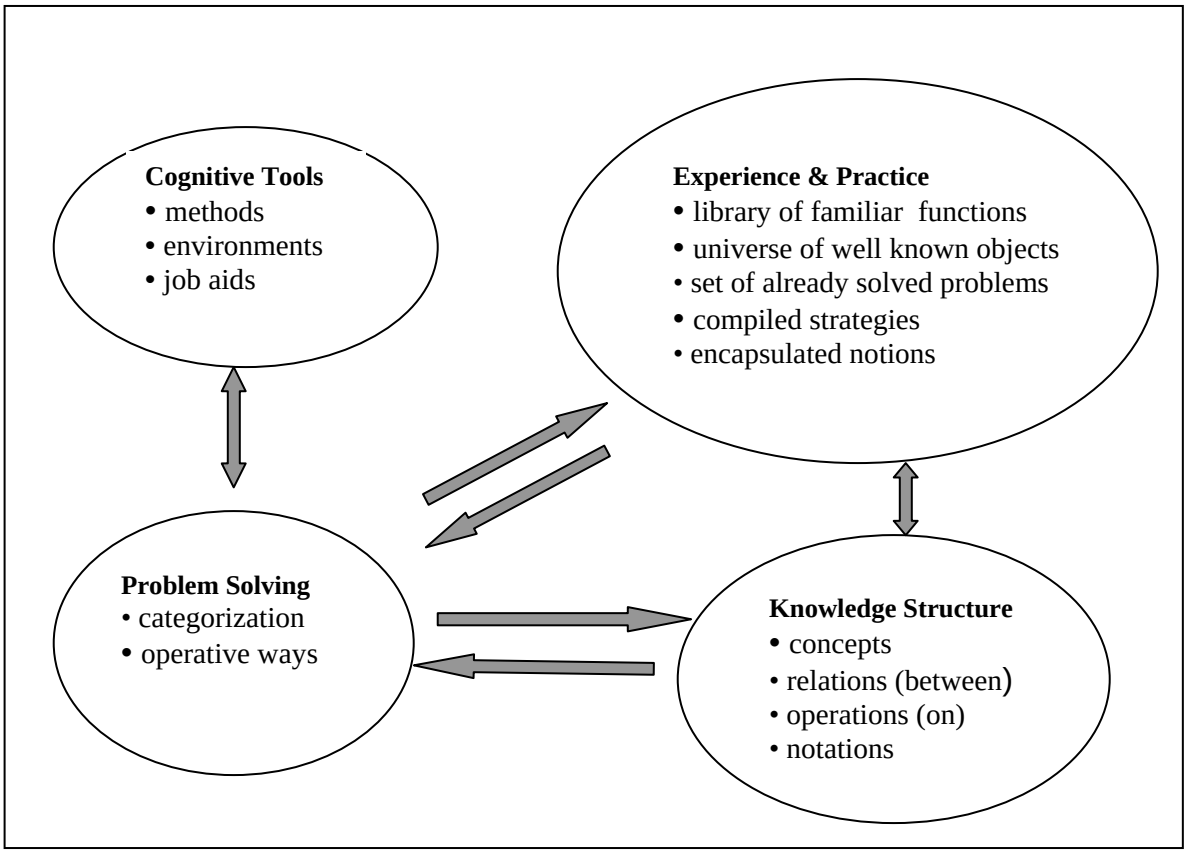


Figure 2.5 Acquisition of programming knowledge and skills (Rogalski and Samurcay, 1990)

Other researchers have complemented this model by using a problem-based learning approach, where language features are introduced only in the context of the students' solutions to specific problems. Taking this further, Green (1990) suggests that programming is best regarded as an exploratory process where programs are created "opportunistically and incrementally". Visser (as cited in Robins, *et al*., 2003) and Davies somewhat concur on a similar note:

> … emerging models of programming behaviour suggest an incremental problem-solving process where strategy is determined by localized problem-solving episodes and frequent re-evaluation. (Davies, 1993, p. 265)

The skill of solving problems grows out of practice, and patterns are directly useful at this level of learning. In the study done by Kaasbøll (1998), he found that none of the teaching

models had any significant advantages and, therefore, suggested a fourth model based on the software development process. A learning model also emerged from the study.

Putting the pieces of code together was reported, by Spohrer and Soloway (1986), to be one of the major hurdles in learning to program. This suggests that the step from understanding the programming concepts, to mastering the skill of coding, poses considerable difficulty. The learning model that has emerged from Kaasbøll's (1998) study is represented in Figure 2.6, which suggests a stepwise process of learning programming. According to Booth (1992), regarding programs as means of communication between *programmers* is a more advanced level of competence. Correspondingly, understanding programs as means for communication between *programmers and users* is another level of understanding.

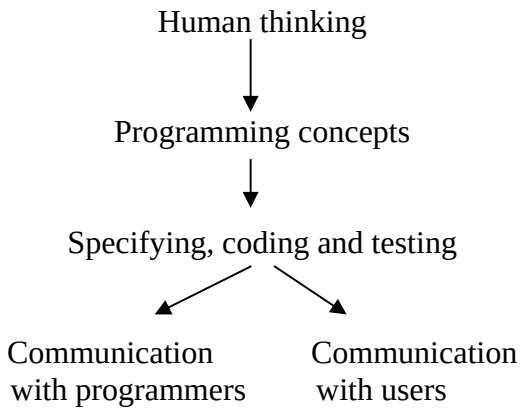


Figure 2.6 Learning programming

Many students, when starting to learn programming, make mistakes of the type which Pea (1986) calls the "superbug": the programming language is used as if the computer were a person. For example, a student in the introductory programming course wrote,

while > 0 do…

and could not understand why the code refused to compile. When the instructor asked the student "what is going to be greater than zero?" the student replied, "it". The students apply the human principles of thinking and acting to the computer. Therefore, it seems that the first step of learning to program, requires understanding the movement from human thinking to programming concepts as shown in figure 2.6 above. This includes understanding the way in which programs are executed. Putting pieces of code together, as

discussed in section 2.4, is one of the major obstacles in learning programming. Therefore, there seems to be a step from understanding the programming concepts to mastering the skill of coding. Leading from this, a fourth model of teaching is, proposed by Kaasbøll (1998) as shown in Figure 2.7.

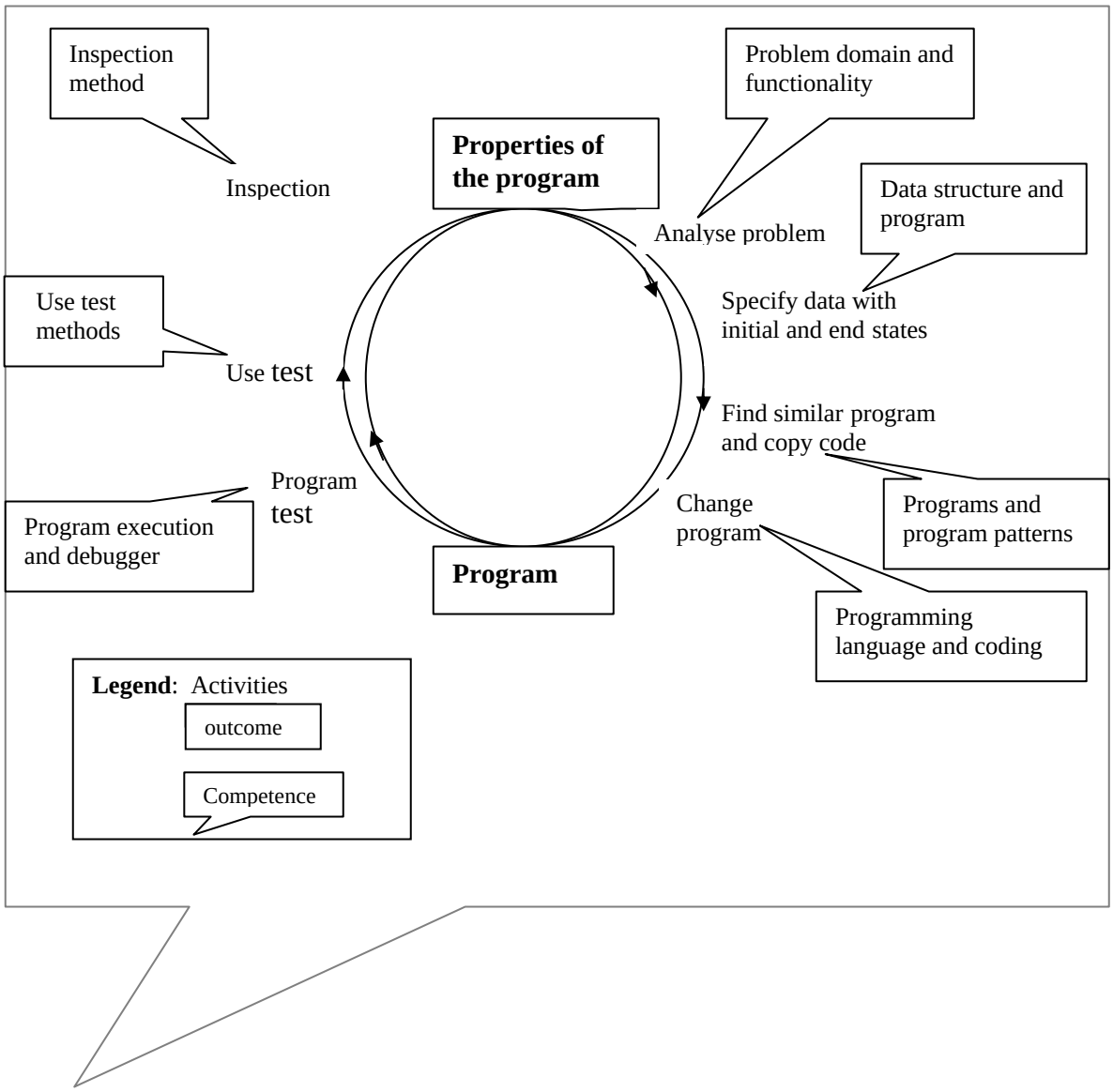


Figure 2.7 The iterative software development model

In this section, three teaching models were reviewed. Interviews of teachers indicated that, in general, they did not relate to these models (Kaasbøll, 1998). A fourth teaching model and a learning model emerged from his study. However, these models need to be validated

and developed with further studies. *With respect to this research, this might have important implications for teaching.*

2.9.2 Bloom's Taxonomy[1]

Bloom's (1971) taxonomy suggests that learning takes place in stages. The idea of the taxonomy is that educational objectives can be arranged in a hierarchy, from less to more complex. Table 2.2 (adapted from Porter & Calder, 2004) shows how Bloom's taxonomy may be related to some of the key tasks in programming.

Table 2.2 Bloom's categories in programming terms

Bloom's categories	**Learning to program**
Knowledge	Tools, constructs, syntax
Comprehension	Relating concepts
Application	Flow, semantics
Analysis	Understand the problem and assess other options
Synthesis	Create the solution
Evaluation	Evaluate/assess the programmed solution

Comprehension requires knowledge, application requires comprehension and so analysis requires the previous three categories of learning levels and so on. Learning to program is, therefore, considered difficult as it requires high level cognitive activities, such as analysis and synthesis, early in the learning process. The pedagogical implication is that students should be given an appreciation of the synthesis (putting together base concepts) right from the start. It has been practice, and seems to continue to be so, to give the students experience with small parts of programs, rather than expect them to write whole programs from scratch (Buck & Stucki, 2000). Porter and Calder (2004) suggest that the relevant aspects of programming that need to be understood for the application and analysis levels

[1]Benjamin Bloom created this taxonomy for categorizing level of abstraction of questions that commonly occur in educational settings. The taxonomy provides a useful structure in which to categorize test questions.

should be introduced in small increments. This approach would allow students to assimilate the aspects of relevance and, more importantly, to concentrate on the real skill of programming which is solving problems.

2.9.3 Problem solving for life long learning

From an Australian perspective, Slay (as cited in Slay, 2000) noted that many students who have difficulties across the first year of computer science or information systems as a whole do not know where to start with a task, irrespective of the subject area. Some effort has been made to incorporate training in problem solving skills and techniques into early computer science education to address this problem. This ranges from the use of Edward de Bono's ("Edward de Bono- lateral…", accessed February 2006) tools for lateral thinking, to the development of Polya's (1957) approach of understand, design and review for problem solving. They have offered courses in these techniques within, and parallel to, early programming subjects. However, there is no empirical evidence to suggest that general problem solving skills improve programming. It may be argued that programming contributes to a wider development of the individual. Mayer *et al*. (1989) claims that programming may improve, or in turn be improved by, prior experience with very closely related skills, such as translating word problems into equations.

There is, however, a process by which one can evaluate any new approach to a learning problem and that is to carry out an experiment based on the comparison of the old with the new. While the experiment, or the qualitative assessments of effectiveness might be easy to carry out, it is difficult to be clear-cut about the results, given the complexity of learning and programming. Research on evaluating teaching practice is important in improving practice, as is aptly captured in the quotation below.

> Given the current conditions, it is especially important to distinguish truth from assumption, to have practice that is well-founded. Evolving teaching practice is normal to good teaching, but evaluation reliant on anecdote is not good enough. Adding a research perspective allows educators to learn more from their practice. (Daniels & Berglund, 1998)

2.10 Thinking in programming

It is imperative to consider the thinking skills required in programming so that students can be helped in acquiring an easier way of understanding this novel aspect of learning. Students lack ways of thinking about information systems and their implementation. Possible thinking frameworks, such as systems theory, are seen as things to learn and not as frameworks for understanding, interpreting and developing information systems solutions. Thompson (2003) suggests the following thinking frameworks with regard to programming:

- Logic patterns
- Systems thinking patterns
- Higher-order processing patterns
- Design patterns, and
- Process or organizational patterns

Logic patterns: In this framework, the patterns are those of sequence, conditional, loop, and procedure (method) call. These form the core foundation for procedural oriented languages.
Systems thinking patterns: Programmers might more easily identify with input-process-output pattern as being the core foundation of their programming. In object-oriented programming this may not be explicitly stated. They may also identify with a hierarchy of systems. The concepts of encapsulation and cohesion rely on the idea of a hierarchy of systems.
Higher-order processing patterns: Emphasis on different programming paradigms used in implementing systems can further extend the students' ability to work with complex program structures and logic.
Design patterns: Design patterns can be seen as how-techniques. For example, on hearing the client talk of an invoice or order, an experienced analyst immediately has a pattern in mind of how that would be implemented. The creation, structure and behaviour of design patterns can provide the same thinking tools for the programmers both in writing their own codes and in interpreting the complexity of programming systems.

Process or organizational patterns: Knowing how to approach a programming task is essential in accepting the challenge of a complex and unfamiliar task. Process or organizational patterns provide ways of thinking about how to approach different challenges and tasks.

Bransford *et al.* (2000) emphasize the need to draw out, and work with, pre-existing understandings that their students bring with them. In the case of thinking patterns, this means helping students make explicit their current thinking patterns. The teaching process can then use this either to reinforce, or to test, those thinking patterns. The learners need to see the relevance of the thinking patterns in the context of the problem. In the discussion of thinking patterns, Bransford *et al.* (2000) explain that experts know which knowledge is appropriate for each problem situation. The students need to develop this same conditioning of knowledge so that they can make appropriate choices during problem solving.

In a separate study, Eckerdal and Berglund (2005) have discussed the issue of what it takes to learn 'programming thinking' and compare it with the discussion by Hazzan (2003). The discussion with regard to the 'process-object duality' was developed in mathematics education as a means of reducing abstraction. This idea refers back to the work of Piaget (1972). Hazzan explains this in terms of a journey from the "process conception" to the "object conception", which he refers to as 'process-object duality'.

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

In Hazzan's discussion of these theories, she concludes that "when a mathematical concept is learned, its conception as a process precedes – and is less abstract than – its conception as an object". It would, therefore, be an instinctive process when learning abstract concepts to start at the 'process conception'. This is conceivably a desirable development when learning computer science, including object-oriented programming. Hazzan refers to

‘canonical procedures’ as a means of reducing the abstraction level when dealing with concepts from different subjects. She explains this idea succinctly as follows:

> A *canonical* procedure is a procedure that is more or less automatically triggered by a given problem. This can happen either because the procedure is naturally suggested by the nature of the problem, or because prior training has firmly linked this kind of problem with this procedure. The availability of a canonical procedure enables students to obtain a solution without worrying too much about the mathematical properties of the concepts involved. It seems that this technical work gives students the assurance of following a well-known, step-by-step procedure, where each step has a clear outcome. In contrast, relying on abstract reasoning, for example by exploring properties of concepts or by relying on theorems, may be a shaky mental approach for the students. Using the process-object duality terminology we may say that solving a problem by relying on a canonical procedure is an expression of process conception of the concepts under discussion; solving a problem by analyzing the essence and properties of concepts is an expression of object conception of the concepts under discussion. (Hazzan, 2003, p. 108)

In learning object-oriented programming, understanding abstract concepts like object, class, inheritance and other object-oriented concepts is essential. Therefore, the ‘process-object duality’, as discussed above, is of immediate interest. Although programming is a skill, it also requires a deep understanding of the abstract concepts. There is a parallel between the analysis of these essential abstract concepts and the analysis and design phase in a programming problem. The analysis and design of the problem concepts are abstract skills. Just as in mathematics, where there are standard solutions to certain types of problems (canonical procedures) to learn and discover, these also exist in programming. Experienced programmers use these standard solutions to simplify and speed up the process. It is a desirable goal to help students to perceive such procedures (repertoire of templates). Eckerdal and Berglund (2005) compared their results with the discussion on ‘process-object duality’, developed in mathematics education and found that both point in the same direction. It is, therefore, of importance that students reach an understanding, that “learning to program is a way of thinking, which enables problem solving, and which is experienced as a “method” of thinking” (Eckerdal & Berglund, 2005). Students need to comprehend that ‘programming thinking’ is a special way of thinking.

In another, very recent study by Gary *et al.* (2005), which was published while I was completing this review, it has been found (predictably so) that more abstract concepts are those most likely to have little meaning for novice programmers. Furthermore, the study indicates that high performers are more likely, while low performers are less likely, to state

explicitly that a concept lacks meaning for them. The low performers would rather choose to try some pattern-matching fit into the conceptual structure. This might be the cause of many misconceptions about low performers' understanding. This has implications for the classroom teaching in trying to avoid these misconceptions. Furthermore, evidence was found that the meaningfulness of a concept is likely to be related to vocabulary used in the classroom. This suggests that students may assimilate abstract concepts into their conceptual structures quickly if one uses the terms more frequently in the classroom. For example, it was found that use of the "if-then-else" was always meaningful for 63% of the students while a related concept, "choice", was always meaningful for only 33% of the students in the study. When teaching programming, the concept 'if-then-else' is frequently used, while the concept 'choice' is less likely to be used with high frequency. In a similar fashion, while iteration may be taught without frequently saying "iteration", it is odd to teach "recursion" without saying the term. The rationale is that hearing the terms less frequently provides students with fewer opportunities consciously to consider and integrate them into their conceptual structures. The implication of this is that instructors may be able to help students build their conceptual organization by using key abstract terms frequently.

2.11 Mental models and program comprehension

I am (and so are other instructors as noted in the literature survey) very interested in the question of how students learn to program. This area is set in the context of cognitive psychology and other topics, such as knowledge representation, problem solving, working memory and mental models.

When writing programs, many different kinds of "mental model" are involved. This is apart from a model or knowledge of the language itself. As indicated earlier, programs are usually written for a purpose in order to solve a task. It is clear that an understanding, or mental model, of this problem domain must precede any effort to write an appropriate program.

2.12 Summary

Research on student learning of programming shows that students have difficulties with programming concepts and principles, and that they are often anxious about learning to program. The body of research on teaching and learning to program is vital and growing. While issues such as problem solving and programming, experts versus novices, knowledge versus strategies, comprehension versus generation and misconceptions in programming are discussed in this chapter, the difficulty of programming is acknowledged by considering factors such as aptitude, learning style and motivation. In the light of the above discussion, a number of teaching models have received attention. The kind of thinking that is characteristic of programming, and the role mental models play in the comprehension of programs, was discussed. Further, a concern with describing the above issues, while indicating an awareness of the problems facing programming education, does not offer a model for understanding the impasse — a first step towards alleviating it. *However one must ask what characterises progressively successful introductory programming students, in particular a group of pre- and in-service students?* My approach is to explore the issues underlying this impasse within a theoretical framework which relates students, their actions and the context (see Gordon, 1995c).

3 A FRAMEWORK FOR THE STUDY

This chapter describes the theoretical perspectives relevant to this study. The main theoretical perspectives drawn upon are naturalistic inquiry and phenomenography. Activity theory is used to complement phenomenography by analyzing an aspect of the system, namely, the learning context. Section 3.1 situates the study within the naturalistic inquiry paradigm, as naturalistic inquiry has informed the direction of the fieldwork and data analysis within the study.

In section 3.2 phenomenography is discussed in depth, with particular emphasis on its fundamental concern with "ways of experiencing" and "structure of awareness". Other learning perspectives are considered in section 3.3. The phenomenographic dichotomy of a "deep" versus "surface" approach to learning, as well as the implications of a relational approach to learning to program are considered in section 3.4.
The limitations of phenomenography are discussed in section 3.5. Section 3.6 briefly describes the system of activity theory and elements of activity theory which extend, and possibly complement, phenomenography that are appropriate for this study, are discussed.

3.1 Naturalistic Inquiry

From a naturalistic perspective, doing research means that the researcher is forced into the natural setting. The researcher cannot specify in advance what is important to control or even study in the setting. In general, naturalistic inquiry is an accepted (natural) way of

conducting qualitative research. Arguably, naturalistic inquiry, as espoused by Lincoln and Guba (1985), is a broader and more comprehensive statement of a paradigm, than Glaser and Strauss's (1967) ground-breaking work, which is rich in methodological detail. The principles of both grounded theory and naturalistic inquiry derive from the 'new paradigm' of research, which has been developed largely in the twentieth century as a reaction to the dominance of logical positivism. Reese (1980, p. 450, cited in Lincoln and Guba, 1985) defines positivism as "a family of philosophies characterized by an extremely positive evaluation of science and scientific method." Whereas positivism is characterized by a rational view of the world, which prizes objectivity and detachment from research problems as the greatest good, the new paradigm places intuition, interpretation and researcher involvement at the centre of the research enterprise. I will seek to examine the concrete features which set naturalistic inquiry apart as a distinct manifestation of the qualitative tradition of research for this study.

Key aspects of the naturalistic inquiry

There are a number of defining features of the naturalistic paradigm outlined in Lincoln and Guba (1985), the chief architects of the term "naturalistic inquiry". I shall examine aspects which are particularly relevant to this study, and go on to provide a diagrammatic summary of the research process, adapted from Lincoln and Guba (1985).

The first key aspect of naturalism is that the researcher carries out research in the context of the proposed reality (i.e. naturally). The implication of a contextual understanding of research is that these settings are not contrived, but natural. The natural setting provides clues by which the researcher is enabled to probe and pick at meanings, gestures, and words in a way that will enable the researcher to construct a reliable picture of the phenomenon of interest.

A second key dimension of naturalistic inquiry is the use of humans as instruments. The researcher is an integral part of the research. The responsiveness, adaptability, and interpretive skills are valued aspects of the researcher's capacity to operate as an instrument. The human as instrument is the key instrument in grasping and evaluating the

meaning of differential interaction, i.e. the discrete differences in communication that may occur in relation to the interviews of, or descriptions by, students.

Qualitative research methods are a third feature of the naturalistic research paradigm. These include, among others, interviewing, observation, the study of documents and records, case studies and journal writing. These methods expose, more directly, the nature of the issue (phenomenon) between the investigator and the respondent. Hence, a review of the extent to which the phenomenon is described in terms of the investigator's own position is made easier.

A fourth feature of the naturalistic paradigm is that of focus-determined boundaries. Based on the emergent focus of the inquiry, one is likely to set boundaries to the inquiry. Boundaries can only appropriately be set if intimate contextual knowledge exists. Knowledge includes factors that affect the learning and the actual learning material.

The fifth feature of the naturalistic inquiry which is applicable to this study is that of purposive sampling. The inquirer is more likely to avoid random or representative sampling in favour of purposive sampling. Its purpose is to maximize information, not facilitate generalizations. Its procedures depend on the particular ebb and flow of information as the study is carried out, rather than on a priori considerations. The criterion used to stop sampling is informational redundancy, not a statistical confidence level.

The sixth dimension of naturalistic inquiry which this study will employ is tacit knowledge. Heron (1981, cited in Lincoln & Guba, 1985) describes three kinds of knowledge which are employed in the process of inquiry. The first is propositional knowledge, i.e. knowledge of scientific laws and generalisations; the second is practical knowledge, (often referred to procedural knowledge) that is *how* it is done or rather the proficiency of doing it; and the third form of knowledge is tacit knowledge (that is intuitive), which could also be termed experiential knowledge. Every research inquiry employs some form of tacit knowledge in its conception and execution. The strength of the naturalistic paradigm lies in its willingness to acknowledge tacit knowledge as part of the process, rather than allowing it

to influence the process of research without being made explicit, or even recognised. The naturalistic inquirer seeks to make explicit his or her tacit knowledge at the outset, and this action adds rigor to, rather than detracting from, the credibility of a study.

Another key feature of this inquiry is the preference to use inductive data analysis, because data analysis is more likely to identify the mutually shaping influences that interact, and to conclude that human-as-instrument is more inclined towards methods that are augmentations of human activities, namely: observing, speaking, reading, listening and the like. The human will be able to take into account nonverbal cues as well as be able to interpret unintentional self-effacing measures. These are more suitable to qualitative measures, which in turn are readily done using the inductive data analysis.

The final dimension of naturalistic inquiry is that the design is emergent, rather than predetermined. For a number of reasons, including the human interactions and contextual richness of research of this nature, naturalistic inquiry is unpredictable. The indeterminacy and uncertainty of social and contextual factors prevents the research from being pre-planned and organized to fit a fixed design. More significantly, the data analysis runs alongside and concurrently with data collection, and informs the next steps in the process. But perhaps most significant in the unfolding or emergence of research design, is the nature of data analysis and its very purpose. Whereas conventional research methods are deductive, in that, general laws and theories are confirmed or refuted by data, naturalistic inquiry builds up hypotheses, research questions and theory from raw data. The design is, therefore, necessarily emerging as the researcher pieces together, revises, and clarifies questions which emerge out of data. The purpose of naturalistic inquiry is not to predict or generalize behaviour from tests, but to understand experience as it is 'lived' or 'experienced', in order to create conceptual maps to aid in the understanding of various social behaviours (and in this investigation learning styles and approaches) and relationships.

The following diagram, adapted from (Lincoln & Guba, 1985, p. 188) depicts the research process and summarizes the key aspects of the naturalistic paradigm relevant to this study.

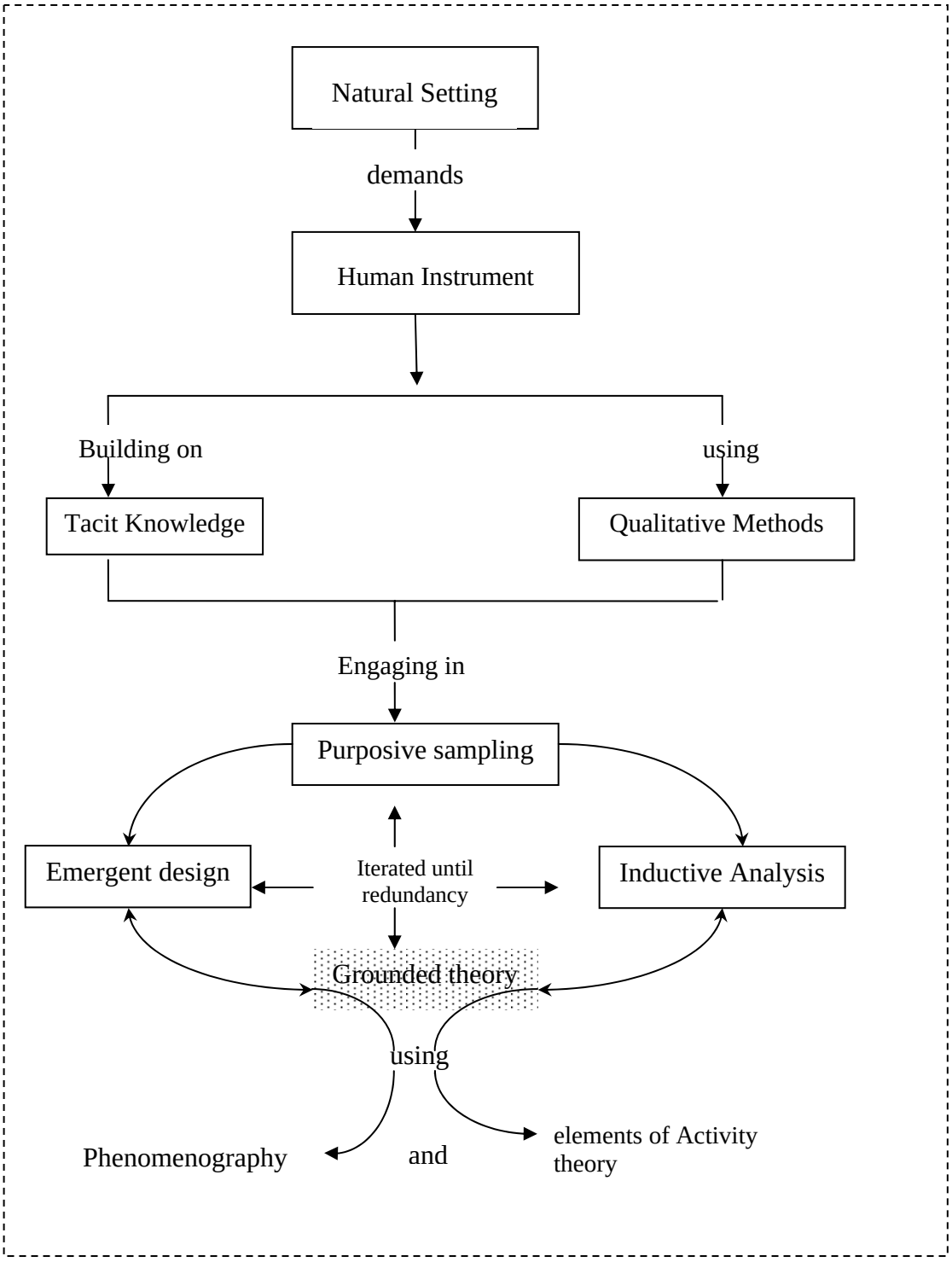


Figure 3.1 The flow of research process (*adapted from Lincoln & Guba, 1985, p. 188*)

As a result of the emergent nature of naturalist inquiry, what is of concern is to make use of the best means to "make sense" of the qualitative data in ways that will firstly, facilitate the

ongoing unfolding of the inquiry and, secondly, lead to a maximal understanding of the phenomenon being studied in its context. Therefore "expected end products are also difficult to specify". Probably all that can be promised in advance is that "understanding will be increased" (Lincoln & Guba, 1985). The following quotation aptly describes the characteristic nature of data collection within the naturalistic paradigm (Lincoln & Guba, 1985, p. 332).

> *Data is the substance of things hoped for, the evidence of things not seen.*
> *- (with apologies to King James)*

The term naturalistic in Naturalistic Inquiry alludes to the possibilities for collecting empirical material for analysis from real situations (Lincoln & Guba, 1985; Blumer, 1969). It is not about producing data in experiments or extracting data from speech events specifically arranged for analysis, but about recording what is actually said or what happens in a given situation, without direct manipulation or involvement from the researcher. In other words, what is registered as data is also possible to observe "naturally" as part of the routine interactions in a specific setting. The data can, therefore, generally be representative of other similar such settings. Against this background, the data and the interpretation thereof can be seen in a framework of the research specialization called phenomenography. A brief overview of the phenomenographic perspective will be presented in fairly broad terms.

3.2 An overview of the phenomenographic perspective

Phenomenography is an interpretative research approach. Marton (1986, p. 31) describes phenomenography as a method "for investigating the qualitatively different ways in which people experience, conceptualise, perceive and understand various aspects of, and phenomena in, the world around them". The variation in ways of experiencing the phenomenon of interest (Marton & Booth, 1997, p. 111), and the implications thereof, is the object of the research. Marton and Booth point out that although phenomenography is not a method in itself, nor is it a theory of experience, there are methodological elements associated with it and theoretical elements to be derived from it. In other words,

phenomenography is a way of (or an approach to) identifying, formulating and tackling certain sorts of research questions which are particularly of relevance to learning and understanding in an educational setting (Marton, 1986). A fundamental assumption underlying phenomenographic research is that there are a number of qualitatively different understandings of a particular phenomenon. The research focus of phenomenography is the intention to uncover variation in participants' experience. Sampling of participants for inclusion in a study, therefore, aims at capturing the breadth of variation in perspective in the population targeted. The phenomenographic approach was selected for this study on the basis of its potential to reveal variation in ways (Bowden & Marton, 1998) introductory programming students (pre-service) and experienced students (in-service teachers) of programming experience the act of learning to program in a new language; the new language being an object-oriented language.

Using a phenomenographic perspective in this investigation implies a particular way of looking at learning in terms of method and epistemology. From the point of view of method, phenomenography is concerned with what Marton (1981) terms an "insider's perspective" of what the learner is trying to achieve within the process of learning. An insider's perspective (Marton, 1981) is a second order perspective. This means that our concern is primarily focused on how the learner construes (experiences) the world. From a first order perspective, focus would be on the object of the research; that being tacit statements about the world (in other words, about phenomena) while keeping the experience in abeyance[1].

The insider's perspective that the phenomenographic method uncovers is not about the "inner world" of the learner, but about how the learner sees his or her relation to the world. The phenomenographic method brings to the fore "the student's externalization of his or her relation to the learning task" (Ramsden, 1988, p. 20). As such, a conception (category) of a particular phenomenon is not regarded as something that is inside the individual, but as something "between the student and the learning task". This relational perspective has important implications with regard to learning. The unit of research in phenomenography is

[1] To suspend judgement according to Marton & Booth (1997).

"a way of experiencing something", which has been posited to be related to how a person's awareness is structured. The experience as such contains a "what" aspect i.e. the content that is being learned, refers to the direct object, and a "how" (act) aspect i.e. the way learning relates to the phenomenon. The "*how*" aspect of learning is made up of its own "*how*" and "*what*" aspects. The latter "what" refers to an indirect object that means the quality of the act of learning, while the "how" aspect refers to the act of learning. Marton and Booth's diagrammatic representation of the structure of learning (as shown Figure 3.2) enables us to consider the meaning of learning in terms of the *how* and *what* of learning. Marton and Booth (1997) remind us that these distinctions are made merely to distinguish different research points of view, even though they are not separate entities.

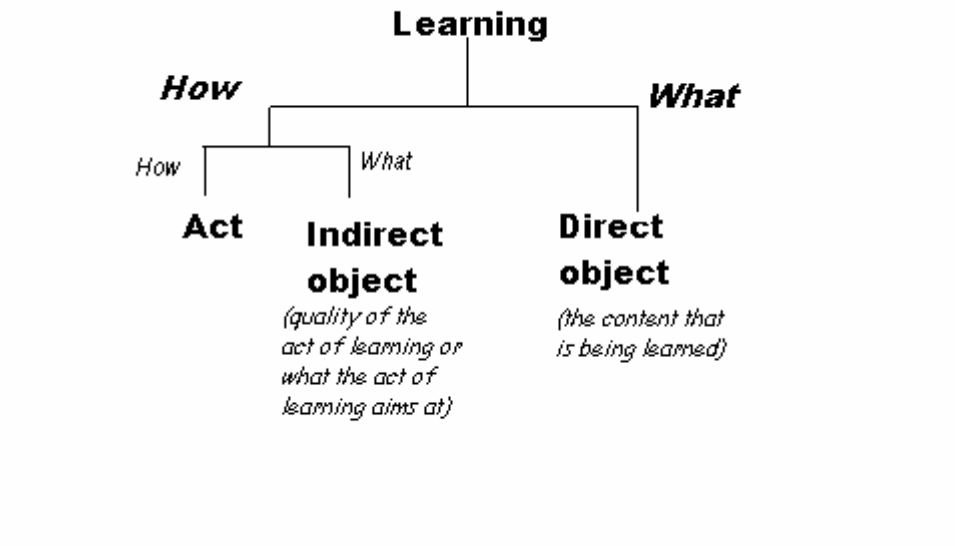


Figure 3.2 Structure of Learning

This study adopted a relational perspective in order to explore the relation between the structural aspects (how) and the referential aspect (what – the meaning) of the experience of learning to program. The referential aspect relates to the overall meaning that the students attribute to the research questions. The structural aspect of the experience is further delimited into external horizon (the boundary) and the internal horizon.

The rationale behind this type of research is the acknowledgement that people act on their interpretation of the situation in which they find themselves (Säljö, 1988; 36). Phenomenography shares this approach with other research perspectives, such as individual and social constructivism (Säljö, 1988).

Phenomenographers are not only interested in the variation in ways of experiencing a particular phenomenon, but also in the "change in capabilities" for experiencing particular phenomena in the world. These capabilities, as a rule, can be hierarchically ordered with some capabilities being more complex than others. Differences between them may be "educationally critical" (Marton & Booth, 1997, pp. 125-126) differences, and changes between them are considered to be the most important kind of learning. In this regard, Dahlgren (1984) aptly describes learning as a change in capabilities as follows:

> In other words, when learning has occurred, there is a shift from one conception to another which is qualitatively distinct. …Thus learning, within this perspective, is not a discrete and self-contained entity but one which has the potential of enabling individuals to consider afresh some part or aspect of the world around them. (Dahlgren, 1984, p. 31).

The set of variation in ways of experiencing a phenomenon is presumed "finite" (limited), but not closed. With new discoveries, there may be possibilities of new ways of seeing a phenomenon. It is prudent to point out that, phenomenographically, we claim only that the student has shown a capability for experiencing something in a certain way, and we do not say that he or she is incapable of experiencing it in another, perhaps more complete, advanced or competent, way. If one conducts another similar study, the possibility may exist that the variation in ways of experiencing learning to program in an object-oriented language could change.

3.2.1 Categories of description

The characterizations of the variation in ways in which a phenomenon is experienced are logically related to one another, generated by, in phenomenographic terms, "categories of description" (Johansson, Marton & Svensson, 1985). The qualities of the set of categories of description are determined by a set of criteria that are considered to be grounded in the structure of awareness. These criteria are:

(i) "each category tells us something distinct about a particular way of experiencing the phenomenon";

(ii) "the categories have to stand in a logical relationship with one another" and can be hierarchically arranged; and

(iii) "as few categories should be explicated as is feasible and reasonable, for capturing the critical variation in the data." (Marton & Booth, 1997, p. 125)

The set of categories of description capturing the different ways of experiencing a phenomenon is called the "*outcome space*" (Marton, 1992b). The outcome space is a "complex of categories of description comprising distinct groupings of aspects of the phenomenon and the relationships between them." When one talks about a way of experiencing something one usually does so in terms of individual awareness, and when one talks about categories of description one usually does so in terms of qualitatively different ways in which a phenomenon may appear to people at a "collective level" (Marton & Booth, 1997). The system of categories should be complete with respect to the collective experience of the population under investigation, in this case, the pre- and in-service students. However, the system of categories presented can never be claimed to form an exhaustive system.

3.2.2 The Structure of Awareness

From the text comprehension studies (Marton & Säljö, 1976a), it was found that the outcome of learning is always related to the approach that the learner employed to arrive at it. Similarly, in the study of conceptions, it was found that people's understandings of phenomena are related to the ways in which they approach them. A way of experiencing is, therefore, discernible as containing a "what" and a "how" aspect. Marton (1988) explained this in terms of the theory of gestalt, which means that for whatever one sees or experiences, one perceives a gestalt quality; that is, a whole (figure) which is discernible from its surrounding (ground). Furthermore, a gestalt is never without a structure, as a gestalt can be defined as an ensemble of items which mutually support and determine one another (Gurtwistch, 1964).

Based on the gestalt theory Marton and Booth (1997) analysed a way of experiencing as a basic unit and explained the awareness as constituting two aspects: (i) the delimitation of a theme (or figure) from the context; and (ii) the discernment of the parts that contribute to the theme. To experience something in a particular way, there is always a discernment of the whole from the context. A figure is composed of its component parts and the interplay of each part contributes functionally to the whole figure, which carries an overall meaning. In other words, it involves the delimitation of the parts and their relationship to one another within the whole (Svensson, 1984).

The delimitation of the theme further suggests that certain elements or aspects of the context come into our focal awareness to constitute the theme, while other aspects recede to the ground of a particular conception (understanding). It would be wise to ask the question: "how can a certain aspect or aspects of the phenomenon become one that occupies our focus and become discerned?" Marton and Booth (1997) explain the answer in terms of variation. They argue that without variability, many of the conceptions that we employ would not have been there ("e.g. gender would not be noticed if there were only one (Pong, 1999)!"). The whole, the parts, and the relationships between them, are discerned in terms of various aspects such as frames of reference, topics, subtopics, etc. Such aspects represent dimensions of either explicit or implicit variation in awareness.

At this point, it is necessary to clarify the two aspects of variation alluded to in the discussion of phenomenography. Marton and Pang (1999) explain this in terms of two faces of variation within phenomenography. The first face of variation refers to the categories of descriptions and the outcome space which are derived from the study of the variation between ways of experiencing the same phenomena. The second face of variation refers to the variation corresponding to the critical aspects (discerned and focused upon simultaneously) of the phenomenon, i.e. the dimensions of variation, which are scrutinized within the framework of the structure of awareness, within a unit of conception.

In section 3.4, phenomenography and its implications for this study will be discussed in detail. But for now it seems appropriate briefly to review cognitivist perspectives on learning.

3.3 Cognitivist perspectives on learning

3.3.1 Constructivism

Empiricists emphasize those aspects of scientific knowledge that are closely related to experience, especially as formed through deliberate experimental arrangements. Marton and Booth (1997) describe the empiricist's view of learning as knowledge that exists "out there" and is taken in "ready-made" from the environment. Constructivism provides an alternative explanatory framework for this "inner" and "outer" dichotomy (Marton & Booth, 1997), by assuming that the human being constructs his or her knowledge through his or her acts (an active role) and through interaction with the environment (cultural practices, language, and other people). Constructivism emphasizes not merely how individuals receive materials to be learned and how they 'construct' such material inside their heads, but how they and their teachers construct it between them, through dialogue. Bruner (1973), one of the chief proponents of constructivism, believes learning is an active process in which learners construct new ideas or concepts based upon their current and past knowledge. Different emphases exist within this perspective. Plato and the likes assumed that knowledge is innate, whilst "radical" constructivists assumed that knowledge about the world corresponds to reality as we experience and make sense of it. The perspective offered by the radical constructivists is that human knowledge is evaluated according to its cognitive viability in the minds of individuals. Radical constructivism is closely related to individual constructivism, which grew out of the work of the Swiss psychologist, Jean Piaget (Huit & Hummel, 2003).

3.3.1.1 Piaget's individual constructivism

Piaget's epistemology is based on the biological influences on "how we come to know". He believed that what distinguishes human beings from other animals is our ability to do "abstract symbolic reasoning". The concept of cognitive structure is central to his theory.

Cognitive structures are patterns of physical or mental action that underlie specific acts of intelligence. Cognitive structures change through the processes of adaptation: assimilation and accommodation. Assimilation involves the interpretation of events in terms of existing cognitive structure, whereas accommodation refers to changing the cognitive structure to make sense of the environment. Cognitive development consists of a constant effort to adapt to the environment in terms of assimilation and accommodation. In this sense, Piaget's theory is similar in nature to other constructivist perspectives of learning (e.g., Bruner, 1973; Vygotsky, 1978). In a study of conceptions, Marton (1981) compares the work of Piaget with phenomenography, suggesting that Piaget's initial research was very much akin to the work of phenomenography. A departure occurs when Piaget shifts from describing the world as it is seen by the child, to describing the child as he or she is seen by the researcher.

3.3.1.2 Vygotsky's social constructivism

As mentioned earlier, constructivism — particularly in its "social" forms — suggests that the learner is actively involved, in a joint enterprise with the teacher, of creating ("constructing") new meanings. Whereas Piaget has been particularly associated with individual constructivism, referred to as radical constructivism by Bettencourt (1993), the Russian psychologist, L.S. Vygotsky has contributed to the notion of social interaction in the construction of knowledge. Vygotsky (1978) states:

> Every function in the child's cultural development appears twice: first, on the social level, and later, on the individual level; first between people (interpsychological) and then inside the child (intrapsychological). This applies equally to voluntary attention, to logical memory, and to the formation of concepts. (Vygotsky, 1978, p. 57)

Vygotsky believes that the development of language and the articulation of ideas is central to learning and development. For Piaget, maturation is the central factor in development, for Vygotsky it is the social world. The potential for cognitive development depends upon the "zone of proximal development" (ZPD): a level of development attained when children engage in social behavior. Full cognitive (of ZPD) requires social interaction. The range of skill that can be developed with adult guidance or peer collaboration exceeds what can be

attained alone. Stated differently, learning is seen as an activity that takes place "between the individuals in a social group rather than solely within the individual." Although Vygotsky considers teaching and nurturing as preceding development, he recognizes a "zone of proximal development" within which instruction is most feasible and productive. In this regard, he comes very close to the Piagetian view. The difference between Piaget and Vygotsky can be summed up as follows: where Piaget believes that cognitive development consists of four main periods of cognitive growth (an internal structure): sensorimotor, preoperational, concrete operations, and formal operations, a theory that suggests that development has an endpoint in goal, Vygotsky believes that life long process of development was dependent on social interaction (emphasis is on external structure) and that social learning actually leads to cognitive development. In this sense, Piaget's theory (individual constructivism) and Vygotsky's theory (social constructivism) are "mirror images" of each other (Marton & Booth, 1997).

3.3.2 Cognitivist perspectives on problem-solving

Information processing: an expert centred approach

Information processing (IP) is a general perspective held by cognitive psychologists. This perspective holds that cognition means input, storage, transduction and transmission of processed information.

Cognitivsm embraces strains of both rationalism and empiricism (Marton & Booth, 1997). Empiricists claim that ideas and knowledge comes through experience (that is from the outside); while rationalists believe that knowledge comes from within. In its broadest sense rationalists have a view that appeals to reason as a source of knowledge and justification. This implies that, as far as problem solving is concerned, students will solve problems on the basis of their internal schemata[1]. The views of rationalist and empiricism are not mutually exclusive. It can be said therefore, that information processing derives from both the empiricist and the rationalist perspective.

[1] See paradoxes four, five and six (Marton and Booth, 1997: 8-11). How is knowledge received from the outer world meaningfully integrated by the learner if that knowledge is not pre-possessed by the learner?

Information processing emerged as a result of research on logic of the brain and the analogy between the brain and the computer (Capra, 1997). Artificial neural nets (a branch of AI) are basically mathematical models of information processing. They are modeled by the architecture of the brain – biological neural network – with neurons (which are discrete) as its basic processing elements. Capra contends that these basic elements are the means by which the human nervous system processes information. The process of cognition, therefore, involves the cognitive system "picking up" these discrete elements which presumably already exist in the outside world. Shneiderman and Mayer (1979) proposed a model on a similar basis to account for the cognitive structures, as well as the cognitive processes, involved in programming. According to the model, information from the outside world enters via perception into short-term memory (to some extent a limited memory store) to be processed (Miller, 1956). Two processes most likely to move information into long-term memory are elaboration and distributed practice. The organization of information in long-term memory may take the form of declarative, procedural, and /or imagery. To determine individuals' performance in problem solving, researchers compared their performance with how well they remembered and organized the information to complete the steps leading to the solution. An expert centered approach to problem solving will require a similar type of organization of concepts, information and relationships into a meaningful system.

3.4 A phenomenographic perspective on learning: deep versus surface approach

Most phenomenographic studies on learning have used Marton's categorization of a "deep" and a "surface" approach to learning. This study, however, has placed its investigation at a slightly different level. Rather than addressing the matter of learning approach (relating directly to Marton's categorization), the three research questions in the study respectively refer to learning to program experiences (research question 1), problem-solving strategies (research question 2) and factors influencing the strategies (research question 3). As will be seen in the analysis (chapters 5, 6, 7 and 8), different tendencies were observed with regard to these issues.

The deep versus surface dichotomy, therefore, does not offer a specific framework of analysis for the study. This does not mean, however, that it is without relevance. It was possible to perceive features in the tendencies highlighted that, even though they did not correspond directly to the deep and surface categorization, they certainly demanded an interpretation in the light of it.

The work of Biggs (1999), Entwistle (1988) and Ramsden (1992) provides some very valuable characteristics of the approaches to learning, and highlights the way in which how the curriculum is managed has a bearing on the learning process.

The compiled work of the above researchers, with regard to the distinction between the deep and surface approach, is reproduced in the table to follow (Deep and surface…, n.d.). It compares the characteristics and factors that encourage deep and surface approaches to learning.

Table 3.1 Deep and Surface Learning (*compiled from Biggs (1999), Entwistle (1988) and Ramsden (1992)*)

	Deep Learning	**Surface Learning**
Definition:	Examining new facts and ideas critically, tying them into existing cognitive structures and making numerous links between ideas.	Accepting new facts and ideas uncritically and attempting to store them as isolated, unconnected, items.
Characteristics	• Looking for meaning. • Focusing on the central argument or concepts needed to solve a problem. • Interacting actively. • Distinguishing between argument and evidence. • Making connections between different modules. • Relating new and previous knowledge. • Linking course content to real life.	• Relying on rote learning. • Focusing on outwards signs and the formulae needed to solve a problem. • Receiving information passively. Failing to distinguish principles from examples. • Treating parts of modules and programs as separate. • Not recognizing new material as building on previous work. • Seeing course content simply as material to be learnt for the exam.
Encouraged by Students	• Having an intrinsic curiosity in the subject. • Being determined to do well and mentally engaging when	• Studying a degree for the qualification and not being interested in the subject. • Not focusing on academic

	doing academic work. • Having the appropriate background knowledge for a sound foundation. • Having time to pursue interests through good time management. • Positive experience of education leading to confidence in ability to understand and succeed.	areas, but emphasizing others (e.g. social, sport). • Lacking background knowledge and understanding necessary to understand material. • Not enough time / too high a workload. • Cynical view of education, believing that factual recall is what is required. • High anxiety.
Encouraged by Teachers	• Showing personal interest in the subject. • Bringing out the structure of the subject. • Concentrating on and ensuring plenty of time for key concepts. • Confronting students' misconceptions. Engaging students in active learning. • Using assessments that require thought, and requires ideas to be used together. • Relating new material to what students already know and understand. • Allowing students to make mistakes without penalty and rewarding effort. • Being consistent and fair in assessing declared learning outcomes, and hence establishing trust (see constructive alignment).	• Conveying disinterest or even a negative attitude to the material. • Presenting material so that it can be perceived as a series of unrelated facts and ideas. • Allowing students to be passive. • Assessing for independent facts (short answer questions). • Rushing to cover too much material. • Emphasizing coverage at the expense of depth. • Creating undue anxiety, or low expectations of success, by discouraging statements or excessive workload. • Having a short assessment cycle.

The relational view of learning furnished by phenomenography has formed the basis of much of the research into student learning in higher education. Phenomenography has been fruitful in exploring students' conceptions, and understanding, of a number of topics, including physics (Prosser & Millar, 1989); literature interpretation (Marton, Carlsson & Halász, 1992); essay writing (Prosser & Webb, 1994) and computer programming (Booth, 1992, 1997).

In summary, this research indicates that learners' experiences should be considered as involving the ways students relate to the learning environment, their goals and intentions, as well as their learning strategies.

A diagrammatic representation of the phenomenon to be investigated follows.

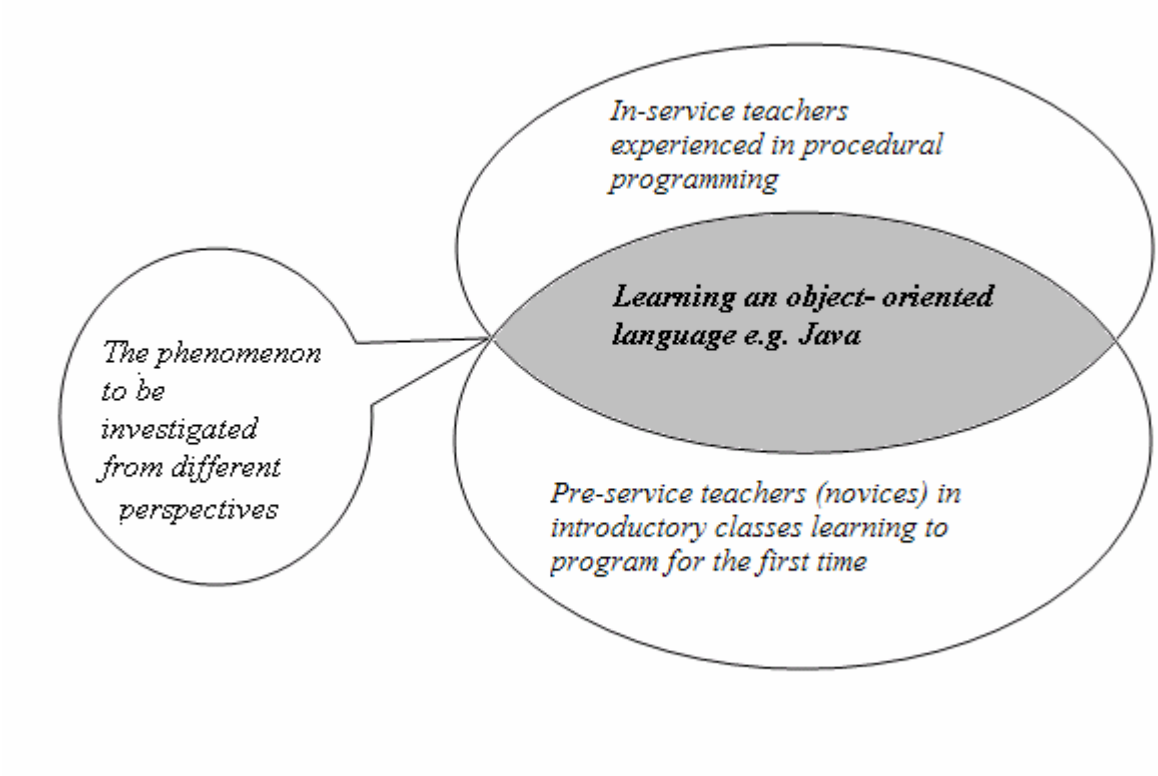


Figure 3.3 The Phenomenon being investigated

3.5 A limitation of Phenomenography

In the phenomenographic tradition, issues related to the learning environment and its role in learning, are normally relegated to the background, since research in this tradition focuses on and explores different aspects of the relation between the learner and what is learnt. As was expressed in Adawi *et al*. (2002), "the analysis deliberately strips away contextual features of the data in order to focus clearly and exclusively on the

phenomenon". However, there is always a situation with social, spatial and temporal dimensions which lends the phenomenon meaning. Marton and Booth (1997) point out that

> not only is our experience of the situation moulded by the phenomena as we experience them, but our experience of the phenomena is modified, transformed and developed through the situations we experience them in (p. 83).

Adawi *et al*. (2002) proposes that if "situation" is replaced with the synonymous "context" then they would be persuaded to study the context which gives meaning to the phenomenon. The use of phenomenography alone, as a guide to research in teaching and learning in an educational setting, is limiting, as any educator should know the importance of the learning environment to learning. This study, therefore, borrows elements of activity theory to complement phenomenography. A brief summary of activity theory is offered in the next section.

3.6 Activity Theory

The theory of activity derives mainly from the work of Vygotsky (1962, 1978). Vygotsky's contribution to the notion of social interaction in the construction of knowledge relates directly to an activity system; that is, according to Engeström (1993), knowledge construction relates to the context. In this study the activity system is the learning environment centred around the course which the subjects' (in this study, students) aim at learning. Vygotsky's (1978) definition of the zone of proximal development (ZPD)[1] emphasizes the child's potential to develop with strategic help. This support is called scaffolding by some researchers. Activity theory aims at formulating a framework for describing, analyzing and understanding complex systems or activities. Hence an activity system basically describes the interaction between the *subjects* (students), and the object (a programming task, or a computer science concept to understand) that is transformed, by interaction, into an outcome.

[1] See section 3.3.1 for a full explanation.

Wertsch (1981) describes the actual framework as being concerned with how abilities are developed: “to carry out socially-formulated, goal-directed actions with the help of mediating devices.” Activity conveys the notion of purpose and affect, and includes intellectual, as well as physical, processes. These notions are bound up with the social world surrounding the individual.

According to Leont’ev (1981) activity both orients individuals in the world in which they live, and continuously transforms the object. Leont’ev (1981, p. 46) describes activity as a functional unit of life: “a system with its own structure, its own internal transformations, and its own development.”

The idea about the design of an activity system, being a functional and dynamic unit, sheds light on how students’ learning activities develop and how they both shape, and are organised by, the setting surrounding them. The activity of learning is a process in which people grapple with new information to make it meaningful, to solve problems and to adapt to new conditions. The analysis of research question 3 is on the individual acting in his/her social and cultural world, what the learner does, why he/she takes those actions and how those actions relate to the learning context.

The ways that students experience learning to program are also likely to contribute to the difficulties that many students have with the subject. How students understand programming is related to how they interpret it in the context in which it is presented. This is an aspect of students’ learning to program which has received scant attention in the research literature and, in my experience, is rarely accorded importance in the teaching of programming courses in higher education.

Leont’ev (1978) makes an important distinction between meaning and personal sense. According to Leont’ev, meanings are connected with the reality of the outside objective world, the life of society. Personal sense, on the other hand, is connected with the reality of the person’s own life and motives. In other words, personal sense involves the incorporation of socially constituted meanings into the psychology of the individual.

The main principles of activity theory are summarized in the next section.

3.6.1 Principles of Activity Theory

3.6.1.1 Activity is the explanatory principle of psychology

Leont'ev (1981) stresses that consciousness is "generated" by and inseparable from external activity. In his view, activity must have associated external behaviour and mental images are direct products of contact with the world of objects. Leont'ev proposes that activity is at the centre of a person's connection with others and the external world. Activity includes interaction with others and it also includes reflection, but Leont'ev's emphasis is on external actions (Leont'ev, 1981). In other words individual psychological experiences are expressed through external, practical activity. Activity connects individuals to their communities, and connects material objects to the mind that illuminates these objects.

3.6.1.2 Activity Is Purposive

According to Leont'ev (1981, p. 59), there can be no activity without a motive. 'Unmotivated' activity is not activity devoid of a motive; it is activity with a motive that is subjectively and objectively concealed. Hence to Leont'ev, a motive, whether conscious or unconscious, always stands behind activity. Motives, needs and desires are translated into action by means of goals. Motivation and goal formulation are inherently socially constituted. They connect psychological phenomena to the social world. Leont'ev (1981, p. 50) proposes that when a desire meets the object of desire, it is filled "with content from the surrounding world".

3.6.1.3 Activity Is Mediated

Activity can be thought of as a structure consisting of the individual, the environment and the tool, which, as a cultural mediator, is both the agent and reflector of change. Leont'ev (1981) explains that the tool mediates activity and, in this way, connects humans with concrete objects, and also with other people. He specifies that it is not the cultural tool, as an artifact, but the use of that tool that mediates or shapes mental processes. That is, "the

formation of uniquely human functional systems takes place as a result of mastering tools (means) and operations" (Leont'ev, 1981, p. 67).

3.6.1.4 Activity Theory emphasises the interdependence and interaction of sociocultural and cognitive processes

The view that the cognitive abilities of the child or novice emerge from interaction with adults or experts was developed by Vygotsky (1962), who proposed that cognitive processes are carried out from the social to the individual plane. That is, they are first carried out initially with help from an adult or expert, and then transformed to a way of working independently, via the zone of proximal development (Vygotsky, 1978). Speech is the major mediational means which directs this process.

In order to present a framework of how activity theory can be applied to computer programming, it is important to explain both the context of the learning being investigated, and the theory by which the findings are interpreted. The theory used is drawn from the framework developed by Leont'ev (1981).

3.6.2 The Activity of Learning to Program

While activity may be viewed as an abstract construct, Leont'ev (1981) emphasises that in reality we always deal with specific activities, within a finite time, space and setting. His three levels of analysis provide three vantage points on a fundamental question for any investigation from the perspective of activity theory: the question of what an individual or group is doing in a particular setting.

In order to understand what a particular group of students is doing in learning to program, responses can be formulated in terms of each of these three levels. The first level concerns the cultural or historical situation of the students' actions. This is an account of students' learning from the perspective of the global level of activity. To respond to the question at this level it is necessary to understand students' actions in the institutional context — while studying a university course.

A university has its own well-defined social practices. Students' interpretations of these practices are bound up with their perceptions of what is expected of them and will be fundamental to their actions. According to Wertsch (1985, p. 212): An activity setting is grounded in a set of assumptions about appropriate roles, goals and means used by the participants in that setting. This setting guides the selection of actions and choice of tools. It also determines the function of the activity. Wertsch (1985) explains that settings are not determined by the physical context, but are created by the participants in the activity. Further, assumptions about the setting are often implicit, rather than consciously identified. Participants may not identify what organises their performance. An individual's understanding of the setting emerges as a "byproduct" of interacting with others in it (Wertsch, 1985, p. 216). To say that a student is engaged in the activity of learning a university subject, programming, simply tells us that the student is working in a particular socioculturally-defined setting.

To understand activity at Leont'ev's (1981) second level of analysis, one must look at the actions which, in his view, are defined by goals or partial goals. These goals are not automatic or fixed in advance. They are tested by action and remain fluid throughout the process of selection and testing.

Finally, a response to the question about a student's learning to program can be formulated in terms of the tools and operations that the student has at his or her command. This is the third level of the activity of learning to program that may be investigated. The availability of tools as mediating devices will partly be determined by the setting, for example, what resources are deemed appropriate for students to use. Tools and operations also depend on the student's personal history, i.e. his/her experiences, including the student's repertoire of skills, such as his/her ability to use a computer and the specific IDE of the language being used. Affective elements are important as well. If the student's memory of using a computer at school is linked to negative emotions, this tool will not be readily utilised. The tools or operations used by a student affect how successfully the student is able to carry out his/her intentions.

3.6.3 Background to my application of Activity Theory

In this section I describe how activity theory can be used as a lens to understand students' learning to program.

A focus on learning, as an activity that is socially and culturally situated, is related to exploring what Varela, Thompson and Rosch (1991, p. 213) term "histories that are lived". From a societal perspective, each person is spatially and temporally situated. The self speaks "not only as an individual voice but also as a collective voice", reflecting values and beliefs of communities and cultures which are significant to the individual (Hermans & Kempen, 1995, p. 112). From an individual perspective, each person develops a personal story, a history of his/her own experiences which have left their mark. These perspectives of lived histories, act in tandem in human activity; such as the activity of learning to program. A student's activity is related to his/her situation in an institutional and societal setting. Interwoven with these other histories, is the personal narrative of the student, both the biological and social development, which contributes to his/her ways of understanding programming and of interpreting the context in which he/she encounters it.

A key argument of activity theory is that personally meaningful goals have an "energising" function (Leont'ev, 1981, p. 60). Varela *et al.* (1991) specify two aspects of goal directedness or intention. These correspond to (p. 206) what the system takes as the possibilities for action to be, and to how the resulting situations fulfil, or fail to fulfil, these possibilities. In this investigation, it means that actions taken to learn to program are directed by how the student construes his/her goals in context. In turn, how these goals become fulfilled, or fail to become fulfilled, is affected by the conditions surrounding the students's actions. Hence the principle that learning is purposeful, or goal driven, links the two aspects of activity theory that have been explained; the sociocultural situatedness of activity, and the directed actions comprising the activity.

In summary, interpreting a student's learning from the activity perspective means attending to the "system of activities" (Leont'ev, 1981, p. 46) or practices in which the student is engaged and understanding how the student is oriented or positioned with respect to these

particular practices. At the level of the individual, activity brings together a goal, the means for realising this goal, the actual process of transformation and its effect or outcome. Each of these is organised within a societal setting. The "system" (Leont'ev, 1981, p. 47) of collective activity which relates people to one another, their cultural history and their realm of action is called an "activity system" by Engeström (1993, p. 66).

In the next section, some elements of the framework of activity theory, which are of interest in defining the activity of learning to program, are presented. These features are illustrated with examples drawn from my teaching practice.

3.6.4 Views of activity from My Practice

In order to clarify the ideas introduced in section 3.6.1 and 3.6.3, a summary is presented of my interpretation of key features of activity theory, by means of examples. The following excerpts are from transcribed interviews with students during the programming course, and journal entries of in-service teachers. They function as "snapshots" taken from different angles, of an activity system of students acting in their institutional worlds. They capture images of ongoing and dynamic processes.

Activities are distinguished by their energising motives

In an interview with a pre-service teacher, he said this:

> *S2: It's challenging, it tackles your mind; it makes you think. The fact that your program is running, makes you feel good it makes you think that you can do something, therefore, I feel challenged.*

The above quote illustrates the essence of activity as encompassing his total perspective on programming. His actions in the educational setting were guided, and directed, by his underlying need to learn to program, and this resulted in a very favourable outcome.

Activity theory emphasises the interdependence and interaction of learning processes and cultural background

The following excerpts are from the journal entries of two experienced in-service teachers.

> We find that by just meeting and talking about our subject has a positive impact on us. We wish to continue with our meetings even after the course is over. We also feel that we can learn a lot from each other and share our materials and resources.
>
> I think the idea of study groups is excellent – but I would rather work with people I see almost every day – not with people I've never seen in my life. I think the advantage of impromptu discussions (sometimes at funny places) is useful.

From the activity theory point of view, the role of social interaction is important in the learning process.

Actions are socially formulated

> *S6:* *From the onset and I did speak to Mr*[lecturer]*…, I did not have the grades to do this course and no mathematics. I had to write a motivational letter. All because I wanted to do computer science education because the way things are going technologically speaking*
> *It's going to be a must for kids to come. Somewhere along the line it's changed now. We've become a little bit of a family now, those guys that made it from first to second year. Although it is a primary motivation, I keep going each day and each lesson and also I don't want to let the guys down or Mr … down, myself down.*

Hence student S6's actions were prescribed by unwritten codes of social conduct in lectures, as he perceived them.

Activity involves self regulation

> *S7:* *I… as I was saying to Mr… I was so fine with last years work and I was getting to know it properly. But now we started this object-oriented, I'm a bit lost with that. I don't…But I was like that at the beginning with the other one, so at the moment I'm a bit unstable…like ooh! like I don't know what I'm doing, but I'll get there.*

The above quote illustrates the common formation of the student's conceptions of learning to program, and her regulation of her actions to learn it.

3.7 Summary and Conclusion

In this chapter a background to phenomenography as a research tool is provided using a naturalistic perspective to draw on the rich accounts of pre- and in-service teachers. The results of phenomenographic analysis are categories of description, which in itself is not the sole aim of the investigation, but rather the investigation of learning and teaching based on these categories of description. The importance of contextual factors, which are stripped

away in the phenomenographic analysis, is considered. Elements of activity theory are introduced to overcome this limitation.

Activity theory provides a framework for understanding the processes of teaching and learning. In this study, I focus on the students' activities in learning to program, as the students themselves report them; their actions and perceptions and how these relate to the contexts surrounding them. These activities implicitly include students' goals and evaluations. Other aspects remain in the background.

As stated in chapter 1, the national curriculum for the discipline, computer studies, at school, spells out a change in the programming language from one of procedural, to one of object-orientation. It is, therefore, necessary for an understanding of both in-service and pre-service teachers' experiences of learning to program. In this study, the research framework presented here serves as a tool to study the experience of learning to program in the perceived learning context. The use of phenomenography is therefore extended to include also the variations contextual to the study object. I try to analyse the emerging relationships among components of the students' orientations to learning to program: affective aspects, personal histories, conceptions of the subject matter and approaches to learning to program and problem solving.

My methodologies for these studies will be explained in the next chapter.

4 RESEARCH DESIGN AND METHODOLOGY

This study consists of investigations into understanding how pre-service and in-service teachers learn to program in an object-oriented language. This study is based mainly on the journal entries of in-service and pre-service students from which I tried to elicit students' feelings about learning to program, their conceptions of the subject matter and their approaches to learning to program and programming (problem solving). I also interviewed the pre-service students (see Appendix D) and selected in-service teachers involved with teaching procedural programming.

As was seen in chapter 3, the theoretical framework adopted in this study not only takes into consideration the relational aspect of learning as espoused in phenomenography, but also the relational aspect of context-place, as espoused in the activity system within the naturalistic paradigm. In practical terms, this framework translated into the method described below (section 4.1 – 4.4).

This chapter describes the principal and sub-principal research questions, explains how a range of research tools are used to investigate them and describes some key features of my approach. In Section 4.1 some issues and controversies, which are important to the current discourse on methodology in education research, are presented. The methodology, which explains the rationale for the ways of collecting and analysing data are outlined. The procedures followed are summarised. I indicate how I have attempted to satisfy criteria for good research as advocated by researchers such as Kirk and Miller (1986); Merriam (1988,

1996); Miles and Huberman (1984, 1994) and Van Maanen (1983). In section 4.2, the pilot study, which gave me insight into a very problematic aspect of programming, problem solving, is described. A description of the main study is presented in section 4.3 in which the student (participant) profile; the methods of data collection and the research setting are discussed. In section 4.4, the method of analysis is discussed. In particular, I explain the research specializations based on Marton's (1986) and Leont'ev's (1978) work, from which I developed my research approaches. These research specialisations, or combinations of research orientations and research approaches, include naturalistic inquiry, phenomenography and elements of activity theory. The validity of the study methods is considered in section 4.5. Section 4.6 summarizes and concludes the chapter.

Research Questions

The main research questions which I investigate are:

- What are the qualitatively different ways in which pre- and in-service teachers learn to program using the object-oriented language?
- What are the qualitatively different ways in which the students program (problem solve)?
- What factors (contextual) affect their approaches to programming?

As explained in the previous chapters, these questions are important, not only in isolation, but also from an activity theory[1] perspective; the way students orient or position themselves to learn is critical to the quality of learning that unfolds. That is, the way students feel about learning to program, their conceptions of programming and their approaches to learning reflect their engagement with the learning task — their activities. These activities are inseparable from the setting in which the students' learning is organised. They are shaped by, and transform, that setting continuously.

[1] The activity theory perspective suggests that a focus on teaching, should be related to students' perceptions about their own learning and its context.

4.1 Research design and rationale

This section, in which the methodology is embedded, the issues and assumptions and rationale that guided my planning and investigations are explored.

4.1.1 Research Methods

Education research is the systematic investigation of educational phenomena: a mode of thinking rather than a shortcut to answers (Nisbet, 1980.p. 10).

Qualitative methods have interpretation and exploration as the rationale for research. Some major characteristics of qualitative methodology are described by Merriam (1988, pp. 19-20) as follows:

- the primary instrument is the researcher;
- the research involves natural settings or fieldwork;
- the results are qualitative description, that is "thick" rich text (Guba and Lincoln, 1981, p.119) which gives depth to the experience;
- the inductive, rather than deductive, method is followed; that is, abstractions are built from examples.

Qualitative research is an empirical, socially-located phenomenon, defined by its own history, not simply a residual of all things that are 'not quantitative'. To me, the essence of the qualitative approach to research is the contextualising of the data. As Van Maanen (1983, p. 9) explains:

> The data developed by qualitative methods originate when a researcher figuratively puts brackets around a temporal and spatial domain of the social world.

This awareness of context implicitly signifies that the research framework is nonpositivistic — the research cannot uncover laws about humans which apply at all times and places. This is particularly important for research inspired by Vygotsky's insights in which human processes are "historically and socially determined" (Vygotsky, 1962, p. 23). In this investigation, the context in which students learn to program is an important aspect of my analytic understanding of their learning processes.

4.1.2 Methodology

The primary aim of the methodology used is holistic and descriptive interpretation, seeking for depth of understanding. Consistent with a Vygotskian perspective, I consider the sociocultural context to be inseparable from individual actions. Hence my primary concern is to interpret the network of relationships between learner, subject matter and learning context. In attempting to make the data meaningful, I have drawn on various techniques of data collection and analysis. Overall, my mode of doing research is consistent with what Lincoln and Guba (1985, p. 36) term the "naturalistic paradigm" because I take it as axiomatic that: realities are multiple; myself as researcher and the objects of research are inseparable; my working hypotheses are bound by time; and context and all entities are in a state of mutual, simultaneous shaping. Moreover, my inquiry is "value bound" (Lincoln and Guba, 1985, p. 161). That is, my investigation is bound by my assumptions, theories and perspectives and is regulated by both cultural norms, and my individual beliefs.

As with any style of research, reliability, validity and generalisability are prime concerns. However, these terms have different meanings according to the position of the researcher. In a positivist framework there is an assumption that there are "true" scores which, given a good enough instrument, can be measured reliably and with validity (Kirk & Miller, 1986). From a non-positivist perspective, however, there are no absolute truths or universal laws about human processes which can be captured. In such frameworks, the ways in which research findings are validated are socially construed (Lancy, 1993; Lincoln & Guba, 1985; Merriam, 1988, 1996; Miles & Huberman, 1984, 1994; Van Maanen, 1983).

4.1.3 Model for Methodology

The design of this study reflects my consideration that the naturalistic paradigm (Lincoln & Guba, 1985) fits best with the aims described above. That is, in this study I have adopted the attributes of the naturalistic paradigm described below (Lincoln and Guba, 1985, pp. 39-43). What I studied was, to some extent, described in advance, informed by my understanding of the importance of context, which relies on the activity theory proposed by

Leont'ev (1981). The attributes of the naturalistic paradigm are detailed below, along with a description of how I complied with each in the study.

• **Characteristic 1: Natural setting**

The study was carried out in the context of my colleagues teaching the pre-service students during their two semester course of introductory programming as well as the in-service students learning to program through a distance learning course. My observations of them, for the purpose of the study, are inseparable from my actions designed to assist them with their learning.

• **Characteristic 2: Human instrument**

I consider myself to be the primary data gathering instrument. That is, the evaluation of what the students said, did or wrote, is personal and subject to my own perceptions, beliefs and values.

• **Characteristic 3: Utilisation of intuitive knowledge (tacit)**

My analysis takes account of my experience of teaching them, and others who have sought my assistance in learning to program in general.

• **Characteristic 4: Qualitative methods**

Qualitative methods of data collection and analysis were used, namely descriptive and exploratory techniques, aiming to illuminate multiple truths in context.

• **Characteristic 5: Purposive sampling**

The students were not selected as being representative of any general group of students. Rather, as described above, it appeared to me that owing to their singular characteristics (described in section 4.3) exploring these students' orientations to learning to program (in particular object-oriented programming) would be of value in understanding how students learn programming.

• **Characteristic 6: Inductive data analysis**

I have tried to construct general themes from the particular examples provided by the students' accounts.

• **Characteristic 7: Emergent design**

The design unfolded as I observed the students and it was influenced by my interactions with them. My research questions evolved from informal observations and intuitions. They

were incorporated into the questionnaires completed by the students during their study of learning to program (both during the pilot and main study) and, thence, into the semi-structured interviews conducted with the students after they had completed a test on programming. Themes emerged in these interviews, as well as subsequent interactions with colleagues and experts in education. The design of the study was only fully completed when I had written up a report on the pilot study (Govender & Grayson, 2006) for review.

• Characteristic 8: Tentative application

While I hope that my research will be of value to other education researchers, and to practitioners of teaching programming, my findings are confined by temporal and physical boundaries — a "snapshot" perspective of a situation. However, as outlined in section 4.2.2, I have taken what steps I can to ensure that the extent of transferability and applicability of the study can be judged by others.

• Characteristic 9: Focus-determined boundaries

The boundaries of this study are determined on the basis of the theoretical concepts which frame the study, and by the particular activities undertaken by the students in the social setting.

4.2 The Pilot study

A pilot study was carried out among a group of pre-service teachers learning to program in the University of KZN, and a group of grade 10 and 12 learners, learning to program in a high school. In the first few lessons the grade 10s were given an introductory course on computer architecture before learning to program.

The participants were:

- 40 students in grade 10 starting to learn to program.
- 35 students in grade 12, after having studied computer programming in grades 10 and 11.
- 20 pre-service teachers learning to program over 2 semesters.

The collection of data for the pilot study was informed by two data sources. These were the author's personal notes, based on observations of teachers teaching computer programming in the classroom over a period of two terms, and the results of a study conducted in computer programming with pre-service teachers of computer science education over a semester. Research instruments used were three questionnaires, one of which dealt with, among other aspects the background knowledge of pre-service teachers with regards to mathematics, computing knowledge, grade 12 achievements and general comments (Appendix A1). The second questionnaire was used to capture the problem solving steps followed and the experiences in programming (Appendix A2). The group of pre-service teachers and the group of grade 12s were given a problem (included in Appendix A2) to solve using the computer and were asked to answer the questionnaire. The questionnaires were administered towards the end of the semester for the pre-service teachers and during the middle of the year for the grade 12 students, bearing in mind that the students had a fairly good knowledge of the language of programming (which were Delphi and Pascal, respectively). In planning their solution, they were asked to write down their thinking process at each step of the plan, using the questionnaire which captured their approach to solving the problem. In addition to the use of this questionnaire, five of the pre-service teachers were interviewed. The pre-service teachers were asked to explore their understanding of the problem and I was able to capture a sense of their feelings, frustrations and/or achievements during the course of their study in programming. These semi-structured interviews were audio taped. Furthermore, the teachings in the classroom of two sets of grades 10 and 12 students, from two different schools, were video-taped for later analysis. This enabled me to observe both teachers and students teaching and learning respectively.

To further consolidate the experiences of teaching and learning programming in the high school, a (third) general questionnaire (Appendix A3) on problem solving was administered to the grade 12 students. This questionnaire consisted of questions related to the kind of problems they solve, as well as the processes used to solve problems during the last two years of their study. Students ranked the statements with strongly disagree, disagree, neutral, agree, and strongly agree.

I was interested in the processes students used to complete programming problems in general. The survey responses to the ranking questions were analyzed by simply calculating the frequencies of the agreements and disagreements for each question. The responses were further categorised according to the questions: “How do you learn to solve problems?”, and “What types of problems are you given?”

The analysis of the pilot study paid attention to the issues with regard to high school learners, pre-service teachers and in-service teachers, learning and teaching programming. From the pilot study it became clear that:

- ✓ Maths is essential to successful programming, in particular the problem solving abilities that are gained from mathematics.
- ✓ The participants’ problem solving skill, within the context of programming, is weak. Learners are challenged by programming and often drop out of the course, maintaining that it is difficult.
- ✓ Teachers do not teach problem solving techniques explicitly and, hence do not emphasise the problem solving techniques required to solve problems in programming. More time and effort is given to the syntax of the programming language.

These findings point to an important concern in programming education: problem solving and the teaching thereof. In this sense, the pilot study has motivated and contributed in part to the main study: understanding pre- and in-service students learning to program, particularly in the light of the new programming language being introduced.

4.3 The main study: How pre-service and in-service students learn to program in a new language

4.3.1 Selection criteria for research participants

The participants were pre-service teachers in an introductory programming course and in-service teachers within a distant learning programming course. It is important that focus be placed not only on the experience of learning but also on problem solving within the formal situation in which problem solving takes place, namely that of the test, assignment and examination. The tests were the normal, routine tests, given as part of the course work for the pre-service students, while we relied on the assignments as part of the course work for the in-service students and the examinations gave us the overall assessment of students' ability in programming.

4.3.1.1 Student profile

The participants were:

- A group of 315 in-service teachers registered at UNISA in the department of CIMSTE for the course "Computing for Teachers 1b: Introduction to programming using Java". The purpose of this formal course was to help the practicing teacher (in-service), not only to learn how to program in Java, but also to teach programming in an effective way. The in-service teachers resided in different provinces in South Africa; however, the majority of the participants (in-service teachers) resided in the province of KwaZulu-Natal. It must be noted that the full complement of journal assignments and background questionnaires were not submitted for assessment and analysis. There may be many reasons for this; however, with most distant learning courses, it is common for many students not to submit all assignments or for students to drop out during the course of the year. Furthermore, the full complement of the in-service teachers did not sit for the examination.
- A group of 12 pre-service students learning to program over a period of two semesters.

Features of the students' profiles are highlighted and summarized in Table 4.1 below.

Table 4.1 Student profile

In-service teachers		Pre-service students	
• Mature • Experienced in Pascal programming • Strong pedagogical content knowledge (pck) of Pascal programming	• Mature • No previous experience in Pascal or general programming • No pedagogical content knowledge	• Not so mature • No previous programming experience • No pedagogical content knowledge	• Not so mature • Previous high school programming experience • No pedagogical content knowledge

Distinguishable features of both in-service and pre-service teachers are tabulated. The kinds of comments made in their journals are indicative of their differing characteristics. Observations made by in-service teachers, who are experienced in programming (Pascal), are insightful and supply valuable material for stimulating reflections on teaching (Slotnick *et al.*, 1993).

4.3.2 Research Instruments: Data collection

4.3.2.1. Reflective journals

Teachers' reflections on their experiences in teaching programming and learning a new language in programming, were reflected in the journal. The journals served as records of growth and were the main source of information about the in-service teachers' learning processes. These journals were part of the compulsory assessment for the in-service teachers, who were enrolled at UNISA. The in-service teachers submitted two journals during the (year) course of their study. In a tutorial letter, the following extract was given to students to clarify the kind of entries to be written.

What to write in your journal

Other than the specific activities which we ask you to do in your journals, you are also required to reflect on your experiences as you work through the course material. It is important that you discipline yourself to make regular entries as you progress. Your journal will help you to reflect on your own learning and also think about how to teach the various aspects of Java to your learners. Some of the things you may want to write in your journal are:

- **Thoughts and ideas** you have as you go along e.g. if you notice something that relates to your prior knowledge, if you find yourself making an analogy, if you get stuck and what you do to get "unstuck".
- **Questions and problems** you have as you go along. Note that you might find answers to these questions later on, but still jot them down. It can be useful to see later on what questions you had at the time.
- Your **feelings** (frustrations, excitement, confusion, etc) as you are working through a section. This could serve as a reminder to you how you felt when you worked through a given section, so that you can anticipate how your learners might feel.
- **Notes** to yourself. These can include any new facts that you want to remember or summaries of sections, etc. One teacher used her journal to keep a record of the file names she used for each exercise, so that she could easily find them later. Include anything that you would find useful.
- Ideas you get along the way about **how to teach** a certain topic, or problems your pupils may have.
- Specific **comments about the workbooks and the study guide** which could lead to improvements, e.g. "on page 24 I found it confusing when it said…..it would be easier if it said…'.

In assessing the journal, marks were awarded based on how regularly they wrote in their journals, and how much careful thought, honesty and effort went into writing in the journal. There was no right or wrong answers.

These journals are useful tools in aiding understanding of the mental processes that students engage in as they read, write and problem solve (Carr, 2002). The group of pre-service students was also asked to keep a journal of their reflections of learning to program as part of their assessment. The pre-service students submitted one journal at the end of the

semester. A similar extract to the one given to in-service teachers was given to the pre-service students to clarify the kind of information to be entered. Their questions and comments on problems were based on a Java guide produced for the pre-service students at the University of KZN.

4.3.2.2 The interviews

Two sets of interviews were conducted with 7 pre-service students during the course of the semester programme. The first set of interviews dealt with their learning of programming in general. The second set of interviews was conducted after a test was written. The interviews were audiotaped and transcribed.

The interview focus

In the light of the research questions, the interviews primarily sought to elicit the following:

With respect to research question 1 (which dealt with the qualitatively different ways in which the students go about learning to program) there was an overriding concern with students' low achievements in programming; i.e. they had to explain what they *did* during the process of learning to program.

With respect to research question 2 and 3 (which dealt with the different ways in which the students go about solving problems as a means of learning to program) there was an overriding concern with students problem solving strategies, students' appraisals of the context of learning to program and how they saw themselves in that context.

The interview situation

Before the interviews took place, analysis was done of the pre-service students' attempts at given problems, as reflected in their test scripts and observations during lab time. Although the course of the interview was still largely dependent on the student, this procedure helped in formulating the questions to be used during the interview. It is in this sense that the interview could be regarded as "semi-structured". This approach also made it possible to draw attention to discrepancies between students' problem-solving attempts, in the test and during the interview (as well as other problem-solving contexts), and to elicit the students' rationalizations of such discrepancies. This brought into focus the question of "contextual

dependency" – the extent to which the different contexts of problem-solving lend themselves to different approaches, strategies and conceptions on the part of the students.

The interview design and method

It is important for both the researcher and the interviewees to understand the parameters of the interview contexts, so as to ensure that the interviewees do not focus on "perceived contextual demands, but on the content of the problems under discussion" (Booth, 1992:60). In this sense, the validity of the research study is seen by Booth to be dependent on the interview. The two most significant qualities that characterize a phenomenographic interview method are its sensitivity with regard to "shift in focus" and with regard to "potentially productive turns in the discourse" (Booth, 1992, p. 60-61). Therefore, the degree of awareness and reflexivity on the part of the researcher are important concerns (Hammersley & Atkinson, 1983).

On the basis of the research questions, the semi-structured interview concentrated on two sets of questions. The first set of questions used in this study was based on those from previous studies on learning to program (Booth, 1992; Stoodley *et al*., 2004).

1. What is "programming"? What do you understand by the term?
2. Can you write a program that works? How do you know?
3. Can you write a good program? How do you know?
4. How do you see your current ability to program?
5. Can you describe the process you go through when you write a program?
6. How have you learnt to program?
7. Describe for me how you went about learning to program when you first started? What language did you learn first?
8. What are the most demanding/ difficult parts of programming?
9. Do you enjoy programming? What makes it enjoyable or not?

The second set of questions was adopted from Good and Smith's framework observation of student's problem-solving practice (see Good & Smith, 1987, p. 33-34). Good and Smith's framework was adopted for one important reason; that is, they clearly advocate the use of naturalistic methods into problem-solving inquiry.

1. What were your feelings about the test?
2. How did you prepare for the test?
3. How did you interpret the problem?
4. Why did you follow this interpretation?
5. How did you go about solving the problem?

6. Why did you go about solving the problem the way you did?

4.3.2.3 Questionnaire

Two sets of questionnaires were used in this study. The initial questionnaire dealt with, among other issues, the background knowledge of pre-service teachers with regards to mathematics, computing knowledge, grade 12 achievements and general comments with regard to their perception of programming (refer to Appendix A1). A similar questionnaire was also administered to in-service teachers learning the new language (see Appendix B).

4.4 Analysis of data

4.4.1 Criteria

Sieber (1976) suggests that good analysis usually involves the following:

- intertwining of analysis and data collection;
- formulating classes of phenomena — that is, categorisation of concepts;
- identifying themes — linking concepts, noting regularities and patterns;
- provisional testing of hypotheses — looking for associated variation, ruling out confounding factors, and identifying intervening variables, which are relevant for quantitative research.

4.4.2 Analysis of data related to Research Question 1

What are the qualitatively different ways in which pre- and in-service teachers learn to program using the object-oriented language?

I based my analysis of students' experiences of learning to program on the phenomenographic method that was developed by Marton and Booth (1997), in which categories of description are formulated. The categories of description, and their patterns of distribution, are considered to be major results of the study. These will be described in detail in chapter five.

The phenomenographic analysis of the data on students' conceptions involved three stages.

The first stage in the analysis of the data was to identify a set of qualitatively different categories of description of students' conceptions of learning to program. This involved the following procedure:

1. An initial set of categories was identified by another colleague at my institution and I. This was achieved by our independently reading and classifying the entire set of journal entries, as well as the interview transcripts.
2. We then compared and discussed our initial categories and agreed on a draft set of categories and sub-categories.

The second stage consisted of classifying the in-service and pre-service teachers' responses into these categories. The analysis also involved identifying themes which connected the categories. The themes were derived from what the students said; that is, common threads in the interview data and journal entries. During the course of this process the categories were clarified and refined.

In the third and later stage I checked each transcript to identify any that I felt had been misclassified, and submitted these to further examination and analysis. The analysis involved integrating my interpretations and explanations into a coherent description resulting in an outcome space as depicted in figure 5.2.

4.4.3 Analysis of data related to Research Question 2

What are the qualitatively different ways (strategies) in which the (in-service and pre-service) teachers go about solving introductory computer programming problems?

The strategies with which the problems were solved constituted the students' structure of awareness (see section 3.2.2). These strategies were analysed on the basis of different steps that characterized the students' problem-solving process.

A phenomenographic method was used in identifying different steps in the students' approaches to programming (problem solving). Each pre-service student's test script and the interview transcript were analysed and distinct steps were identified. The interview transcripts from the pilot study were subjected to this analysis as well. The in-service

journal entries were also considered and during this process, the steps were clarified and refined. The second stage of analysis involved grouping the steps into approaches to programming by checking each transcript for the specific step identified, which posed the greatest challenge.

4.4.4 Analysis of data related to Research Question 3

How does the setting or learning context influence the approach adopted during learning to program and problem-solving?

In this study, the thematic field consists of those spatial and temporal factors that have a bearing on the students' approaches, as reflected through their intentions and conceptions of problem-solving and the meanings attached to the different settings of problem-solving. Familiarity (structure of relevance) is an important concept for characterizing the meanings students attached to the various instances of problem-solving across the different settings. The purpose of the analysis was to discern how the learning context, delineated into different settings, influenced their learning and problem solving approaches. The exploration of students' familiarity with the problems brought to the fore the students' personal contexts, which related to the meaning the students attached to their problem-solving approaches. This level of analysis (personal context) incorporated ideas of institutional context.

In the first level of analysis, I arrived at categories of description that were: demographic information; students' goals with respect to learning to program; their perceptions of programming; perceptions of context in the activity of learning to program and approaches taken to learn to program. Themes were derived from what the students said and wrote in the journals. Leont'ev (1981) identified three levels of analysis of activity: milieu or social setting of the activities, the goals directing the actions and the operations. For the purpose of this study, the global level (that is, the milieu or social setting of activities) was used as a vantage point for analysis.

The following considerations served as guidelines to the analysis:

a) How students used particular settings (and – in the case of studying, for example – the different means of studying at their disposal). How the setting used reflected the influence of physical context on problem-solving; and
b) based on the findings of a), How the students related the physical context to their personal contexts – bringing to the fore their intentions and conceptions.

A further dimension of the learning context was considered: teachers' perceptions and experiences of teaching programming. Two teachers were interviewed, which offered some insight into the influence of teachers' perceptions.

4.4.5 Higher Levels of Analysis: Developing Relationships and Themes

I analysed students' background questionnaires from the pilot study, as well as those of the main study, to explore the inter-relationships among the variables. That is, I explored the relationships among students' conceptions of programming, their attainment in tests and examinations on programming, and various demographic and individual difference variables, such as prior level of programming studied. Strong patterns and relationships were found. These are discussed in chapter Eight.

The data collection and analysis was not a linear process. I was interviewing students at the same time as I was analysing the tests and exam results, and some of the interviews of teachers (from the pilot study). Each stage of the analysis and data collection resulted in my reviewing the earlier stages and, in many cases, rethinking my conclusions and proposing new conjectures. For example, the analysis of data on students' marks, which became available at the end of the year, guided me to new insights about the students' experiences. My exploration of students' responses also suggested questions about lecturers' perceptions and led to my (informally) interviewing two lecturers. Themes emerged — such as the relation between students' "personal sense" (Leont'ev, 1978, p. 92) of programming and the meaning of programming as collectively understood.

Finally, themes arising from the study are synthesized and are explained in chapter nine.

4.5 Validity and reliability of the study

Three aspects of validity are highlighted in phenomenographic research:

- Content-related validity – the research has to be grounded on a sound understanding of the subject content;
- Methodological validity – the phenomenographic perspective should permeate the study from its data collection stage, to the analysis and presentation of the results; and,
- Communicative validity – the study should have both "internal" (relative to the participants in the study) and "external" (relative to other researchers, both within phenomenography and outside) reference (Booth, 1992).

With regard to content validity, the researcher has to have a "deep but open familiarity with the topics taken up by the interviewees" (Booth, 1992, p. 65). As was mentioned earlier (chapter 1) I had taught computer programming to pre-service teachers and was part of the examining panel for the senior certificate examination, as well as instrumental in assessing the assignments of the in-service teachers learning the new language. While assessing the journals of both in-service and pre-service teachers, as well as discussions with lecturers who taught the programming modules, I was able to gain a sound perspective on their expectations and understanding of the programming course.

The phenomenographic perspective informed all the stages of the study: formulating the research questions, collecting data, analyzing data and summarizing the results. Particular care was taken with regard to documenting the variation in the students' conceptions. At the same time, the analysis of the data, in particular, was done in such a way as to draw attention to the variation in the experience of the phenomenon, learning to program. The study, however, also drew on other research perspectives, most notably activity theory in terms of the context in which the students experienced learning to program. While it cannot prescribe methods to be applied in specific instructional settings, it does provide a perspective for educators and researchers to see students, the learning context and the links between them in new ways. The activity approach provides insight into the roles of

teachers. It suggests a view of students as active participators in their own learning — a view which, I think, fits with the intuitions of many teachers. It draws attention to the diverse ways in which students seek meaning within the cultural and social arena of their actions.

Phenomenography deals with descriptions of lived experience. Booth compares a phenomenographic researcher to an explorer. She points out that if another (second) explorer were to be given "original charts, the observations and the sightings, the diaries and notebooks", she would be likely to reach similar results to the first, on condition that "both the explorers have similarly thorough experiences of what it is to explore foreign lands, and prior understanding of the sort of territory and culture which might be encountered" (Booth, 1992, pp. 66-67).

Cross-checks to improve the methodological reliability of this study were made during the process of iteration in the data analysis phase, in which the categories of description were formulated. Fortunately, another colleague, who was also involved in the assessment of the students' journals, was able to agree on the descriptions of the categories of conception. Informal discussions with students (both pre- and in-service) also revealed consensus about these categories, and the contextual factors that influence their experiences and perspectives of learning to program.

To what extent can this study draw on other researchers? As shown in chapter 2, programming research has, to a large extent, been dominated by cognitivist "expert versus novice" studies. While several phenomenographic studies have shown interest in students' conceptions of learning to program, as well as particular programming constructs (for example Booth, 1992; Bruce *et al*., 2003), relatively little attention has been paid to studying learning to program (from the perspective of in-service and pre-service teachers) and problem solving as a process which takes place in space and time. A particular challenge in this study was, therefore, the difficulty of characterizing the problem solving process.

4.6 Conclusion

The aim of research is to illuminate the phenomena under investigation. To do this, a spectrum of methodologies is available to the researcher. Methodologies are not defined by the types of data collection or analytic techniques deployed, but by the purpose of the investigation. In this research, my aim was to understand students' orientations to learning to program from their own perspectives, taking into account the context of their responses. The research tools I used depended on the questions I wanted to answer. In this study, these required exploration, rich description and qualitative interpretation. Moreover, some questions required a more systemic analysis, involving categorization of concepts, distributions, patterns and relationships. By using a variety of methods of data collection (interviews, journal writings and questionnaires) and analysis, I have endeavoured to shed light on some aspects of the complexity of learning to program as experienced by pre- and in-service teachers.

Chapters 3 and 4 have set the theoretical and methodological frameworks used in this study. This has laid the foundation for the presentation of the results of the study.

5 A PHENOMENOGRAPHIC ANALYSIS OF LEARNING TO PROGRAM

Data analysis, in this chapter, aims at describing the variation of understanding of a phenomenon, learning to program, found in a group of pre-service and in-service teachers. Hence research question 1 is addressed in this chapter. The process involves formulating the essence of understanding as categories of description. It is important to state that the analysis is on a collective level, and not individual students' understandings. For each category of description, those aspects of learning to program (dimensions of variation) which differ in each category are found. In order to achieve this level of analysis a rigorous, iterative approach was required involving different sources of data as input.

Initial analysis of the data involved becoming familiar with the interview transcripts and the journal entries. Each transcript was read and re-read numerous times in an attempt to reveal differences in the data to which Marton and Booth (1997) refer as "a pool of meaning". During the initial reading of the in-service teachers' journals, notes were made with regard to commonalities in the teachers' experiences. Relevant quotes to the common categories were then extracted from the transcripts and journal entries. These quotes were then analysed for their meaning. This served as a means of decontextualising the quotations from the individual respondent. Analysing the quotes for their meaning was used further to clarify and refine categories, and to reveal structural differences (how students experience the phenomenon) between the categories. While the variation in the way the act of learning to program experienced by students was being analysed and seemed to be the focus, it was becoming clearer in the analysis of the journals and transcripts, that problem solving was the key issue in learning to program.

5.1 Ways of experiencing the act of learning to program

Illustrative data analysis of how the students experience learning to program

The categories of description that emerged from analysis of the data are described below. Direct quotations[1] from interviews and journals are provided to illustrate the described features of the categories. Before the categories of description of experience are described, some important terminology with regard to phenomenography will be discussed to enable the reader to better understand the experiences of learning to program.

Phenomenography is based on Marton and Booth's (1997) understanding of learning. An experience has a *structural* aspect and a *referential* (meaning) aspect. The structural aspect of a way of experiencing something is twofold: discernment of the whole from the context on the one hand, and discernment of the parts and their relationships within the whole, on the other. The aspects that surround the phenomenon (in this case, learning to program) are referred to as its *external horizon*. The parts and their relationships, together with the contours of the phenomenon, are referred to as its *internal horizon*. The referential aspect is the overall meaning which is "intimately intertwined with the structural aspect of the experience."

In the analysis presented in this chapter, the names given to the categories of ways of experiencing learning to program refer to their referential aspects.

I extend figure 3.2 and obtain figure 5.1 to illustrate a unit (category of description) of experience of learning to program relevant to this study.

[1] The grammatical errors made in the transcriptions are acknowledged. I have not altered any part of the transcribed statements in order that they may reflect the students' truthful comments.

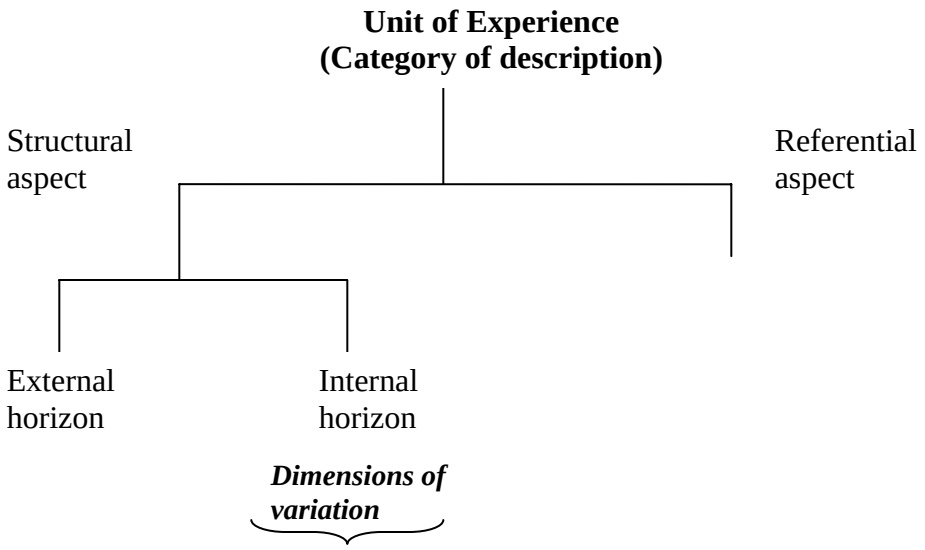


Figure 5.1 Structure of a unit of experience

Five categories of description were created, namely: meeting the requirements, learning the syntax / learning by comparison, understanding and assimilation, problem solving and programming in the large. In the sections that follow, the referential and structural aspects of each category will be described, together with the dimensions of variability.

5.1.1 Category 1: Meeting the requirements

Referential Aspect

Students whose experience is classified into this category are those who are acutely aware of what must be done in order to get through the particular unit or module, e.g. making sure that the assignments are done and submitted on time. It seems the motivating factor for them is to get credit for the course. Those aspects of the course that will allow students to generate marks become the focus of the learning activity. Feedback is sought from the teaching staff, particularly to indicate how well they have achieved in the task at hand.
The following quotes from students' journals illustrate the approach to learning to program taken by people whose experience falls into this category:

> While examining the activities that's required, we are not guided as to the depth that we should explain the activities, amount of pages needed? I really enjoyed the first assignment and hope to get 100%
>
> ….
>
> I am aware that I have to do these problems and that I have a goal in mind to complete this Unisa assignment with success. [in-service teacher]
>
> ---
>
> I'm worried that I am behind the schedule in the proposed timetable provided for us as a guide to complete all the work. Although I work almost every day I am still behind. However, I do make my own notes in each section. Perhaps this is taking time.
>
> So far I managed to meet the deadlines for the assignments. However, I find that there is a long period between the date of the last assignment and the examination date. Perhaps we should be given more time to go through the sections and complete the assignments [in-service teacher]
>
> ---
>
> I can't believe today is our last lecture and it's up to me to pass the exam. [pre-service teacher]

There are some students, both in-service and pre-service teachers, who see the course as a means to an end. In other words these students see the acquisition of a certificate (that indicates that they have knowledge of Java programming) as all that matters. Meeting the requirements of the course fulfils this goal.

Structural Aspect

External horizon

The act of learning is seen within the bounds of the learning institution or formal learning environment. In the case of the pre-service teachers this formal learning environment includes the attendance of lectures, and issues such as achievement in tests and assignments, and feedback from lecturers forms the backbone of the learning environment.
In the case of the in-service teachers, the act of learning is focused on the course requirements, which include the submission of, and achievements in, assignments, and feedback from the assignments and tutorial letters.

Internal horizon

Students focus simultaneously on the structure of the course and the tasks that need to be handed in for assessment and feedback. The score obtained form the focus. The structure of the course seems to play a role in the way in which the student learns to program.

Dimensions of variation

Learning approaches

Students approach learning at a surface level when they experience learning to program in this category. The course content is seen simply as material to be learnt for the exam. The activities that require completion by the course are pursued actively in order to meet the requirements of the course, even if it means that some aspects of the course are not well understood.

Learning the programming language

In-service teachers that have prior programming experience understand learning the programming language as learning the syntax of a new language. Ordinarily, this would be what is expected of in-service students, given that they have a grounding of problem solving in another programming language. However, it must be stated that generic problem solving skills are applicable to "programming in the small"[1]; the kind of complexity that can occur within a single subroutine. On this level, complexity is provided by control structures. The two types of control structures (loop and branches) can be used to repeat a sequence of statements, and to choose among two or more possible courses of action respectively. In a study conducted by Wiedenbeck and Ramalingam (1999), they found that there were no differences between the two cohorts of students studying procedural and object-oriented programming respectively, for short programs (consisting of a single class with encapsulation): programming in the small. Programming in the large requires a different approach to problem solving from the procedural approach.

[1] Refer to chapter 2 for a discussion of "programming in the small."

Pre-service teachers, on the other hand, see learning to program as following the course doggedly as a means to an end, which is attaining the course certificate, as opposed to seeking an understanding of programming.

Learning motivation

As far as the pre-service students are concerned, the motivation is to get through the course. This is illustrated by the following pre-service teacher's journal entry quote:

> I guess ….. is really going to get his goat… stressing to us once again that we can't afford to miss one single lecture because of the nature of the stuff we're learning …amen to that… I can't afford to even come late once…I really need to succeed with this course.

The in-service students are motivated to learn the new language so that they do not get "left behind". They are driven by the fact that they will have to learn the new language in order to teach it in 2006, even if it means that they do not have a deep understanding of the language.

5.1.2 Category 2: Learning the syntax (Learning by comparison)

Referential Aspect

Learning the rules of the constructs, such as the declaration of variables, writing the different loops, use of methods (subroutine) and so on, is what characterizes this category. Since Java is perceived to be a new, "hip" and popular programming language, the importance of coding is a priority for the students learning Java whose experience falls into this category. If simple constructs can be coded correctly, then the goal has been achieved. Learning the syntax of the language is a high priority. Some students (particularly in-service teachers) are over-confident in their ability to learn and use the new language. They are learning by comparing a known programming language (in most cases, Pascal) and Java. The following quotations from in-service teachers' journals illustrate this category:

> Because I teach Pascal and Delphi, my understanding of new syntax is always in terms of "What is different, and what remains the same.
>
> ----------

This is really simple as there are no major differences in data types compared to Delphi, having previous programming knowledge is a big plus in doing this course.

Note: difference between the while...and the do…while. The do… while is similar to the repeat…until in Pascal/Delphi.

Length is just like length in Pascal character handling.

The section on variables was easy to understand. It is similar to Turbo Pascal except that a real variable is declared as type double and the syntax is slightly different. Exercise 4B was exciting to work with. I was very pleased to see that I could answer the exercise quite easily.

I realized that the getChar method in Java is similar to the readkey character handling function in Pascal. They both allow the user to type in a single character. In Pascal, the readkey will read a key without displaying the input on the screen. In Java, the getChar method allows the character to be type in without having to press the <enter key>.

After working through chapters 1 and 2 of "exploring Java-grade 10", I find that I can easily relate programming in Java to programming in Turbo Pascal….. I also know that Java is case sensitive unlike Turbo Pascal.

I find units 1 and 2 straight forward – basic to programming. I try to make comparisons to Pascal with regards to syntax. Sometimes the brain is still functioning in Pascal level and you realize you are working in Java.

The above quotations are from seasoned teachers who have a strong Pascal background. The in-service students see learning to program in this category as learning only the syntax of the new language, with the notion that all other aspects of Pascal and Java are the same. They seem to think that only the syntax of the new language is required in order to transfer their existing problem solving skills to the new programming environment.

The above quotations indicate a false sense of confidence in object-oriented programming, because many of them have not realized the full potential of object-oriented programming. However, at this stage of learning, their comparison may be a realistic assessment of learning the new language, from *their perspective*. They are still seeing programming as "programming in the small." The following quotes from the journals of pre-service teachers illustrate this category:

> The first thing I have to mention about programming is that you really have to type a hell of a lot to achieve so little as the end result… no matter I'm glad my program ran eventually… the other thing I grasped was that Java is case-sensitive…
>
> ---

> If you had asked me last semester what syntax and methods were I would never have been able to tell you. The syntax of Java is definitely something to get used to because you don't type what you say in English. You have to convert your language into JAVA language and although this seemed as familiar to me as German in the beginning, I am getting more comfortable with it and learning to use it better. I had no idea what a program was, never mind methods, looked like so this is all completely new to me.
>
> ---

These students see learning the syntax of a programming language as central to learning to program, while it is not to say that it is not essential, it is not the only way to create a working program. As discussed in section 2.1, the syntax is only part of the programming environment. This kind of experience may involve rote learning.

Structural Aspect

External horizon

The act of learning to program in this way is seen within the bounds of the Java programming language syntax. In other words, getting to know the programming language is seen as the ultimate aim. In-service teachers see the Java language syntax as re-skilling themselves in order to be relevant in the wake of the new curriculum.

Internal horizon

Students experiencing learning to program in this way focus on the syntax, the coding task and the IDE (Integrated Development Environment). Practice seems to be the key point with which they engage, in order to become better at coding. The way in which the component parts (syntax, task, IDE) relate to one another forms the internal horizon. The following quotes from in-service teachers illustrate this structure.

> Fortunately, due to experience in Pascal this is quite interesting as the concepts are the same but the syntax is different. Can be quite confusing with different syntax.
>
> ….
>
> There are a lot of things I have not written down because the programming concepts are very similar to Pascal and the rules may change but the idea is still there.
>
> ---

> "The content of the units is fine. I have confidence in my programming ability so am not finding too much difficulty with concepts. This will be my 3rd language so it's just a case of learning new syntax. Practice is essential."
>
> ----------

The above quotation indicates that the student sees the general programming principles as the same in both the languages; however, the student is still programming in the small, where simple principles such as variable declaration, assignment statement, Boolean operators for comparison purposes and so on are similar. The student has not reached the level of discerning the distinction between OO design principles and traditional design principles as alluded to in chapter 2. The students realize that getting used to the new syntax necessitates practice.

Dimensions of Variation

Learning approaches

When experiencing learning to program as *learning the syntax*, students focus on the syntax of each construct and compare it with that of the language in which they are experienced, in this case, Pascal. The approach used is one of comparison. This is particularly characteristic of most[1] of the in-service teachers. This way of learning leans towards a surface approach.

Learning the programming language

Learning to program is seen as learning the syntax of the language. The understanding of learning the programming language in this way influences the learning activities and approaches used by the student. In learning the language, focus is simultaneously on object creation, classes, public and static methods. The following quote, from the journal of a pre-service teacher, illustrates this focus:

> But what I'm really struggling to get to grips with is the whole classes and public story…I asked Mr… the explanation but I didn't want to look silly the second time around…anyway I'm sure I'll get it next time…

[1] There are some in-service teachers who are learning to program for the first time; their issues are large

Learning motivation

Students are motivated by comparing the new language with the syntax of a known language such as Pascal. Perseverance and long hours of work are features in this category of learning. Note-taking for memorization is a key aspect of learning in this way as well. Seeing their program compile and run motivates the students. A quote from the journal of a pre-service teacher illustrates this motivation.

> Today we went through all those masses of questions we had to do during the holidays. I am still finding programming very challenging but it is such an accomplishment when it eventually runs.

After a number of attempts at possibly correcting syntax errors (indicated by the quote "*when it eventually runs*"), the program runs; this gives students a sense of knowing that they are learning.

5.1.3 Category 3: Understanding and Assimilating

Referential aspect

The act of learning to program is seen as learning through understanding and assimilating the concepts involved. When going about learning to program in this way, students see understanding as integral to learning. It is not enough to type in the code and 'see if it works', rather these students seek to understand what they have done in order to affect the outcome. These students may view learning to program as building on prior experience, involving a developing sequence of concepts. They may struggle to understand one concept before moving on to the next, although sometimes they feel the need to progress through the course is more important and so persevere without the sense of understanding they seek.

As an example of students whose experience falls into this category, consider the following strategy that was given by an in-service teacher who was asked to study a given program and was asked why the program prints nothing at all when it is meant to produce the output:

54321
4321
321
21
1

Here is the student's reasoning.

> In the first For statement "row < 0" should be "row > 0" and in the second For statement "num < 0" should be "num > 0". This was easy to spot since the output produces numbers higher than zero.
>
> After running the program I got the following output:
>
> **Enter the size of the number triangle: 5**
> **543214321321211**
>
> I then made the statement "c.println();" part of the first For statement, by inserting opening and closing brackets. The following code is free from errors:

```
c.print ("Enter the size of the number triangle: ");
int size = c.readInt ();
for (int row = size ; row > 0 ; row--)
{               // Opening brackets: Start For…
   for (int num = row ; num > 0 ; num--)
     c.print (num);
   c.println ();
   }            // Closing brackets: End For…   "
```

The student is able to link the Boolean expressions contained in the "*for*" loops with the *c.print(num)* statement and compared it with the expected output given. He realized that the numbers have to be greater than 0. Therefore, he made those changes to the comparison operators. He then further realized that the *c.println()* statement has to be part of the outer *for* loop so that each sequence of numbers is output (printed) on a new line. This suggests an understanding of the individual code statements and the assimilation of the code (program) statements collectively. The quote below from the same student demonstrates this understanding and assimilation.

> The 'switch and case' was easy to understand and enjoyable to work with. Number 2 from exercise 7H took me long to understand and interpret. (Tutorial letter 101 helped me with this question). When I eventually solved this problem I was excited and on top of the world. Some of the problems up to this stage took me a long time to solve and I think that my 'never say die' attitude helped me to achieve the correct solutions.

The above quotations reveal a clear pattern of reasoning that indicates commitment to understanding. The following quote, from the journal of a pre-service teacher, illustrates this category of learning.

> The exercises of page 3.11 were in such a way okay. I kept on referring to the examples on how to tackle the ones that were in the exercise. It took me quite a long time to complete because I wanted to understand what was really required of me. However there was a single program that I wrote or modified where I did not refer to examples on the study guide. When it was marked in class there was only a single error and that made me feel good and I said to myself that I was beginning to master this...

The pre-service teacher's quotation above clearly illustrates that the student is also experiencing learning to program as understanding and assimiliating the concepts required to write a program. Although there is no detail with regard to the concepts used, it is understandable that the pre-service student is still not confident enough in the language to express himself using the technical features of the programming language.

Structural aspect

External horizon

The concepts and principles that underlie programming are seen as the boundary, which is not limited to the language under study, or to a single program, but to the idea of programs and programming in a broader sense. The student might be aware of techniques and / or concepts from his/her previous programming experience that can be brought to the present learning situation.

Internal horizon

Students experiencing learning to program in this way are focusing simultaneously on the task, tracing through the code and the understanding of concepts. The experiences described are in terms of gaining insight from doing tasks apart from those set as part of the course.

Dimensions of Variation

Learning approach

In experiencing learning to program as the means of bringing about understanding and assimilation, it is tracing through programs that have been written by others that are required. A deep approach to learning is adopted in this category of experience. The following quote from the journal of an in-service teacher illustrates a deep approach:

> Passing variables gets a bit confusing. It seems to complicate things.
> I misread the Card program and created a randomized value selection for the card after the user chooses the suit. It worked out fine except I didn't use the constructor to pass value & suit so it got dumped and redone. I thought my little program was quite sweet though.
>
> ---
>
> Units 5 and 6 were difficult to understand. I did not really grasp all the study material presented. I am going to do all the exercises with another student so as to understand the concepts covered… I do understand the concepts of procedures with parameter passing but I still can't grasp this concept in JAVA.

In the first instance the student is able to diagnose her problems with regard to the Card problem given as an assignment. She realizes the importance of the constructor in creating and initializing the values of the instance variables *value* and *suit*. The student is able to make connections and is able to focus on the central concept of constructors in order to diagnose the problem. This is characteristic of a deep approach to learning.

In the second quote, there is the commitment to understand the material contained in units 5 and 6 of the study guide, which deals with writing classes in Java and aspects of object-oriented programming that include passing of parameters in Java. The student even makes reference to eliciting support from a peer ("*I am going to do all the exercises with another student so as to understand the concepts covered.…*"). The implied confidence gained from the peer support is important, especially in light of the course being a distance learning course. The quote suggests that the student has an intrinsic curiosity in the subject matter. A *deep approach* to learning is encouraged by the student.

Learning the Programming language

The language, as well as the understanding of the concepts, and how they relate to one another, becomes the focus of the experience. The desire to understand the structure of the language calls for a focus on code; however, the code may be seen from a broader perspective than in category 2. Learning the language of the program makes possible the reading and understanding of code. The next quote, from the journal of an in-service teacher, illustrates the focus of learning the programming language in this category.

> I always had trouble myself with understanding the concepts of objects, methods and classes. Objects and methods became clear when working a lot with Delphi. Classes are like creating a TYPE or RECORD in Pascal?? I think."
> Constructors: what?
> Ok – I sort of get it. It seems like a lot of information is given here, but I am finding it hard to organize the info in a structured way in my head. I need to try and draw up a diagram or something to put all these terms together. I keep falling back to trying to compare new Java things with older concepts I've learned in other languages. Maybe I haven't made a proper mind shift and therefore I struggle with the classes thin[g].

Learning motivation

Students are intrinsically motivated. The challenge of understanding this novel aspect of learning to program is stimulating. Success in attaining the correct output of the program motivates students to study further. The following quote, from an in-service teacher's journal, shows this motivation:

> I ask other computer studies teachers, I have a lot of friends working in the private sector and my brother is a programmer in C++ & Java. I always make sure that I find the solution or error to be able to get back to a student's problem. I often show learners how to use the step/trace debugging facility in Pascal. It helps them and me to find errors in code not traced by executing the program. I might ask other learners in class to try and find the error (usually in this case – I have found the answer, but want others to be able to identify it as well).
> I will tell learners to use the help file to assist in parameter orders etc. I often give them books where they can read up on how to do certain things that their own text books do not cover. I do this especially when they need help with the graph unit.

Students (in-service teachers) indicated their motivation by their resourcefulness, which is in turn encouraged by them. From the above quote, it is possible to see that the learner views programming in a broader sense than just learning to program in Java. The teacher's quote is with regard to teaching Pascal, yet he is able to enlist help from his brother who is

a programmer in C++ and Java. Another quote, from an in-service teacher's journal below, exhibits the intrinsic motivation of learning to program in this category.

> Having had no previous experience in any programming language except a keen interest in Computers and reading up a little on programming in a computer magazine, I must admit that the first few weeks were extremely challenging to me. Trying to understand the different concepts was an exhilarating experience and took me back to the time I had been studying mathematics years ago.

Even for a novice in-service teacher, the educator is motivated to go on learning to program. His keen interest in computers has helped to motivate him. The statement, "To understand the different concepts, was an exhilarating experience", illustrates an intrinsic motivation and can only be possible if a deep approach to learning is assumed.

5.1.4 Category 4: Problem Solving

Referential aspect

In this category learning to program is experienced as learning to solve problems. When going about learning to program in this way, the student begins with a problem and sets out to discover the means to solve that problem. The understanding that is sought in category 3 is a fundamental component of this category. It does seem that this is one of the more difficult aspects that students experience. With the change from a procedural language to an object-oriented language, this category seems to pose an even greater challenge, especially for the in-service students. The following quote, from an in-service teacher's journal, illustrates this challenge:

> Exercise 7E- At first I really battled with the question. After I spoke to Isabel I realized that I was reading the question completely wrong. They didn't expect you to alphabetize all the entries but only to find the first and the last!! How stupid of me!! I always tell my learners that they lose marks because they do not read the question properly and here I go and do the same thing. Now I can really identify with those "stupid" learners. I feel small now!!!
>
> ---
>
> Problem solving is a process, it includes:
> Planning
> Writing test cases
> Implementation
> Remember; thinking and planning of an algorithm should take 90% of the time and coding only 10%.

I found that most learners especially grade 10 don't plan out their solutions, they are very eager and just want to start typing out their solutions. I try to teach the learners that for every program that they are given they must analyze the problem, outline the requirements and then design the steps to solve the problem. Learners are not allowed to type in their solutions until they have written solutions.

From the above quotation, it may be inferred that novice students find it more natural to tackle a problem in a single step, rather than stepping back from the problem and looking at it more strategically.

The following quotation is an example of a response to the experience with problem solving, based on the questions given as an activity in the journals of in-service teachers:

Do you find it a challenge to know how to start solving a particular problem?

> In some cases it is a challenge. When I am solving a problem, I think about it very logically and break up the steps in my mind. If I'm unsure as to the solution, I take simple test values and see how to solve using these. I make sure each step or method works properly before moving on to the next.

Having a good knowledge of the language constructs does not necessarily mean that one is able to solve a problem. Do you find a similar situation in your class of learners?

> Yes. I find that learners use the language inappropriately. Instead of adapting, they regurgitate code as per the text book. This is a real problem as they are totally confused if the questions are phrased even slightly different to that of their notes.

Do you always have to break down the problems into parts for your learners? Do your learners need a kick-start when given a problem to solve?

> I do. I normally get them to write procedures instead of an entire program. I usually give them the declaration and expect them to use local variables thereafter. They do need hints otherwise they are at a loss!

The above quotations serve to illustrate the perception of learning to program as problem solving and the challenge it poses. The quotations do not necessarily indicate the approach to solving problems at this stage.

Structural aspect

External horizon

The act of learning to program is seen in the realm of commercial programs, and the concepts that underlie programming form the external horizon. The achievement of a working program to solve a problem within the programming environment makes up the external horizon.

Internal horizon

Students experiencing learning to program in this way are focusing simultaneously on the problem to be solved, understanding the concepts, and the method or technique they were taught to solve problems. The following quote, from the journal of an in-service teacher, displays this method:

> Making myself thoroughly familiar with the 4 steps in solving problems in an object-oriented approach, contributed greatly in my successful arrival of most of my solutions in assignments 4 and 5. To re-iterate they include the following:
> 1. Planning the class (data members and methods)
> 2. Test by writing a main method
> 3. Write the methods for those planned in step 1
> 4. Test the class by running the main method.
>
> ---

Dimension of variation

Learning approaches

The problem is seen as a starting point. The approach followed by some students is, understanding the problem and determining the inputs, the processing involved and the outputs required, while others used a trial and error method. A quote, from an in-service teacher's journal, exhibits one of the methods used:

> Trial and error is one of the methods used when I program. Keep trying until the correct solution is obtained. If I sill cannot solve the problem, I try using values and trace tables to get a clear picture of the program and why it is not working.
> If I do not know certain commands, I use the help facility.
> Group work is also a good idea for discussion on solving problems. Many minds and idea put together can come up with the solution.
> Sometimes taking a break helps clear the mind and the solution to the problem is found.

By using the trial and error method, the student is not interpreting the problem as such but is focusing on the program that has to be written, with no specification in mind. While this characterizes a surface approach, Marton and Booth (1997) refer to this as a constructional approach. The quotes below, from the journals of in-service teachers, display other methods which were learnt as a technique to solve problems:

> First understand the problem → what is required.
> Remember the different programming structures and syntaxes in Java.(compared to Pascal)
> Use the IPO (principal) which stands for Input Process Output and use a pseudocode or flowchart as an algorithm to solve the problem.
> Use the KISS principal also – Keep It Simple, that is use simple structure and technique.
> Write the program on paper before typing the program.
> Use test value to verify the program output.

For convenience the following quote will be repeated:

> Problem solving is a process, it includes:
> Planning
> Writing test cases
> Implementation
> Remember; thinking and planning of an algorithm should take 90% of the time and coding only 10%.

The above quotes are characteristic of a deep approach, in which meaning and understanding are sought. The quote (from the journal entry above), which I will repeat below, demonstrates that although the above process is known, the process is not always followed.

> I found that most learners especially grade 10 don't plan out their solutions, they are very eager and just want to start typing out their solutions. I try to teach the learners that for every program that they are given they must analyze the problem, outline the requirements and then design the steps to solve the problem. Learners are not allowed to type in their solutions until they have written solutions.

When experiencing learning to program in this way, there is the acknowledgement that planning is important prior to coding. However, there seems to be tension between the desire to solve the problem in this way, and wanting to code almost immediately in a "hacker's way", which tends to ignore software engineering principles. The above quotations seem to indicate a commitment to a deep level of learning approach, which seems to be encouraged in high school classes by in-service teachers.

Learning the programming language

Learning the programming language is seen as a tool to solving the problem. Students see the importance of mastering the language as a prerequisite to solving the problem.

In the interviews conducted with the pre-service teachers, some of the responses to the questions: "*What do you understand by programming?*" and "*What do you perceive programming to be?"*, are given below.

Quotes, from pre-service teachers' transcripts of interviews, are presented below. The "R" refers to the researcher and S1-S7 refers to the students.

S1: Well I wouldn't say too much. Actually what programming is about ... I think it's the way things are made simpler and implemented … To find easier ways to do things.

S2: Basically its problem solving.

S5: Programming is… a for example you got a question and you need to have a program in which you can get answer for the question... Ja
R: So that is how you perceive programming, an answer to a problem
S5: Ja in which somebody can use it

S6: The first thing that comes to my mind… is the idea of a problem and programming is the way in which we go about solving a particular problem…We use lots and lots of tools… use various tools to do that.

The above quotations from the pre-service students indicate that programming is perceived as problem solving. This is an important view and one that is supported by several studies (Palumbo, 1990a; Liao & Bright, 1991; Rucinski, 1991; Choi & Repman, 1993; Thomas & Sylvester, 1996 and Deek, 1999).

Learning motivation

Having experienced the success of running a program motivates students to want to continue solving problems in this way. The achievement of a working program serves as motivation to learn. The response to the question, "Do you enjoy programming?" and further questions asked of the pre-service students are given below:

S4: *Yes*
R: *What makes it enjoyable?*
S4: *Solving the problems and getting it right. My favourite... I like working out sequences. Like when we get a number like 4, 16, you work out the 12 in the sequence of numbers*
R: *You mean the pattern in a series of numbers?*
S4: *Yes.*

S2: *Yes I do*
R: *What makes it enjoyable?*
S2: *It's challenging, it tackles your mind, and it makes you think. The fact that your program is running, makes you feel good it makes you think that you can do something, therefore I feel challenged.*

S1: *Yes I do because it demands a lot of thinking and when you've thought of something and solved the problem you feel good about yourself. You feel that you're a great thinker.*

S5: *I do at times if I get to the answer then I'm happy but if I don't You get so confused if you don't get to the answer, because you don't know really where the problem is. You find that always you think your answer is right and if it doesn't run you get so confused.*
R: *Does it not motivate you to get down and find out what went wrong?*
S5: *Yes it does. Because for instance when we were doing the corrections it's like 'oh I went wrong there' and you can correct it for the next time.*

R: *What makes it enjoyable? You say you do enjoy it.*

S5: *It's challenging and it gets you to think...I don't know how to put it...to think, .at that time this thing you want to do now.*

The dimension of variation, *learning motivation*, is seen to be similar to that of category 2 (learning the syntax). However, the motivation in this category of experience is enhanced by the conceptual understanding behind the working program. In other words, the focus is not the code/syntax itself, but on coding as a means to achieving understanding and a solution to a problem.

Programming is synonymously known as solving problems by many students and this notion is reflected in the literature reviewed (see chapter 2). Therefore, how students go about programming (solving problems by means of a program) will be further analyzed in the next chapter.

5.1.5 Category 5: Programming in the large

Referential aspect

Learning to program is seen as creating a realistic product (that is, a working product that can be used by an individual or company) and experienced as learning what it takes to be part of the programming community. The following quotation, from pre-service transcripts of the interview, demonstrates this point:

R: *When we talk of programming, what comes to your mind? How do you see programming or learning to program?*

S7: *I feel that it's finding a niche, like a market in a programming atmosphere where somebody hasn't already developed a program to do a certain thing and that you will fill that…that gap and maybe design a program to do something that somebody needs; or even upgrading programs to make them… you know further with technology.*

R: *Do you think it has anything to do with problem solving?*

S7: *Oh sure, definitely, because you have to first, have a problem to solve it and then program it.*

R: *So that is how you perceive programming, an answer to a problem*

S5: *Ja in which somebody can use it*

Structural aspect

External horizon

The creation of a complex program or project, which is a realistic product that can satisfy a customer/client in the application environment, forms the external environment.

Internal horizon

When learning to program in this category, the working product, coding, understanding concepts and design principles are simultaneously focused upon. These aspects of programming form the internal horizon.

Dimensions of variation

Learning approach

This fifth way of understanding programming is conceptual, rather than merely syntactic or semantic (as the first two categories were). It relates to the conceptual framework and is an abstraction of the problem solving and understanding and assimilation that were the focus of the previous two categories. Linking the course content to real life problems, and recognizing the higher level of thinking that is required of programming, is indicative of a deep learning approach.

Programming language

The students are not particularly concerned with what language they learn as long as it is current and allows one to create a realistic product (a program that can be used by a real customer or client). This notion is evident in the quote below, from an in-service teacher's journal:

> Solving problems in general, I find it interesting especially when the problem is associated with reality and current affairs example – working a problem associated with lights and water etc. I sometimes like to bring reality problems into the classroom for the pupils to solve. If the school is having a prom-the pupils will be given a problem associated with the school's prom.

Learning motivation

The realistic, working end-product is what the learners wish to achieve. The internal horizon, again (as in category 4, problem solving) includes the solution to a problem and the programming language, but in addition see the relationship between them in terms of a whole working product. The students are encouraged by their intrinsic curiosity and they are motivated by the fact that somebody can use the program to meet his/her needs. This motivation is highlighted by the quotes, "*... It's challenging, it tackles your mind; it makes you think…The fact that your program is running, makes you feel good it makes you think that you can do something, therefore I feel challenged*" and "*Yes in which somebody can use it*".

The student looks beyond the learning institution and the programming language to understand what is involved in the application environment. Very few students reached this category of experience. It is possibly critical for these students to become capable of experiencing learning to program in terms of 'programming in the large', if the students are to develop into reflective computer programmers, rather than routine-bound producers of 'toy' programs.

The findings of research question 1 are summarized in Table 5.1.

Category of description	Referential aspect	Structural aspect				
		External horizon	Dimensions of Variation			Internal horizon
			Learning approaches	***programming language***	***Motivation***	
Meeting the requirements	Learning to program is doing the prescribed work that enables them to get through the course	The learning institution/or learning environment	Oriented towards a surface approach	*In-service*: see the language as another programming tool *Pre-service*: not evident in the data	Need to pass the course	• Structure of course • Essential tasks • Feedback of score
Learning syntax/learning by comparison	*Pre-service*: learning the rules of the construct for coding *In-service*: learning the new language by constant comparison to the known language	The programming language	Leans towards a surface approach	*In-service*: see the language as another means to arrive at their existing level of programming. *Pre-service*: sees learning the syntax and the structure of the classes and objects.	Want to see some change in their work (e.g. compiling and executing the program)	• Syntax • The coding task • IDE
Understanding and Assimilation	Learning to read and write a program through conceptualizing the principles	Programs/programming	Orientation towards a deep approach	Seeks to understand the language and concepts and how they relate to each other	Motivated by the challenge of understanding novel aspect. Intrinsically motivated.	• Task • Tracing through code • Meaning of concepts and constructs
Problem Solving	Learning to solve problems	The programming / problem task	Leaning towards a deep approach, although trial and error is a strategy used sometimes	Language is a tool to be used to solve the problem	Motivated by the working program that solves a problem	• Problem task • Understanding • Method/technique to be used
Programming in the Large	Creating a realistic product	The application environment	Deep approach indicates an active learning strategy	Wants to create a working product irrespective of the language used	Wants to program usefully and the working product gives a sense of fulfillment	• The working product • Understanding concepts and design principles

Table 5.1 Summary of findings of Research Question1

In summary, the two groups of students, namely the in-service and pre-service teachers, experienced learning to program in the following categories:
In-service: meeting the requirements, learning by comparison, understanding and assimilating, problem solving, and programming in the large.
Pre-service: meeting the requirements, learning the syntax, understanding and assimilating, problem solving, and programming in the large.

Descriptions of the five ways of experiencing learning to program identified in this study empower the teacher to address difficulties students have with programming. They offer a framework within which the teacher can identify what aspects of the total understanding are essential for adequate future development.

5.2 Outcome Space

The outcome space represents the relationship between the different ways of experiencing the phenomenon of interest, namely, learning to program. The outcome space, depicted graphically in Figure 5.2, indicates an evolving awareness by the researcher as the different categories are represented. Those who experienced learning to program in the outer categories had to learn or develop the cognitive accomplishments in the inner categories. This can be compared to Linn's (1985) proposal of the chain of cognitive accomplishments (discussed in chapter 2).

The five ways of experiencing (understanding) learning to program become successively more complete, moving toward the underlying aim of programming. One can infer, from the hierarchical nature of these results, that the development of more sophisticated categories of descriptions does not mean that the preceding levels are relinquished. When an individual shows evidence of holding a particular conception (or experiences programming in a particular way), it can be assumed that he or she also holds conceptions of learning to program that fall below this level. In other words, students who hold a high-level conception of learning to program also have access to the preceding lower levels. By

contrast, students who hold low-level conceptions of learning do not have access, at *this* point in their learning, to higher level conceptions of learning. Some of the students, however, had not come beyond the first category and more had not come beyond the second category at this point in their learning.

It was not possible to determine the exact number of students that fell into each category. In respect of the in-service teachers, not all teachers submitted both the journals for assessment. Therefore, keeping an audit of each teacher's account throughout the year was difficult. Empirically, too, many students' responses indicated a way of "learning to program" encompassing two or more categories (excluding the "meeting the requirements" category). In these cases, their conceptions were classified into the "highest" of the categories indicated. Finally, although the merit of the conception was not used to determine the hierarchy, the higher categories can be seen to express more educationally desirable conceptions of learning to program, than the lower ones.

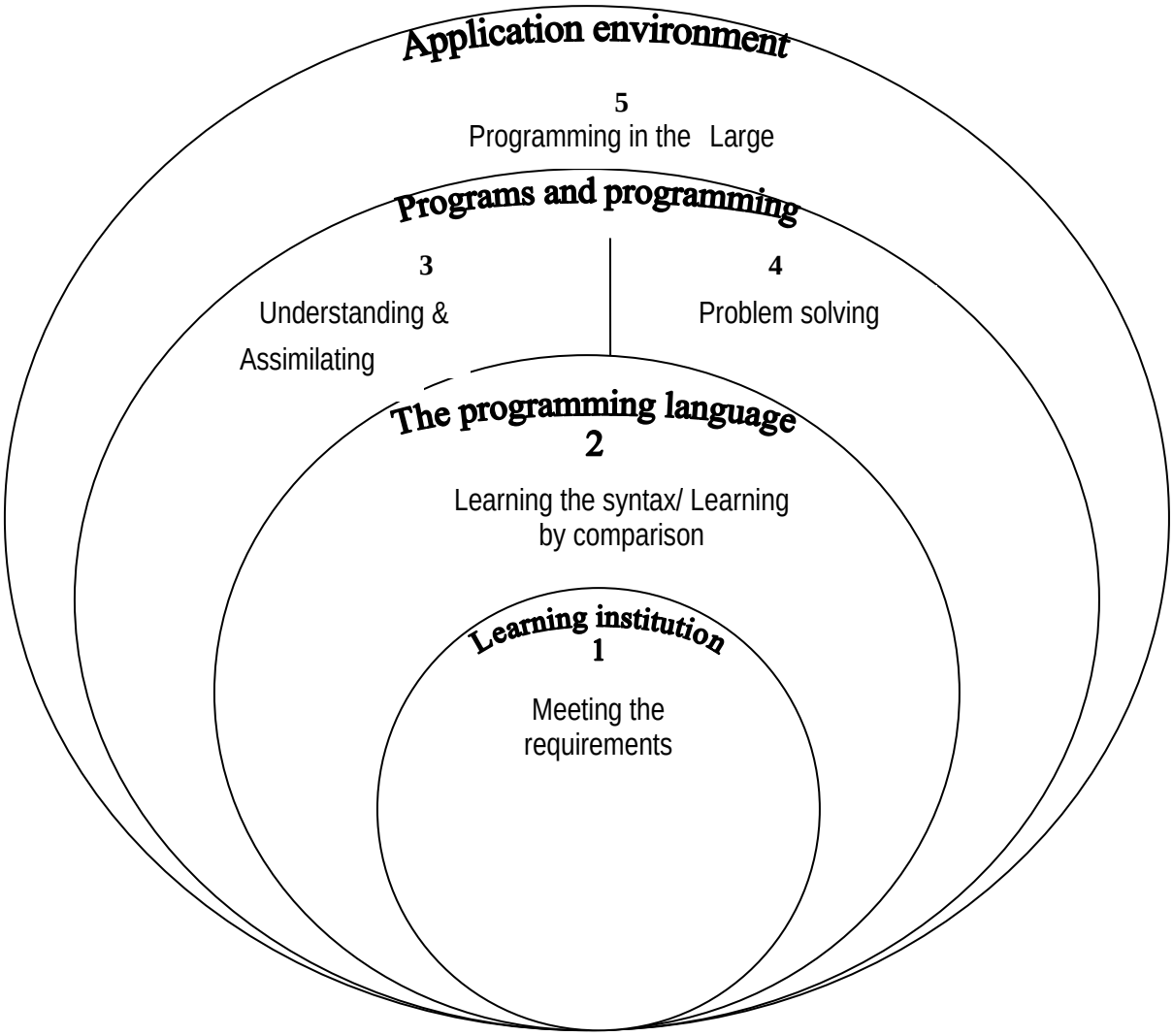


Figure 5.2 Outcome Space depicting the expanding horizon of the experience

5.3 Further analysis

Categories 3 and 4 appear on the same level of the outcome space. Both categories adopt a deep approach to learning and the external horizon (context) is the same for both categories. The activities are related in that, during generation of a program (problem solving), the development and debugging of code necessarily involves reviewing and understanding it. One would, therefore, expect there to be a high level of correlation between the two categories. However, this is not so as studies have shown that this correlation is not necessarily evident in students' attempts of learning to program

internationally (Winslow, 1996). Observations of students and their assessment reveal results that are different, and which will be discussed in chapter 8.

6 DATA ANALYSIS: HOW DO STUDENTS PROGRAM AND SOLVE PROBLEMS?

The aim of this chapter is to present the results of research question 2. As discussed in chapter 4, the results were obtained through an analysis strongly informed by a phenomenographic research perspective. This analysis will have, as its outcome, the different ways in which programming (solving problems) is experienced.

Seven pre-service students were interviewed after they had written a problem solving test in programming for the main study, and five students were interviewed for the pilot study (Appendix D).

6.1 Description of the four "steps" evident in the students' programming (problem solving process)

Four steps of problem solving were identified, which were regarded as constitutive of the students' problem solving strategies. These were (i) scanning; (ii) planning; (iii) translation; (iv) re-interpretation.

(i) Description of the scanning category

This step can be compared to what Laurillard (1984) describes as the students' "initial approach" to the problem. It represents one of the many stages and phases that students go through in trying to discover the essence of what the problem is about. This step, in a way, alludes to how students describe and interpret the problem at hand, through the process of reading and figuring out

the problem, thus allowing glimpses into the different aspects that inform the strategies that students use. Scanning essentially entails the extraction of the familiar and main features perceived to be the defining characteristics of the problem. The following considerations, pertaining to the students' constitution of relevance structure, were seen to be central to this process:

- What are possible sources of difficulty in the problem?
- What is perceived as the ultimate objective of the problem?
- What knowledge is required to make sense of the problem?

(ii) Description of the planning category

This category consists of a starting cue, a direction, a level, and a type of link to explore next.

(iii) Description of the translation category

Translation refers to the students' transformation of the problem statement (its perceived structure of relevance: significant constructs) into a physical representation of program code. The category encompasses four of the heuristic steps to problem solving (see Schoenfeld, 1978): analyzing the problem, exploring the information to be used, planning the solution and executing the plan. It is in this category that descriptions and interpretations of the problem are actualized.

(iv) Description of the re-interpretation category

The research-created interview contexts lend themselves to this step, in that they provide the students with the opportunity to confront themselves with unanswered questions, concepts or misunderstandings. In other words, students are given an opportunity to explain how the problem was perceived during the evaluation test and what the shortcomings were of a particular way of looking at the problem. The process of reflection allows for the joint monitoring (by both the researcher and the student) of students' particular ways of experiencing the problems. In this way, the step encourages new ways of "focusing" on the problem. This results in reliving and experiencing the first three steps again, which relates to what Linder

and Marshall (2001) refer to as "mindful repetition". This reflection and possible change has an impact on the overall evaluation of the problem-solution that was worked out initially.

6.2 The two qualitatively different approaches used by the students

It was observed that the steps of scanning and translation were common to all pre-service teachers, while the steps of planning and re-interpretation were apparent in the data of only some of the students. The step of re-interpretation was uniquely a characteristic of Approach A. This observed variation formed the basis upon which two qualitatively distinct problem solving approaches were identified:

(i) A problem solving strategy which involves a way of focusing that brought about a change in the students' focal awareness of the algorithm they employed during the test (to which I refer as Approach A); and

(ii) A problem solving strategy that does not involve such a way of focusing, meaning that there is no change in the students' focal awareness of the algorithm they employed (to which I refer as Approach B).

6.3 A brief summary of the content of the problem task (test) used in the study

Students were given a simple problem which tested their ability to create objects, in order to compare the two investments, based on the interest rates and the initial capital invested.

Problem: *Write a Java program to calculate the result (Amt) of investing a sum of money (P) at a given interest rate (r %) for a number of years (n). The compound interest formula is: Amt = P (1+ r/100)^n .*

You are required to make use of classes, methods and objects in your programming. Your program must cater for two sets of investments and then compare the overall interest per investment and output an appropriate message as to which is a better investment.

6.4 Illustrative data analysis of Approach A and Approach B

Approach A: Focus on the step of scanning

Scanning

Referential aspect

The step of scanning is characterized by the process of organizing what the students know. It is important to note that the students categorized as having used Approach A employed this process throughout the problem solving algorithm. The following transcripts of pre-service interviews below illustrate the step of scanning.

Structural aspect

The problem task forms the *external horizon* and the focus is on the problem at hand; that is, the *internal horizon*.

Scanning(i): the simultaneous identification of the problem-type as well as the algorithmic skill necessary.

The following transcripts from the pre-service teachers, illustrate this kind of scanning.

S3: I read the question and saw what needed to be inputted, what's needed first what processes were needed and what will be the output.

R: What steps did you follow in order to solve the problem? Think back.

S5: I made my class and I was thinking of what type of methods I use for my objects, and I created those

---

R: How did you go about solving the problem? Can you trace back your steps as soon as you got the test?

S1: Okay I thought about the methods that needed to be in the class and the input values that should be in the constructor and … I just laid it out I just made an outline of the...(inaudible)

Although the problems were focused upon within the conventional context of the application of programming principles, the students interpreted each problem in the light of their own familiarity with the problem. In the previous descriptions, students focus on the importance of understanding the problem. With regard to the OO approach, this means being able to create the relevant objects using constructors. The type of scanning indicated (simultaneous identification of the problem-type and algorithmic skill) allows for the conscious delineation by the students of both the qualitative essence, and the consequential aspects of the problem. The key aspect that characterizes this category of experience is to *understand* the problem. What are the required data (input)? What are the conditions or criteria and so on? The pre-service teacher's transcript of the interview below gives an indication of the kind of understanding involved in the step of scanning:

> *S6: Well I went over 3 specific examples that we had done last week that Mr … had gone over with us as well. I broke it down over the weekend. I printed them all out and looked for similarities between the three. I managed to break down each of the programs into their structure by doing that it finally clicked in my head how this whole object-oriented programming works.*

The student sees the need for understanding the underlying principle of object-oriented programming.

Approach B: Focus on the step of scanning

Whereas the scanning (organization of the knowledge system) of students categorized as having used Approach A reflected an exploration of the meaning of the problem, the scanning of the students categorized under Approach B was essentially "algorithmic" or "sequential". The exploration of the overall meaning of the problem is kept to a minimum, as are the features dominant to the problem. They were more concerned with finding the appropriate algorithm.

Scanning(ii): pattern recognition according to familiarity

A number of the pre-service teachers' transcripts of the interview below are evidence of the kind of scanning that was dominant in their approach to solving problems.

R: *What were your feelings about the test?*
S7: *hmm… I was a bit confused with my objects, but… I didn't think it was too bad. It wasn't a complicated program that he set for us to write.*
R: *How did you prepare for the test?*
S7: *Just did questions that Mr …suggested that …we did programs that he suggested we try to do. We went over the ones that he done with us in class.*
R: *Was it anything similar to what you've done in the test?*
S7: *I hadn't done that one.*
R: *But the similar kind?*
S7: *Yes. Creating objects*

R: *How did you prepare for the test?*
S5: *I worked on examples, previous examples and examples the lecturer had given us and (paused)*
R: *Mostly just examples the lecturer had given you…?*
S5: *Yeah*
R: *Was it similar to what you had in the test?*
S5: *Yes it was similar*

R: *How did you learn for the test?*
S3: *I looked at other examples from what we worked and got an idea of how to do it. …like specific order which you do, like what's needed*
R: *You mean the structure*
S3: *Yes*
R: *Was the problem anything like you did in the lectures. Was it similar?*
S3: *Similar, He took little bits of different questions and put them together.*

R: *Now that you say you had a few problems interpreting it. Which part or aspect do you think you misinterpreted or had a problem with? Was it making use of classes, objects, designing the methods?*
S1: *Creating the objects… creating the objects actually.*
R: *That's where you had a problem.*
S1: *Yes.*
R: *Do you understand what an object is in programming?*
S1: *I do have an understanding of what it is but not the entire thing*

These descriptions point to the criteria according to which the students interpret the problem statement. According to Laurillard, this interpretation focuses "attention, not on

the problem itself, but on the problem as set by the instructor in the context of a particular course" or educational setting (Laurillard, 1984:131). All students looked for some kind of similarity to what they had studied for the test. Furthermore, this way of focusing on the problem depends strongly on familiarity. The impression is created that the greater the familiarity with a particular algorithm, the lesser the need to engage with the problem conceptually.

Approach A: Focus on the step of planning

***Planning*:**

Referential aspect:

Drawing diagrams to indicate both the flow of control, and the actual steps involved in solving routine tasks, is the way in which students in this category solve a problem in programming. Finding a connection or connections between the given information and the unknown is what characterizes the students' problem solving strategy.

Structural aspect

The external horizon

The problem within the programming language forms the boundary, or rather the periphery, of the student's external horizon.

The internal horizon

The focus is on the problem domain, the kind of inputs that are required, the kind of outputs that should be generated and the flow of control.

Planning (i) the simultaneous identification of the problem-type as well as the algorithmic skill required.

The pre-service teachers' transcripts explain this way of planning fairly well:

> *R: Did your program run?*
> *S6: No unfortunately and this is thanks to or no thanks to Mr …*
> *After I understood the problem I decided to have nothing in the main method, but my object creation for instance initializing I was going to call for the different methods that I needed. In other words, in the class itself I had a method for display heading, a method for input request and a*

> *method for working out. The only thing that was going to be in the main method was the output statement and the calling of the methods. So I asked him, I can't remember what I had asked him about, and he came over and looked at my program. He said, what are you doing, why are you putting the display heading in the class it should be… or shouldn't be there… So I realized it can be, normally it is in main method.*
> *I also realized for neatness and my own sake I didn't mind it being in the method because I was calling for the input statements twice I had two separate objects. Subsequently I changed that and couldn't see it run… there wasn't enough time to make anymore changes?*

The above quote suggests planning on the part of the student, who first identifies the type of problem and then plans the algorithm in terms of the objects, methods and constructors that are needed. The student clearly demonstrates engagement with the problem task.

On a more general note, the following quotations, from the journals of in-service teachers, show a form of planning that is closely related to this type of planning:

> If I struggle to solve any computing problems I usually contact my former classmates (some of them are now in top positions in the corporate programming world) and get some help. Alternatively I have a habitual peak at http://java.sun.com or the like.

If the student cannot solve the proposed problem then he or she solicits help either from his/her peers, or from others with more expertise in programming and other alternate resources.

> I would investigate/ research solutions. A good strategy would be to keep myself updated. Read as much as possible about current & future trends. Visit web-sites like Microsoft if I can't solve the problem myself. Communicate with other educators & find out how they solved similar problems. Practice, Play around with programs. Practice makes perfect.

What is noticeable in the above quotes from in-service teachers, is their commitment to obtaining help from more experienced programmers or other available resources.

> First understand the problem → what is required.
> Remember the different programming structures and syntaxes in Java. (compared to Pascal)
> Use the IPO (principal) which stands for Input Process Output and use a pseudocode or flowchart as an algorithm to solve the problem.
> Use the KISS principal also – Keep It Simple, that is use simple structure and technique.
> Write the program on paper before typing the program.
> Use test value to verify the program output.

The previous quote, from an in-service teacher, again indicates a commitment, or rather a strong leaning towards, planning before implementation. There is also a strong emphasis on the need to read and understand the problem. Moreover, what is clearly indicated is the continuing comparison of Pascal to Java. As a result of this constant comparison, it might be possible that they do *not* see identifying the objects in the problem as the defining feature of problem solving in OOP. An extract from an in-service teacher's journal (used in chapter 5 and reproduced for convenience below) indicates some form of planning:

> Problem solving is a process, it includes:
> Planning
> Writing test cases
> Implementation
> Remember; thinking and planning of an algorithm should take 90% of the time and coding only 10%.

The commitment to planning may be misleading in the in-service quotes, as this is "head knowledge", given their previous programming experience. They may inform their students of the stage of planning, but might not actually be doing it themselves. There is not enough empirical evidence to suggest that they are planning in this new approach. The approach to problem solving must be re-examined; if we accept that object-oriented problem decomposition is different from top-down decomposition.

Planning (ii) involves pattern recognition

The following extract, from the transcript of the pre-service teacher's interview, illustrates the planning step described:

> *R: Were you able to grasp the formula for compound interest? Did you understand how that works?*
> *S7: Oh! Ya,because we did compound interest last year in one of our programs.*
> *R: Why did you go about solving it the way you did? Is there any particular way or is it the only way?*
> *S7: No, Yes it's what makes sense to me, it's what I did*
> *R: I just heard that you could not actually get it to run.*

The student used an existing/known schema to solve the problem. The student was able to recognize a common pattern in the problem to the one he did before. However, he failed to

pick up the ‘essence’ of this problem: he failed to apply the general principle that lies at the heart of the problem. There is a lack of conceptual engagement with the problem itself.

A similar situation is experienced by high school students, as reported by an in-service teacher’s quote below:

> When I am solving a problem, I think about it very logically and break up the steps in my mind. If I’m unsure as to the solution, I take simple test values and see how to solve using these. I make sure each step or method works properly before moving on to the next.
> Yes. I find that learners use the language inappropriately. Instead of adapting, they regurgitate code as per the text book. This is a real problem as they are totally confused if the questions are phrased even slightly different to that of their notes.
> I normally get them [learners] to write procedures instead of an entire program. I usually give them the declaration and expect them to use local variables thereafter. They do need hints otherwise they are at a loss!

The quote gives us an indication that students look for familiarity or rather a pattern (*they are totally confused if the questions are phrased even slightly different to that of their notes*) that is recognizable, before embarking on a plan. Clearly the teacher (high school teacher) experiences a similar situation (of lack of conceptual engagement) among the learners, even if it is a procedural language which he/she is referring to. From an analytical perspective, it is important to note the experiences of a teacher of programming in grades 10 to 12. Here, again the above quote highlights the teaching approach adopted by the teacher. Aspects of the entire problem are required to be written in the form of procedures (analogous to methods in OOP). An important observation, (transcribed below) made by an in-service teacher, gives an idea of the difficulty novices experience with regard to programming:

> * A big problem for the inexperienced student is planning and starting.
> We as subject advisors had hoped that this course will bring on board a good number of new educators who can handle programming when the NCS kicks in and so broaden the pool of potential programming teachers in KZN; unfortunately only those educators who already have a programming background are able to handle this course. We are going to lose most of the rest.

The above quote confirms what has been said in the literature; that starting and planning are difficult for, if not absent in, the novices’ problem solving strategy.

Approach A: Focus on the step of translation

Translating

Referential aspect

In this step, converting the plan or idea into code (or language) that is recognizable by the computer, using the necessary language constructs characterizes this moment. It is important to note that this way of experiencing problem solving encompasses the previous step of planning.

Structural aspect

In this step, the *internal* and *external horizons* that are formed by the syntax and semantics of the language blend together. Students experiencing or going about programming this way are focusing on the implementation of code and different constructs that are required.

Translating (i) simultaneous identification of problem-type as well as the application of the algorithm

The following pre-service teachers' transcripts give us some indication of this step of translating:

R: *How did you prepare for the test?*
S4: *Just looked at the examples and wrote out the solution*
R: *How did you interpret this particular problem?*
S4: *What do you mean?*
R: *What's the first thing you did when you saw this question that you? Can you trace back your steps that you did?*
S4: *Okay, first I wrote down the… that calculating the interest part first. First I wrote how you write it in Java, and then I started the main program input, then call for all methods, and output, then I just started typing.*

S5: *I did because it was… compound interest. Yes...and it was quite logical*
R: *What steps you followed in order to solve the problem? Think back.*
S5: *I made my class and I was thinking of what type of methods I use for my objects, and I created those.*

The students are simultaneously aware of the algorithm required, as well as the concepts guiding the various parts of the problem representation. They are able to translate the problem into a structure that resembles a program, using the algorithmic structure needed for this program, even if their algorithm is incorrect.

Approach B: Focus on the step of translation

Translating (ii): pattern recognition according to familiarity

The students' problem analysis and execution of the features elicited in the step of scanning are characterized by a procedure of "pattern recognition and formulae duplication" (Caillot and Dumas-Carre, 1989).

R: Think back and trace back your steps. What was the first thing you did when you saw the problem?
S3: I read the question and saw what needed to be inputted, what's needed first what processes were needed and what will be the output.
R: Were you able to create the objects? Err...
S3: Yes

R: How did you go about solving the problem? Can you trace back your steps as soon as you got the test?
S1: Okay I thought about the methods that needed to be in the class and the input values that should be in the constructor and … I just laid it out I just made an outline of the...(inaudible)
R: So did your program work, run?

R: What steps you followed in order to solve the problem? Think back.
S5: I made my class and I was thinking of what type of methods I use for my objects, and I created those
R: Did your program work, run?
S5: It did run even though... I don't know how.
R: Did it give you the desired output?
S5: I think so

R: How did you interpret the problem? How did you go about solving the problem?
S7: Oh well, I first figured out what kind of methods I needed, I needed to create two objects, then broke down the methods then just programmed it.
R: So you mean you did planning on the paper first.
S7: Yes.

If one looks very carefully at the previous quotations from pre-service teachers, one can see the influence of the procedural approach in some of the reflections. S3 is a student who has a Pascal programming background from high school. The approach of identifying inputs, processes and outputs in a problem are what characterizes the procedural approach. Whereas the other students (S1, S5, S7) are thinking first of the objects, methods and classes that are required.

Approach A: Focus on the step of re-interpretation

Re-interpretation (i)

Referential aspect

This category of experiencing problem solving in programming is characterized by looking back techniques and by examining the solution (or part solution) obtained.

Structural aspect

While the problem domain forms *the external horizon*, the i*nternal horizon* comprises the semantics of the language and the logic of the problem task.

The descriptions below (pre-service teachers' interviews) offer examples of students not simply exploring the problem task, but questioning their own interpretations of algorithms. In this sense, the students arrive at a change in understanding, which is a shift in focus:

S4: mm…I made lot of mistakes, the only thing
R: Mistakes in which respect?
S4: Like missed out semicolons and brackets
The other thing was for the display, when we output it I think it was supposed to be static, I'm not sure, because not each object is not going to be output separately; we output in one so I made that method static.
so I don't know if its right that's what I had problems with.
R: But it still worked?
S4: Yes it still works
R: What does static mean?
S4: Static is… the method does not belong to the object, it belongs to the class; in that problem you have two objects, but in the output you give which is the better of the two. That's why I choose the static method.

S6: No unfortunately and this is thanks to or no thanks to Mr …

After I understood the problem I decided to have nothing in the main method, but my object creation for instance initializing I was going to call for the different methods that I needed. In other words, in the class itself I had a method for display heading, a method for input request and a method for working out. The only thing that was going to be in the main method was the output statement and the calling of the methods. So I asked him, I can't remember what I had asked him about, and he came over and looked at my program. He said, what are you doing, why are you putting the display heading in the class it should be… or shouldn't be there.. So I realized it can be, normally it is in main method.
I also realized for neatness and my own sake I didn't mind it being in the method because I was calling for the input statements twice I had two separate objects. Subsequently I changed that and couldn't see it run... there wasn't enough time to make anymore changes?

During the test, student S4 realizes that the output should be in one method and that the method should be static; however, upon further interpretation, she is not sure if there should be two methods for the output, since there are two objects. Whether this further interpretation is correct or not is irrelevant for the purposes of this study, because it does show that the student is capable of re-interpretation (comprising looking back techniques). The correct re-interpretations would follow once confidence is gained in programming.

Student S6 also displays an instance of re-interpretation with regard to how the methods should be structured. The following pre-service teachers' transcript from the pilot study shows the step of re-interpretation (problem appears in appendix A2).

R: *But once you've interpreted the problem do you think you could actually solve it. You think you could actually implement it in Delphi code?*
S8: *Yes, I think I can. I've been working on it. I know I'm a little bit behind to what we are doing in class. I'm sure I can work this out. I thought I'd done this perfect,…and then I found out it was countdown that was because I did not understand what it was asking for.*
R: *But I think you have, according to what you've done in the test you've actually did what was required. It is this portion here where you had to say that if thousand rand is withdrawn every year, according to you you've divided by a thousand rand.*
S8: *You take off a thousand rand*
R: *But do you divide or subtract?*
S8: *Take that amount, you divide it by that.*
R: *Every year you are withdrawing?*
S8: *Yes, no I understand that but I was saying that in 15 ...you've got R15000*
R: *And at the end of the year you want to withdraw R1000, what will you do?*

S8: You subtract it off, yes sure off-course, but the way I worked it out was I said okay, I've got R15000, you take 15 divided by how much you take out every month and you get how many months you have. Look, I realize I've done it wrong, but…

During the test, the student interpreted the calculation of interest completely incorrectly. He now sees his error in the interpretation and realizes that one has to subtract R1000 every year. While this is a problem task to be solved in programming, it is important to note that the error was not a conceptual programming one, but rather a lack of knowledge regarding interest in the domain of banking. The instructor assumes general knowledge of interest earned at the banking institution.

The instance of re-interpretation is present in the strategy employed.

Research on problem-solving has highlighted the importance of "cognitive monitoring" (see Dufresne *et al*., 1992). Although phenomenography has no such cognitivist view, it does propose that students bring certain relevant structures to the learning situation, and that these learning structures mediate the constitution of understanding. Closely related to the idea of monitoring what one does in a learning situation, is the recent theoretical development in phenomenography of the concept of "reflective learning"[1] by Linder and Marchall (2001, p. 25), which derives from the notion of metacognition.

The above illustration(s), of the "reflective monitoring"[2] that students do during problem solving, indicate that through conscious reflection and testing of their ideas, students do in fact show themselves capable of developing their relevance structure in this way.

The results of research question 2 are summarized in table 6.1.

[1] When learners encounter a novel, complex or confusing phenomenon they need to have a conception of learning which will facilitate the discernment of critical aspects of the phenomenon in order to make sense of it, solving the problem it presents, or conceptualizing what it represents. In other words, they need to confront those aspects of the phenomena, which are taken for granted to become invariant, and vary them. As such *reflective learning* is the exploration of *the object (content)* of learning through a mindfulness of the *act* of learning.

[2] "Reflective monitoring" refers to the monitoring of students' understanding and conceptions during problem solving. This process of monitoring fulfils a diagnostic purpose. It draws the students' attention to the interpretative framework being used to solve or make sense of a particular aspect in the problem task. In other words, reflective monitoring reveals the "status' (Hewson and Thorley, 1989) of students' understanding during problem solving.

Table 6.1 Summary of findings of Research Question 2

Instances of problem-solving/programming	Approach A	Approach B
Scanning	Scanning (i) involves the simultaneous identification of the problem-type as well as the algorithmic skill necessary	Scanning(ii): pattern recognition according to familiarity
Planning	Planning (i) involves the simultaneous identification of the problem-type as well as the algorithmic skill Planning (ii) involves pattern recognition according to familiarity	No planning
Translation	Translation (i) involves the simultaneous identification of the problem-type as well as the algorithmic skill	Translation (ii): pattern recognition according to familiarity
Re-interpretation	Re-interpretation (i) involves a change in focus informed by a new relevance structure to evaluate solution representation	No re-interpretation

6.5 Summary

The results obtained point to the following implications for programming (solving problems in programming). Students categorized as employing Approach A (re-interpretation of the problem representation subsequent to scanning, planning and translation) challenged their understanding of both the concepts, and the algorithms, they used. In this way, modifications of relevant structures/changes in understanding became possible, to which varied levels of commitment were expressed. The students categorized as Approach B (scanning and translation without planning and re-interpretation), in contrast, were more concerned with the formal requirements of the problem tasks than with their understanding of the content of the problem, and did not engage in a conceptual exploration during the solving of the problem task. The factors that were seen to bring about those two distinct strategies are the subject of chapter 7, which explores Research question 3. Approach A and Approach B have a strong leaning towards characteristics of the expert and novice programmers respectively (as discussed in chapter 2, section 2.3).

An important distinction has been observed between the two groups of students; pre-service and in-service teachers. In the planning step which is evident in Approach A, the pre-service students that have had no prior programming experience always approached the problem task by trying to identify the objects and methods that are required (as seen in their scratch sheets that were handed in, together with the test). A diagrammatic representation of the plan, indicating the class and objects of the program, is shown. Although their plans might not be accurate representations of the approach to object-oriented programming, they do indicate that they are thinking in terms of “objects”. On the other hand, the in-service teachers, who seem to have a better grasp of programming, do not display this approach in the planning step. They continue to refer to the inputs, processes and outputs that are required; an approach which is characteristic of the traditional procedural way of thinking. (see figure 2.1 chapter 2) They therefore do not realize that this is a different paradigm.

7 THE LEARNING CONTEXT: IMPACT ON APPROACH TO LEARNING TO PROGRAM AND PROBLEM SOLVING

According to Laurillard (1994),

> [e]ach step, and each strategic decision made, refer to the immediate context of the problem as it occurs in that course. The criterion is not "is that what this type of microcomputer needs?", but "is this what this teacher is looking for?" …The problem is not an isolated event; it comes after a certain lecture… it will be marked by a particular lecturer, and the solution should take that into account as well.

In this chapter, the results relevant to research question 3 are presented. In particular, the influence of the learning context, an aspect of the activity system, is considered. It was possible to identify three sources of influence, on the learning context of learning to program: institutional and its subcategory, disciplinary context; personal context; and teacher perceptions. The institutional and disciplinary context may be further delineated into various settings, namely: studying, lecture, test, previous knowledge. This chapter will argue that students' experiences of the learning context have important implications for teaching and learning. Laurillard (1984) highlights that, while with experimental studies the problem situation can be treated in isolation, the case is different for students solving problems as part of a programming course. The solutions that most students work towards in order to solve a problem are an indication of an essential aspect: the learning context. Both the problem and the learning context have an effect on students' understanding and performance.

In phenomenographic research there is no specific divide between the act of knowing and the context in which it occurs. Marton and Booth (1997) make this point as follows:

> We cannot separate our understanding of the situation and our understanding of the phenomena that lend sense to the situation. Not only is the situation understood in terms of the phenomena involved, but we are aware of the phenomena from the point of view of the particular situation. (Marton and Booth, 1997, p. 83)

The theoretical framework developed in this thesis emphasizes the mutual shaping of the learner and the learning context.

In order to address research question 3, the investigation of learning to program and problem solving is based on Leont'ev's (1981) first level of analysis of the activity theory approach, namely, the context in which the activity takes place.

In section 7.1 a description of the learning context, as used in this study, is discussed. Section 7.2 presents a detailed analysis of research question 3 and the main points are summarized in section 7.3. A further dimension of the learning context, teachers' perceptions of teaching and learning, is discussed in section 7.4. Section 7.5 concludes the chapter.

7.1 Brief description of the context as used in this study

7.1.1 Institutional context

Research question 1 (addressed in chapter 5) and research question 2 (addressed in chapter 6) explored the *content* (Laurillard, 1984) of the experience and strategies that students employed. This chapter, by contrast, explores what constitutes the *context* of these experiences. What do we mean by context? The notion of context is multi-faceted. Ramsden (1984) argues that a student's perception of the learning context is central to his or her experience of learning. This study, conducted within a university course, is obviously concerned with a certain type of institutional context. Most factors that influence the students' approaches to learning to program/problem solving – and the meaning they attach to learning programming – can, in some way, be said to derive from the fact that they are learning programming at an institution of higher education.

A certain type of institutional context – that of a first year programming course at an institution of higher learning – is, however, of significance. In this study, students' experiences of the learning context comprised two types of institutional context: face-to-face and distance mode tuition.

Disciplinary Context

Within the institutional context there are a variety of disciplinary contexts. Each disciplinary context would have features that are specific to that discipline. For example, the discipline, computer programming, involves writing a program to solve a problem using the computer or reading program code to determine the output of the program. This study considers aspects of a university course such as, course content and structure, assessment and the perceived demands and support of the teaching staff. It is, therefore, necessary to give recognition to the features of the disciplinary context and its implications for problem solving (programming). In other words, the discipline of programming constitutes the first and principal context with which we are concerned. It follows that the context refers, and gives rise, to a variety of spatial settings (contexts) that are, in a sense, dependent on institutional context. This chapter is particularly concerned with the influences of these settings: the lecture, test, previous experience, distance learning environment and studying.

7.1.2 Personal context

Personal context refers to those "attitudes and aims which express the student's (individual) relationship with a course of study and the university" (Gibbs, Morgan & Taylor, 1984, p. 165). It is through focusing on this personal context that "we can aim to present a more holistic description of students' experiences of learning" (Gibbs *et al.*, 1984, p. 166).

The notion of enrolment (involving factors forming part of establishing an "identity" of a programmer) is also critical in this case. The outcome of learning is seen not only as the interaction between the students and the task, but, in fact, as a function of how students "enrol" themselves into the discipline of programming. Enrolment is closely related to the notion of progress made by students within a particular discipline. In phenomenography, the concepts of students' intentions and conceptions have been used in this regard.

Most studies that are concerned with factors that influence students' approaches during problem solving (see Laurillard, 1984; Ramsden, 1984) bring to the fore the notions of intentions and conceptions of learning. In phenomenography "intention" is discussed in the phenomenological terms of intentionality, implying a unifying bond between the psychic (psychological) and the physical. The term "conception" relates to "the meaning that people see in and ascribe to what they perceive" (Säljö, 1988, pp. 38-39). The thought (conception) is never merely a thought; it is a form of conception intended towards something (see Marton & Booth, 1997, p. 84).

Both intention and conception (personal context), and setting (physical context) influence what students do during programming. Rather than to establish a hierarchy between these two sets of factors (it would be impossible to argue that the one "precedes" the other), we may simply admit the constant influence – and interaction – of both. Laurillard (1984) makes a similar point that students' choice of approach does not "wholly" derive from their intentions, but depends "on the nature of the problem solving task itself and also on how the requirements are perceived".

7.2 Analysis in terms of the settings

The meaning that the students attached to the different settings of problem-solving

A crucial concept for characterizing the meaning students attached to their various instances of problem solving (programming) across different settings is the concept of *familiarity*. Before looking more closely at this concept and how it informed students' problem solving strategies, it is necessary to describe in detail from where this familiarity is derived. Students' familiarity with the problem tasks was seen to stem from their exposure to the settings mentioned in the previous section, specifically the settings of studying, the lecture, previous knowledge (as in the case of in-service teachers) and the test.

The interview setting is not singled out as a separate setting here, seeing that it served as a mechanism of reflection, through which the meanings of different settings of problem-solving (programming) could be explored.

In analyzing the influence different settings have on the experience of learning to program and problem-solving, two aspects are described for each setting. They are:

(i) how students use particular settings (in the case of studying, for example, the different means of studying at their disposal). How the setting is used is seen to reflect the influence of the physical context on programming;

(ii) how the students relate the physical context to their personal contexts; bringing to the fore their intentions and conceptions of learning to program and programming.

The use of interview transcripts and journal writings of both pre-service and in-service teachers were used to illustrate the influences of the settings by addressing the above two questions.

Some quotations and extracts from transcripts will be repeated, in order to illustrate different aspects during the analysis.

7.2.1. The setting of studying

Studying refers to how the disciplinary "tools" such as textbooks, lectures notes, study guides and problems were used by students. The investigation of the setting of studying suggests perceptions that students have of these means of studying. Most students claimed to have studied in groups; the meaning attached to group versus individual problem solving, therefore, became another interesting aspect of this setting.

(i) How students use the setting of studying

The analysis indicated that the students' use of the disciplinary tools, whether it be the textbook or study guide, can be categorized as aimed at either reproducing or understanding the material learnt. The following descriptions, from the entries of in-service teachers' journals, provide examples characterizing these two distinct ways of focusing on the disciplinary tools within the setting of studying:

> I found the work very easy. Even though some concepts in Java are very new, I coped with the work after going through the sections in the books.
>
> ---
>
> I find that the guide and textbook are good-clear and helpful. The different exercises and tasks lead you from one section to another and it is not that difficult to get answers to your tasks, exercises and assignment questions. I plan to use the grade 10 workbook extensively when teaching Java to learners because it is precise and easy to follow and builds from one concept to another. Everything is not told to you there are things that you have to discover but the guidance makes discovery easy.
> Although the study material is within my ability, it is rather time consuming doing the various exercises and we have to do most exercises-I tried to skip some exercises once and got stuck and had to come back to find out my problem.

An extract, from the transcript of an interview of a pre-service teacher, illustrates the use of a disciplinary tool, the textbook.

> S7: *My textbook is very helpful; it's got a lot more theory than actual programs. But I think it has a lot to do with doing it by yourself and working through the problem and you know trying to do it properly.*

Appreciation of the value of the study guide and reference book (text book) is evident in the descriptions. The importance of developing a solid foundation of programming, in an incremental manner, is apparent in the lines: "I tried to skip some exercises once and got stuck and had to come back to find out my problem".

The heavy reliance on the study guide and work-book is indicative of a distance education mode. Although "it is rather time consuming doing the various exercises and [students] have to do most exercises", it is clear that every exercise teaches or highlights some aspect of programming.

Another important aspect of the setting of studying (particularly with respect to in-service teachers) that became apparent, is the support structure provided by study groups. This support is highlighted by the journal entry excerpts provided below:

> We find that by just meeting and talking about our subject has a positive impact on us. We wish to continue with our meetings even after the course is over. We also feel that we can learn a lot from each other and share our materials and resources.
>
> ---
>
> I am not part of a study group but I did manage to contact someone who is also doing the same course. We met once and managed to do some work together. Just meeting once made me realize the merits of group work. I think that I need to look at the letter which outlines my working group and I need to contact those people concerned so that we can do some constructive work together.
>
> ---
>
> At the beginning we were all interpreting things differently and thanks to a common study session we managed to re-visit questions, brainstorm them and finally get onto the right path …

(ii) How the students relate the physical context (settings) to their personal context

Two distinct ways of relating the physical context to the students' personal contexts were identified. They respectively pointed to the process of repetition geared towards *memorization and reproduction*, and repetition geared toward *understanding*.

Studying setting: repetition concerned with memory and reproduction

A quote from a pre-service teacher's journal indicates some form of memorization and an expectancy to solve a similar problem:

> Everything that was in the test was something that was already done or covered in class. I had no complaints about the test but was not satisfied with my performance.

While it is considered routine to test aspects that are covered in lectures and tutorials in a test, it might not give a true reflection of a student's ability to solve a problem. The above quote refers to a problem solving test. The aspects asked of the problem were similar to what was done in lectures, but the scenario was different.

The following extracts from two pre-service teachers' interviews also illustrate this point:

R: Why are you feeling scared?
S1: Okay…It was something that not what I expected...it was totally different from what I expected.
R: How was it different from what you expected?
S1: The way I studied wasn't what the way it was in the test.
R: Okay…You said that the problem in the test was different from what you expected?
S1: yes
R: So what were you expecting for the test?
S1: The actual problem was very different from what we usually do.
S1: I expected some problems like the ones we did in class.
R: What kind of problems did you do in class?
S1: It's actually the question… the way it was…
R: … worded
S1: Yeah… it was worded

S5: I worked on examples, previous examples and examples the lecturer had given usand…
R: Mostly just examples the lecturer had given you…?
S5: Yeah
R: Was it similar to what you had in the test?
S5: Yes it was similar
R: Do you think if you were given a completely different problem from what you've been going through, would you be able solve it?
S5: I would try but not certain it would work.

The above descriptions point to the importance the students attach to previous encounters with problem tasks. The statements: *"Everything that was in the test was something that was already done or covered in class.", "Okay…It was something that not what I expected...it was totally different from what I expected.", "The actual problem was very different from what we usually do"* shows the familiarity that students expect from tests. These tasks were encountered in different settings, namely the lecture and possibly a tutorial. The way in which the students decide to treat the problem tasks in the setting of studying is characterized by memorization. The intention is clearly to reproduce them in a test setting, in response to the lecturer's hint that the particular problem tasks will be part of the test. The factors that play a role in the students' approach to the problem are clearly related to "external requirements" (Ramsden, 1984).

Studying setting: repetition concerned with understanding

The following quotes are from the journals of in-service teachers:

> In question four of assignment 5, we were required to use the print method of the Card class which was created in another program. I did not know at first how to call the print method of the Card class from the PackOfCard class. After a closer analysis of my first attempt, and in particular the call statement to the Card class, I was able to rewrite the program as required and obtain the expected output.
>
> ---
>
> Exercise 7d took up a lot of my time. I worked with all the problems and managed to solve all the problems (took up a lot of my time but was very proud when I was able to solve the problems). Number 3 from exercise 7d took me a bit of time to understand. I had to go through the question several times before I interpreted the question correctly.
>
> I was able to understand "Comparing strings in Java" the first time round. I worked with the ASCII table before, so I knew what was happening. I enjoyed working with the trace tables. This is a 'tool' which I will teach to my learners since it is an important 'tool' as far as debugging a program goes.

What we observe, however, is an orientation aimed towards understanding the principles underlying the problems to be solved. This is apparent in the lines "After a closer analysis of my first attempt, and in particular the call statement to the Card class, I was able to rewrite the program" and "I had to go through the question several times before I interpreted the question correctly". The descriptions point to the students' awareness of the critical factors which need to be focused on when dealing with the problem task. Through the process of exploration aimed at understanding the underlying structure of the problem tasks, the students' encounters with the tasks are repeated in a meaningful way.

There is little evidence of external motivation, as seen in the previous category; the problem task is done "for its own sake" (Ramsden, 1984).

7.2.2. The setting of the lecture (in the case of pre-service teachers)

Descriptions of the studying setting reveal the differences in how students interpreted the hint of the lecturer during a class to "go over" the problem tasks in their preparation. What happens in the classroom not only depends on how teachers conceptualize their roles, but

also on how students perceive and conceptualize their learning, as well as the (authoritative) role of the teacher.

(i) How students use the setting of the lecture

Many students who referred to the lecture associated it with understanding of programming.

One category was identified for this setting: the lecture as a form of authority or convention.

The lecture as a form of authority / convention

Some of the responses, from pre-service teachers to the questions below, are quoted:

> *R: How have you learned to program? Through lectures, internet, textbooks? Is it just attendance at lectures? What do you use as your main resource?*
>
> *S6: It was exclusively from lectures. I came into this course without very much...knowledge, without any knowledge as far as programming was concerned. So everything was new to me.*
>
> *S7: I think I learnt most obviously in my lectures. Mr… does a lot with us in the lectures. If we miss any lecture we do miss a lot.*
>
> *S5: Only through lectures.*
> *R: Have you not supplemented this learning with any other material?*
> *S5: No*
> *R: Just by attending lectures and listening you were able to come to this point of programming?*
> *S5: Yes*
>
> *S2: In my case it's been mostly from lectures. Last semester the more I practiced the more I understood.*

The following quote, from a pre-service teacher's journal, demonstrates the reliance on lectures as a form of authority:

> I did not know what was happening and wished that somebody would teach me the work like a kindergarten child. I never grasped this section. I did understand when the lecturer was demonstrating on the board but when I was alone, I did not know where to start.

What is apparent in the above descriptions is the authority the students associate with the lecture setting, especially as represented by the lecturer and the programming discipline: "If we miss any lecture we do miss a lot; it was exclusively from lectures I came into this course without very much...knowledge, without any knowledge as far as programming was concerned." The setting of the lecture is strongly associated with the idea of enrolment into the programming discipline through acceptance of authority and convention.

Linn and Clancy (1992) have argued that many introductory programming courses foster the development of syntactic knowledge and do not place enough emphasis on the development of conceptual knowledge and program design skills, which are often left to unguided discovery. This concurs with the results of the pilot study conducted (Govender & Grayson, 2006). In this way, the "discrepancy between the [students'] way of thinking about the subject matter and the new way desired by the teacher" may, indeed, never be confronted (Ramsden, 1988, p. 22).

(ii) How the students relate the physical context to their personal context

The total reliance, by many students, on the lecture setting to promote learning presents itself in two ways. Firstly, the lecture setting contributes to the achievement of understanding. Secondly, there is the passive role the students assume in the setting of the lecture. There seems to be the conception that lectures do not require active learning to program. The students do not participate, they receive. This is suggested by the following pre-service teacher's journal entry below.

> During the lecture Mr… went on with chapter 8, but this time we concentrated on nested loops and he tried by all means to show u how they worked. I found this lecture to be boring and as a result fell half asleep and therefore I have no clue how to deal with nested loops

If students are engaged in active learning to program then they should be able immediately to apply the information "gathered' during lectures and be able to attend to the critical aspects of the principles covered. An excerpt from another pre-service teacher's journal, illustrates this point.

> So today started chapter 4 and "rushed" a bit through it I guess…but at least I did have an inkling of what Mr … was talking about…I really like the way he used the block drawing

> explanation of what goes on in memory…(maybe I'll remember to use that myself when I teach it at school….

In the above pre-service teacher's quotation, it appears that the explanation given, with respect to memory allocation of variables, has stirred his intention to use this method of teaching when he practices at school.

7.2.3. The setting of previous learning/ teaching experience

This setting makes up the bulk of what constitutes the students' previous learning exposure to programming. It reveals the students' pre-university history of programming (problem solving) or previous programming background. The role that previous knowledge plays in any new learning context is accepted by most educationists, irrespective of their theoretical framework – whether they be cognitivists (the importance of background knowledge for task analysis), constructivists (to quote Ausubel (1968); "the most important single factor influencing learning is what the learner already knows; ascertain this and teach him accordingly.") or phenomenography.

(i) How students use the setting of previous learning/teaching experience

The students' reference to a previous learning encounter with the problem tasks at high school or teaching focused on representing the past learning experience in such a way as to acknowledge the "discipline trajectory" along which they had moved. In other words, it focused on how students had enrolled themselves into the discipline of programming. The students showed strong identification with the competencies gained through their history with programming through schooling or teaching. This was apparent in the journal entries of the in-service teachers below:

> I did not find units 5 and 6 difficult because I have previous programming experience with Turbo Pascal. I find that certain structures are the same between Turbo Pascal and Java but that the syntax only differs. Drawing parallels between the two languages has made my task easier.
>
> ---

> After working through chapters 1 and 2 of "exploring Java-grade 10", I find that I can easily relate programming in Java to programming in Turbo Pascal….. I also know that Java is case sensitive unlike Turbo Pascal.
>
> ---

The following quote, from an in-service teacher's journal, reveals how previous teaching experience may be used in the teaching environment:

> In my experience of teaching variables I found that learners have extreme difficulty in understanding the concepts of a variable, that is, how variables are kept in memory. I think using variable box diagrams to teach the concept of a variable is very effective. The variable box diagram is a very fresh, novel and interesting way of introducing variables to learners. When I taught the concept of a variable to my learners in the conventional way at the beginning of the year many learners grappled with the understanding of how variables are stored in memory especially when a variable takes on a new value i.e. how one value replaces another in memory. The box approach has sorted out this difficulty that learners were experiencing. I can therefore report that from this experience the variable box approach is excellent and very effective.

An extract, from a pre-service teacher's interview, illustrates the use of previous knowledge:

R: *Now you've learnt a structured programming language, such as Pascal and Java is an object-oriented language, now that you are learning Java, do you find that you get confused with the two languages? Or is it a problem to shift over?*

S3: *I work with them separately. But I try to still remember what I did Pascal. I don't want to forget. But when I'm working in this code I only think of what will happen in this language. I don't put them together… It doesn't work.*

R: *Okay. That's as far as syntax is concerned, but the actual problem solving is a little bit different?*

S3: *I use the knowledge of the previous language.*

The above descriptions give an idea of the meanings attached to being enrolled in the discipline of programming through past experiences. According to the students, to have learnt to program means to have accumulated certain formal formulations and problem solving techniques; these can be deployed (with or without understanding) in various situations, depending on the demands of the problem tasks. It means being comfortable with certain competencies in programming, such as: "The box approach has sorted out this difficulty that learners were experiencing"; "Drawing parallels between the two languages has made my task easier"; "I find that I can easily relate programming in Java to programming in Turbo Pascal." Past experiences (learning or teaching) seem to provide background knowledge necessary for solving problem tasks.

(ii) How the students relate the physical context to their personal context

The previous learning or teaching experience is assumed. This experience highlights the fundamental nature of enrolment which, in a sense, "precedes" personal context. The students draw upon a variety of competencies (such as those mentioned above) and, in very few cases, it was found that these have been related to the students' personal contexts in terms of seeking meaning. From the in-service teacher's quotes below, it emerges that an attitude of enjoyment is necessary for successful meaning-making:

> This will be my 3rd language so it's just a case of learning new syntax. I enjoy programming very much and I feel anyone who programs needs to enjoy it or else they will not progress. Practice is essential.

The next quote of an in-service teacher, gives an indication of the way previous knowledge of programming helps to confirm the notion of enrolment into the discipline of programming:

> I think that my knowledge from Turbo Pascal is helping me to understand these concepts. I also found that having a programming background was an advantage and I am sure that if I started without one I would have found things much more difficult.

7.2.4. The setting of the test

In learning to program, tests fulfill the role of assessing knowledge acquisition, which includes problem solving. From the point of view of the instructor, tests are perceived to provide a setting in which students demonstrate their understanding through the application of concepts and principles to the problem tasks given in tests. According to Laurillard (1984), "knowing without the ability to apply is rightly seen as a poor commodity". In this sense, problem solving tasks are regarded as an important part of learning. The students' perceptions of the test setting are significant to this study. Is the test setting perceived as a setting conducive for solving computer programming problems?

(i) How students use the setting of the test

It would be highly unlikely, in a test setting, that problem tasks are done "for their own sake" (Ramsden, 1984); the entire setting is structured in such a way so as to emphasize factors that are external to the problem tasks, such as: mark allocation, time limit and the stress of "having to pass". Not all students respond to assessment (test) pressures in the same way. But the students all "use" the test setting in the same way. All the students are, to some extent, influenced by the formal requirements of the test setting, which will also impact on what they do. However, within this formal set of requirements, there is some difference between those students who attend to the demands of the task as required by the test, and those (very few) who extend their focus beyond these requirements.

(ii) How the students relate the physical context to their personal context

The link is explored between successful problem solving (programming) and "repetition" during the test. It was earlier pointed out that students generally knew, as a result of the lecturer's "hints" or guidance that the problem tasks were going to appear in the test. The way they went about preparing themselves for the test differed. They differed according to the intention either to reproduce/memorize or to explore meaningfully.

The test setting: attending to the problem at a test requirement level

The issue of acquiring a good mark (particularly a pass mark) for the test is emphasized in the description below, from a pre-service teacher's journal:

> We're nervously getting closer to the end of the course and we know how important each mark is now.

An extract, from a pre-service teacher's interview below, confirms this notion:

R: *Why are you feeling scared?*
S1: *Okay…It was something that not what I expected...it was totally different from what I expected.*
R: *How was it different from what you expected?*
S1: *The way I studied wasn't what the way it was in the test.*
R: *Okay…You said that the problem in the test was different from what you expected?*
S1: *yes*
R: *So what were you expecting for the test?*
S1: *The actual problem was very different from what we usually do.*

I expected some problems like the ones we did in class.
R: What kind of problems did you do in class?
S1: It's actually the question… the way it was…
R: …(completing) worded
S1: Yeah… it was worded
R: Did you understand the problem and what was required?
S1: Yes I did understand the problem. I had some problems interpreting it

For the students who attended to the problem at the requirement level, the ability to transfer what they had done in the studying and lecture settings to what they did (or expected to do) in the test, is seen as the main issue in relating the physical context to their personal contexts.

Even if they might have explored the problem task in one of the preceding settings, their "over awareness" (Ramsden, 1984) of the test requirement precluded them from attempting a similar exploration during the test. The need to reproduce something that is familiar becomes the issue. Therefore this can be seen as limiting the students' willingness meaningfully to engage with the problem task. The students showed confidence in repeating the practices to which they had been exposed in previous settings. However, they lacked the confidence in their own ability to relate these practices (techniques and methods) in such a way so as to make sense of the problems, (if presented slightly differently) encountered in the test setting.

The test setting: attending primarily to the content

The following descriptions illustrate problem solving that is primarily motivated by an understanding, or at least an awareness, of the critical aspects underlying the problem tasks:

S1: Well …you never say you know…you know …
R: Until you work the problem (completing the sentence)
S1: It's not something that you can work the problem and say you know. You really need to… know programming.

S6: *After I understood the problem I decided to have nothing in the main method, but my object creation for instance initializing I was going to call for the different methods that I needed. In other words, in the class itself I had a method for display heading, a method for input request and a method for working out. The only thing that was going to be in the main method was the output statement and the calling of the methods. So I asked him, I can't remember what I had asked him about, and he came over and looked at my program. He said, what are you doing, why are you putting the display heading in the class it should be… or shouldn't be there… So I realized it can be, normally it is in main method.*
I also realized for neatness and my own sake I didn't mind it being in the method because I was calling for the input statements twice I had two separate objects. Subsequently I changed that and couldn't see it run.. there wasn't enough time to make anymore changes?

Even when they focus on the mark (and the authority of the lecturer), the students are still guided by the content of the problem tasks (reasonableness of the solution).

For students attending primarily to the content of the problem, problem solving in the test setting is seen as an indication of understanding. Emphasis is placed on the concepts that underlie the problem tasks. In cases where students ran into difficulty with the problem tasks, they were prepared to try to re-evaluate their understanding in a systematic way, guided by the problem at hand. Problem solving in the test setting may be perceived as a means by which students relate their learning experiences in a meaningful way.

7.3 Summary of the institutional settings

An overview of the various settings in which students are exposed to problem solving served to provide a contextual framework for how students learn to program (or problem-solve). The quotations of students suggest that familiarity with a programming task influences their approach to learning and solving problems. The research question on factors influencing what students do during programming/problem solving can be seen as seeking out those aspects of familiarity[1] that affect their problem solving. In other words,

[1] From a phenomenographic perspective familiarity should be seen as the principle component of relevance structure. Relevance structure relates to what is called for to make sense of things, and to the criteria by which some parts of the problem are seen as more (or less) relevant.

students' familiarity with the problem task (acquired across the different settings) appeared to influence what they did.

Table 7.1 summarizes the findings of institutional and disciplinary contexts in the form of different settings.

Table 7.1 Summary of main findings of research question 3

Settings of programming (problem solving)	How do students use the setting?	How do students relate the physical context to their personal context?
Studying	• To reproduce material under study • To understand material under study	• Repetition related to memorization • Repetition related to understanding
Lecture	• Lecture as form of authority / convention for teacher –to-student transfer of content	• Learning to program is associated with a total reliance on the lecture setting • Students' role in lectures perceived as one of passive participation
High-school/previous learning/teaching experience	• To compare the two languages to solving problems • Previous programming experience used with authority in solving problems using OO language	• Identity of being a programmer (enrolment) precedes the attitudes and intentions of students (personal context)
Test	• Attending to the problem at the test requirement level • Attending to the problem at the level beyond the test requirements	• Problem solving is confirming the lecturer's approach • Problem solving is interconnecting learning experiences in a meaningful way

7.4 Adding to the context: University lecturers' perceptions of learning and teaching programming

Chapters five and six indicated the diversity in students' conceptions of learning to program; approaches to learning it and implicit attitudes and intentions to learning to program. Similarly, there will be diversity among the perspectives of the educators

responsible for moulding a course. Teachers are an important part of this network of relationships. They influence the perceptions and actions of the students. In turn, teachers' perceptions and behaviour are formed by their own experiences, both their past experiences and their current awareness about the students and setting. Teachers' actions are constrained by the conditions surrounding them. Hence, part of understanding student learning relates to interpreting the activities of the teachers of programming, as they manage their classrooms (see pilot study).

The following extracts, from transcribed interviews, show how two lecturers responsible for the pre-service course perceived the setting for teaching and learning programming. I shall denote the teachers by T1 and T2. Each was interviewed separately and the interviews were semi-structured. A guide to the questions is reproduced in Appendix E. However, as far as possible, I allowed the teachers to tell me about their perceptions, with minimal interference. Similar themes emerged from the two interviews and I shall report on these together for purposes of comparison.

Interviews with T1 and T2

Both T1 and T2 had clear ideas of the purpose of learning to program. T1 saw learning to program as a way of enabling the students to teach programming to a level that would prepare students for university computer science. In response to the question,
"What do you see as the purpose of teaching programming to pre-service teachers?" T1 said:

> Well ultimately, to get people to be able to teach computer studies/programming to such a level that the student that comes out with a matric will be capable of going into programming at university level, that's the idea I had, that's what I did when I was at high school. I was able to enter university with a computer studies background. Computer science at high school helped me to prepare for the BSc degree with comp sc and maths. The computer science was prepared by the high school studies. It's to enable students to reach a level of competency so that they can prepare others to enter university to do computer science.

He hoped that what students would get out of the programming course was:

> I think, the main thing to me is always to have the right thinking processes. Sometimes students in whatever course it might be would focus on the concept of 'what must I put down on the piece of paper to get the right mark', that's one of the goals. The other thing is

> to the thinking behind it, a person knows the concept and understands them as it refers to programming from C++ to Java and facet of computer science, if you got the thinking right then you can adapt quite easily.

T2 saw computers as an important aspect in our everyday lives and related this idea to the future. He said:

> In this day and age everything centres around computers and for computers to function somebody needs to write a program. Even though everything is made very simple today for the user, but it's the programmer that needs to do the entire problem solving and make the application programs work, so that the computer user is not bogged down with issues of programming.

T2 then went on to say that he hoped that the problem solving skills will be developed:

> Therefore we hope that we will try to sharpen their problem solving skills so that especially those from school who want to get into tertiary education institutions, they can then pursue computer sc, where programming is taught, programming will enable them to become much better thinkers, problem solvers; and I think the bottom line is writing programs that are correct.

These teachers were asked how they tried to achieve these educational aims for programming in their teaching. Their responses showed that, while practical examples were of key importance, their emphasis was slightly different. T1 followed the textbook and explained the concepts by drawing on everyday analogies, and T2 encouraged problem-solving ability and algorithm development.

T1: You cover the material that's available in the textbook. When you go through it, you try to explain the concepts, where for example we are doing basic introduction to java programming with objects. You try to get them to see how it is a more natural way of programming paradigm than a procedural paradigm and discuss the advantages of one and the advantages of the other. You can decide, depends on the situation show how this way of doing it resembles the real world. Try to compare the way real world objects would interact with each other then see how they correlate with the PC. Talking about students for instance in real life, the student has all these faculties about himself; that is arms, colour of eyes etc. For instance, you don't need to worry about a student sticking out his arm to shake hands, he knows he has this arm which comes naturally. They draw on all these attributes. In the same way other objects have attributes. In this way I try to get their understanding to that level.

T2: It's the art of teaching computer programming; starting off with simple algorithms. So that students are aware of what it means to solve a problem before even going into a programming language. So simple problem solving steps, that students need to be familiar with, which could even be similar to solving problems in other subjects such as

> mathematics and physics, the same problem solving skills are required and together with the logic that's required.

There seems to be a fine line between the conceptions of what it means to learn to program from the perspective of each lecturer. This difference emerged more clearly from their comments about how a student's understanding of programming was measured. For example, in response to the question, "How do you know if they have understood whatever you've taught them?" the following comments were made:

T1: > This was the hard part for me. The students were not very responsive. For me the small class and the interactivity should have been more evident. But that did not happen. You explain the concept and ask them if they are okay with it, they were not interacting, I don't know whether they were completely lost and so did not have the confidence to answer. Sometimes they may think they know it but when you ask them to go and code it algorithm, then you see the problems they have, and that's when I realized how they struggle.

T2: > As long as they are able to apply, again it depends how you are going to test them. When you teach a concept as long as they are able to apply… If there's a problem the student must be able to write a program. You check that they applied some basic programming principles.

T1 evidently put more emphasis on the programs to be written as a product of learning, while T2 emphasized an incremental, conceptual understanding.

The institutional setting shapes educator's activities as much as it does students' perceptions and actions. T1 and T2 agreed that a shortage of teaching time for programming was a major constraint.

T1: > The three modules of programming are not enough for a teacher in training. If more is expected out of them maybe they can adapt to the new way of working. Maybe more prac time should be given. When I studied we had 4 lectures and 4 tuts for practical lab time. It is difficult to enable the students to get to a level of competency of an educator within this time.

In addition to lack of time, T1 felt that the language barrier was a constraint in the process of understanding the concepts:

> The one thing in a sense that was highlighted for me is the language barrier. One student was speaking in slightly broken English. I was trying to understand him. Maybe a first language person would be better able to explain to him. Ultimately he may have the skills to learn the concept, but he already has a language barrier that set him back.

T2 felt that the students were not academically inclined towards this way of thinking:

> Well the concern that we experience especially in this faculty of education, we do not attract students of a very high academic ability. So they come in,…they have their exemption, and when we compare them to those registered for the pure sciences, those students probably have a higher academic ability, therefore that's one of the constraints. Because they are training to become teachers we don't attract very strong academic students.

It seemed that the most important factors around which students organize their activities are tests and examinations. T1 explained his ideas on the part played by assessments and how students respond to them:

> Some of the things they had do, like, searching and sorting, when I lectured to them they seemed understand it and when I tested them it reinforced the concepts. It was hard to assess the programs like they had compiler errors.
> For instance if they were using the for-loop, they did not realize that the loop counter i was the same as the i further down that could have been used for the output directly. Like when I was explaining the solution after the test, they wanted an explicit solution. I expected that when you explain the solution they would take it down, but they wanted an explicit answer. Kind of rote-learning; they are students who are going to become educators in about 4 years, they should be able to understand and adapt, but for them it's like wanting to get through the examination.

The activity framework developed highlights the need to recognize that the classroom is not just a place where instruction is received, but a social structure formed by the actions of students and teachers. The interviews with T1 and T2 provide some insights into this constitution.

7.5 Conclusion

In this chapter, different types of context, (namely, institutional, disciplinary and personal) were considered as influences to learning and approach. Several aspects of institutional context were presented in detail, namely the settings of studying, lecture, previous knowledge and test. Students' intentions and conceptions of learning to program and problem solving were elicited through an exploration of both the physical and personal contexts that constitute the students' problem solving histories. Two qualitatively distinct intentions were seen to underpin the students' programming strategies (see chapter 6): intention to memorize, and intention to understand. Furthermore, the analysis uncovered

two qualitatively different conceptions of programming: problem solving as “reproduction” and problem solving as “meaning making”.

In this study, a range of categories of description of learning to program emerged. The roots of many of these categories, such as meeting the requirements, learning the syntax, or understanding and assimilating, can be seen in the descriptions given by the two members of the teaching staff. Hence teachers’ views and perceptions are very important elements of students’ learning. It is to their lecturers and tutors that students look for guidance in interpreting their tasks.

8 ANALYSIS OF STUDENTS' PERFORMANCE IN EXAMINATION QUESTIONS

This chapter, presents an enhanced understanding of student learning within the subject field, learning to program, by analyzing facets of students' performance in examination questions. Sections 8.1 and 8.2 relate some information about the participants in the study: distributions of variables, such as performance in examinations, prior level of programming studied, and qualitative differences in performance in different types of questions.

Different and complementary methods of data analysis are used to explore the network of relationships among the variables of interest. These methods include correlations and cluster graphs which illustrate links among variables. The findings show striking trends and interdependencies among the variables, and reveal strong differences in students' experiences.

While the subject matter presented, institution and lecturer were the same for the surveyed pre-service students, on the one hand, and in-service students, on the other, students' orientations to learning to program were associated with a complex system of individual, social and contextual variables.

Therefore, in the next level of analysis, these interacting variables are considered. The dynamics of the students' activities are explored by looking at relationships between the different aspects of their learning and the outcomes. These links and outcomes are described qualitatively and, in some cases, are quantified.

8.1 Performance in Assessments of In-service teachers

This section looks at the influence of prior programming experience on performance in assessment to try to answer the following question:

Is prior programming experience a predictor of success for the new object-oriented language?

Descriptive statistics

There were 315 students who had enrolled for the JAVA course (Introduction to programming using JAVA) at UNISA. These teachers were from all provinces in South Africa. Of the 315 teachers 155 wrote the examination and 127 returned the background questionnaire. A Venn diagram illustrates these statistics.

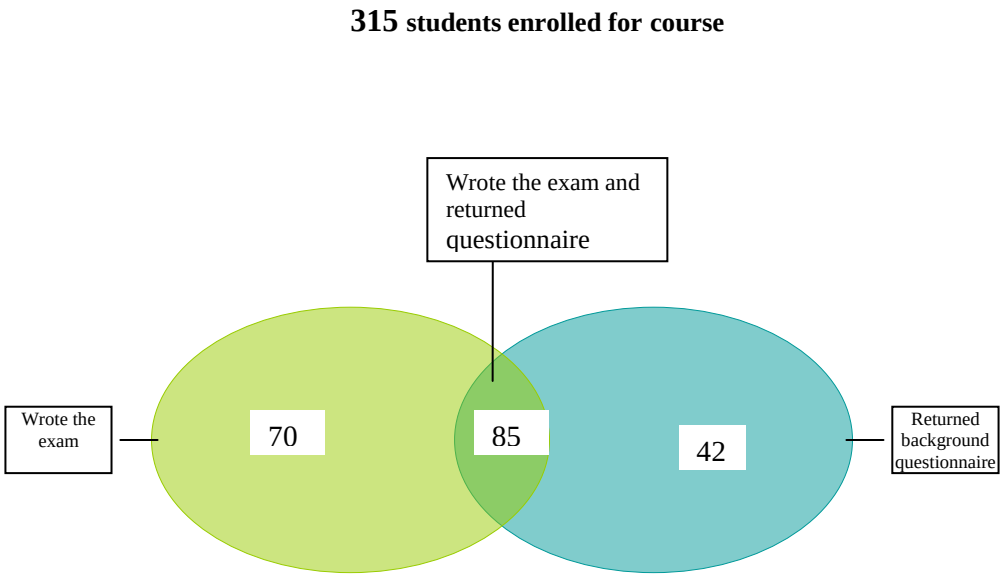


Figure 8.1 Venn diagram illustrating the statistics

The background questionnaire (see appendix B) drew responses with respect to previous knowledge of programming and programming language used, together with the number of years either teaching or using the language. Students were also asked to rate their experience of programming with a specific language, as: *limited knowledge, know the basics* and *know the language well*. Using SPSS, a statistical analysis program, the data of students' raw scores obtained in the examination together with their background knowledge (programming experience), were captured. Their previous programming knowledge was coded according to the level of experience and number of years teaching or using the language. Students' programming knowledge was, therefore, rated on a scale from 0 to 5, according to the following table:

Table 8.1 Students' programming knowledge

Know the language well – teaching / using more than 5 years	Know the language well – teaching/ using 1-4 years	Know the basics – teaching/ using more than 8 years	Know the basics -- teaching/ using 1-4 years	Know the basics -- teaching/ using 0 years	Limited knowledge of Pascal	Limited knowledge of any other language	none
5	4	4	3	2	2	1	0

Graphical representation of the raw scores obtained in the examination, and the corresponding level of programming background, are illustrated in the graph to follow:

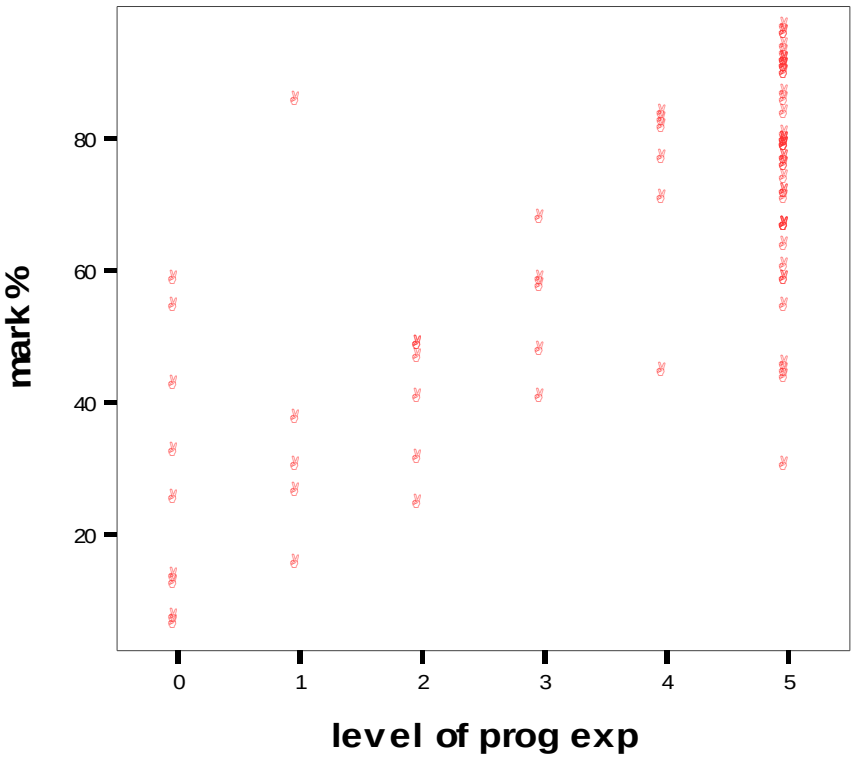


Figure 8.2 A scatterplot of the percentage mark obtained in relation to the level of programming experience.

On close examination of the graph, it is evident that there are general clusters of circles/points. The levels of programming experience rated as 3, 4 and 5 are associated with percentage marks above 50% and level of programming experience rated as 0, 1 and 2, are associated with percentage marks of below 50%. Representing the results using an interactive graph yields a clear picture. The correlation between the level of programming experience or exposure, and the average percentage mark, is reflected in the graph to follow:

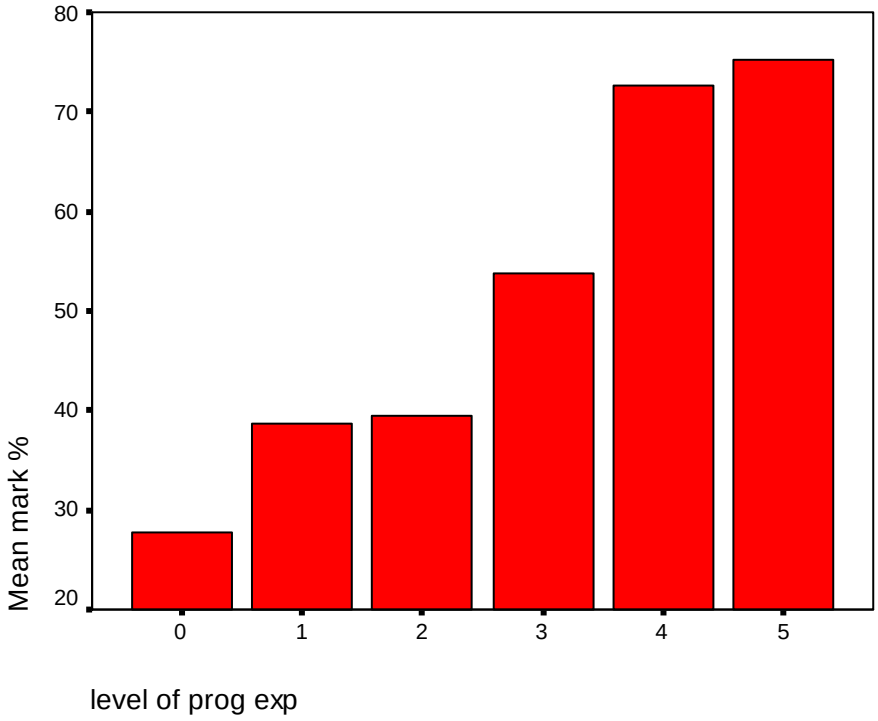


Figure 8.3 An interactive graph of prior programming experience and mean percentage obtained

The mean percentage marks of 27, 39 and 40 correspond to prior level of experiences rated as 0, 1 and 2 respectively, while the mean percentage marks of 55, 72 and 75 are associated with prior level of experiences rated as 3, 4 and 5 respectively. This result seems to indicate that previous programming experience or knowledge is a predictor of success in learning a specific new programming language, Java. On the surface, it would appear that shifting from a procedural paradigm to an objected-oriented paradigm is achieved with relative ease. However, the course was a short intense course for teachers learning through a distance learning medium. Those in-service teachers who scored well had an advantage of past programming concepts and principles as a kick start to the course. Learning the details of programming, such as variables, loops, if…then…else, arithmetic and Boolean operators were familiar to them. In this regard, all they needed to get used to was the new syntax. This result seems to fly in the face of the general literature that indicates that students with a procedural background will take a longer period of time to make a shift to the object-oriented paradigm. However, a detailed examination of the questions and data reveals more than is immediately evident. The problem task indicated breaks up the task into neat little

units that need to be implemented (see question 7 in exam paper, appendix C) or if asked to solve a complete problem (as in the case of question 2) then only a one-class program is required. One class, with several static methods, is similar to procedural programming, so that procedural[1] design rules apply. This certainly favours the students who have had strong procedural programming background knowledge.

The following quotation, from one of the journal entries of in-service teachers, indicates the dependence on hints and small chunks of code, which they have been using in their teaching. Similar characteristics are used in the examining process of the in-service teachers.

> All pupils prefer, in a test situation, if you break the problem into parts and/or give hints. They are more stressed when doing exams/tests than classwork and giving them the "starting board" to the solution, reduces the stress for them.
> I personally don't give them hints very often (I tend to "help" the Grade 10 students this way for the first half of the year) but I do break down the bigger problems into parts – more to ensure clarity of the question and to clearly set out the major objectives of the problem. (I try by the end of Grade 11 having them read a general scenario and they must do all the problem solving themselves. I feel this does develop their logic skills. Unfortunately the kids that battle with logic are then very lost, so this is often done in big classwork examples rather than exams. I don't set out the problem so much for them that they don't need to do any thinking themselves, but I do "help" with process of clarifying of the given information)

What is pertinent to point out at this stage is that, although the quotations above are from in-service teachers studying this course, they are also teachers of the subject matter, albeit of a procedural language. There is a strong possibility that they will carry this mode of teaching and assessment into their classrooms. The students are still programming in the small. It is not clear whether or not they have mastered the true OO design principles and characteristics of OOP. It is reasonable to assume, therefore, that prior programming experience is helpful for programming in the "small".

Students who did not score well (below 60), and who are associated with low levels of prior programming experience, had to make an enormous effort in order to pass. In addition to learning the basic programming structures (looping, if ….then…else, variables, etc), they had to learn the concepts of objects, classes, inheritance, constructors etc. In a short

[1] Procedural is used interchangeably with imperative.

space of time, the novice learners needed to go through a steep learning curve. Given this novel way of thinking in programming, and the problems associated with distance learning, as well as being full-time teachers, it is understandably so. Possibly, given a longer period to follow, these students may show different results. To answer the question posed above (*Is prior programming experience a predictor of success for the new object-oriented language?)* yes, prior *procedural* programming experience is a predictor of success for object-oriented *language*; however, no, prior *procedural* programming experience is not necessarily a predictor of success for object-oriented *programming*. This result suggests that the introductory course concentrates on the procedural aspects of programming in Java and the OO aspects are to a large extent neglected. Hence students' experience of the introductory course is procedural, rather than object-oriented. Their "history" (previous learning) of programming experience certainly has an influence on their learning, as indicated in chapter 7.

8.2 Secondary analysis of the performance in specific questions

8.2.1 Qualitative analysis of selected solutions of In-service teachers

This section considers the qualitative difference in performance in different questions to try to answer the following question:

> *How do students differ in their ability to answer comprehension (tracing) type questions and generation of code type of questions?*

On inspection of the in-service teachers' examination, key observations which relate directly to one or more categories of conceptions (see chapter 5), were made. Questions 2 and 5 were specifically analysed because they relate to the generation type and comprehension type question respectively. The mark obtained for questions 2 and 5 and the total score for the examination have been extracted from the data.

The following table, Table 8.2, summarizes the means and standard deviations of the surveyed students on the two questions, and on the final mark in programming. All marks are unscaled (raw) and are expressed as percentages.

Table 8.2 Descriptive statistics for surveyed students' performances on assessment tasks

	Variable Mean (%)	Standard Deviation (%)	Cases(N)
Q2	86	24	155
Q5	53	33	155
Final score	64	24	155

Positive correlations were found between students' marks obtained in question 2 and the final mark. The Pearson's correlation matrix is shown below in Table 8.3. All correlations are statistically significant ($p<0.01$).

Table 8.3: Pearson's correlation coefficients for assessment in programming questions

	2	*5*	*Total*
2	1		
5	0.584003	1	
Total	0.759031	0.900337	1

There was a strong correlation of 0.75 ($p<0.01$) between students' total scores in programming, and their mark in question 2; however, while there is a positive correlation between Q2 and Q5, it is not as strong as that between Q2 and the total score. What does this tell us? Firstly, comprehension of the answers and hence explanations required to substantiate them, is not strong enough. Question 5 required students to use "variable box diagrams and arrows" (memory diagrams[1]) to explain the problem in the given code. Hence, a thorough understanding of object-oriented concepts such as message passing, static methods and variables, inheritance and memory allocation of objects is necessary. The lack of adequate ability to use these memory diagrams is suggestive of a poor understanding of the concepts. In effect, it was a debugging exercise. While literature (see section 2.5) suggests that it might be easier to comprehend (explain, edit or modify) existing code, in this particular instance, contrary to the implication in the literature, students performed better at generation of code (Q2) than in the comprehension of code

[1] A memory diagram represents the state of objects in memory at a particular point in the execution of a program.

(Q5). The mean scores of Q2 and Q5 in Table 8.2 reflect this scenario. However, on examining the question in detail, it suggests that question (Q2) was a simple, "common" calculation which required a one-class program. It was simply an exercise and not a problem to be solved. This traditional examination question (Q2) does not test conceptual understanding of object-oriented programming (or problem solving for that matter), in that a perfectly acceptable program can be written with no more than the syntactic understanding. For convenience I will reproduce part of the questions here:

> *Question 2*
> *The Zoo charges the following entrance fees:*
> - *Entrance fee per adult = R8.50*
> - *Entrance fee per child = R4.50*
>
> *2.1 Write a program that asks the user to enter the number of adults and the number of children. Your program should then calculate and print the total entrance fee. Marks will be given for showing planning in the form of comments in your program.*
>
> *Question 5*
> *5.1 Explain why the program is not working as it should and what can be done to fix the problem.*
> *5.2 Describe in detail how you could use variable box diagrams and arrows (if need be) to explain this problem to a class of learners. Your diagram must be accompanied by a textual description. Explained what happens with and without the change as described.*

The memory diagrams are certainly useful for understanding object references in a code fragment. Even the in-service teachers, who have had sufficient experience with learning and teaching procedural programming, have found the memory diagrams very useful. This is indicated by the following quotations from the in-service teachers' journal entries:

> In my experience of teaching variables I found that learners have extreme difficulty in understanding the concepts of a variable, that is, how variables are kept in memory. I think using variable box diagrams to teach the concept of a variable is very effective. The variable box diagram is a very fresh, novel and interesting way of introducing variables to learners. When I taught the concept of a variable to my learners in the conventional way at the beginning of the year many learners grappled with the understanding of how variables are stored in memory especially when a variable takes on a new value i.e. how one value replaces another in memory. The box approach has sorted out this difficulty that learners were experiencing. I can therefore report that from this experience the variable box approach is excellent and very effective.
>
> ----------

> The novice programmer generally struggles with the concept of variables. However if introduced correctly, the foundation is set firmly & the learner soon overcomes this hurdle.
>
> ---
>
> It is also similar to tracing through programs which helps learners see exactly what is happening at each statement and understand how the program works together with seeing the exact output.

While the above quotations are with respect to procedural programming, they certainly are true for object-oriented programming as well, in which more abstract references are required. Memory diagrams, with respect to object references, still continue to be problematic, which concurs with the reviewed literature in chapter 2 (section 2.6).

8.2.2 Qualitative analysis of selected solutions of Pre-service teachers

In order to better understand some of the outcomes and results in the previous section, a similar analysis of the solutions of the pre-service teachers was performed. Questions 4 and 6 were chosen for similar reasons stated in the previous section. The marks obtained for questions 4 and 6, and the total score for the examination, have been extracted from the data of the pre-service teachers' examination. The following table, Table 8.4, summarizes the means and standard deviations of the surveyed students on the two questions and on the final mark in programming. All marks are unscaled (raw) and are expressed as percentages:

Table 8.4 Descriptive Statistics

	N	Mean	Std. Deviation
Q4%	12	46.58	29.944
Q6%	12	68.58	31.064
TOTAL	12	59.08	21.360
Valid N (listwise)	12		

A pattern similar to that of the in-service teachers has emerged. Students performed better in the question that required solving a problem, rather than the comprehension required of question 4. Students seem to find the reading and understanding of code more difficult, as opposed to writing a complete program. A possible explanation for this result could be that students were given this problem before (possibly as a class exercise), which would make it

an exercise to do rather than a problem to be solved. Another possible explanation is that marks are given for method, even if the problem was not solved completely.

The Pearson's correlation matrix is shown below, in Table 8.5.

Table 8.5 Pearson's correlation

	Q4%	*Q6%*	*Total*
Q4%	1		
Q6%	0.01172	1	
Total	0.557502	0.811264	1

For the above statistical test the null hypothesis is:

$H_{0:}$ A student doing well in Q4 (comprehension type question) does not have a positive correlation with the student doing well in Q6 (writing a program to solve a problem).
The alternative hypothesis is:

H_a : A student doing well in Q4 does have a positive correlation with the student doing well in Q6.

From table 8.5, it is clear that the null hypothesis is accepted.

The outcome space (see chapter 5) suggests that reading and understanding programming code and problem solving are complementary categories of conceptions, which seems to reject the null hypothesis. Possible explanations of the differences in performances in questions 4 and 6 are: (1) Students may have been facing a learning problem, and (2) the instructor had given the students an example to study or solve, which was a complete solution to a similar problem to Q6 (as was done in the preparation for tests; see interviews in Appendix D). It is also possible for a student with a partial understanding to be able to write correct code up to a certain level, without completely understanding the code. Marks are, therefore, accumulated as they are allocated for method. Therefore, it would appear that an assessment that involves generating a program may not be a true reflection of students' competence in programming, if students were exposed to the problem before, or if assessment techniques used, encouraged accumulation of marks for method.

8.3 Summary

In this chapter, the performance in certain questions of the assessment has been examined in depth. Important trends were found. Firstly, in-service teachers with prior programming experience performed better than those without prior programming experience. This is an obvious notion. However, the examination of the data revealed that the questions were procedurally posed and, hence, those with prior programming experience had a head start to programming development. Secondly, questions that required an understanding of the program execution (Q5) were more poorly answered than those that required writing a simple one-class program (Q2) (procedural by nature). The implication of this is that understanding of memory diagrams, or rather representation of programs in memory for the object-oriented programs, were low and writing a simple one-class program that is procedural in nature, is more easily accomplished. Similarly, it was found that for, pre-service teachers, questions involving understanding and tracing code (Q4) were more poorly answered than those that required solving a problem (Q6; generating code). This result has similar implications as the result for the in-service teachers. Moreover, familiarity of problem tasks allows one to perform better in assessments.

These findings and those of chapters 5, 6 and 7 have implications for teaching, which will be discussed in the next chapter.

9 SYNTHESIZING THE STUDY: INSIGHTS, DISCUSSION AND REVIEW

This chapter focuses on relating the findings to the research questions and sub-conclusions.

The goals in this research have been twofold. The first goal was to raise issues about learning and teaching programming, in particular, object-oriented programming education at university and high school. As explained in chapter two, there is considerable research on teaching programming internationally. This reflects concerns in the community of programming educators about setting objectives for introductory programming education and implementing these objectives in appropriate ways. In this study, the focus has been on the other side of the coin: students' actions, experiences and their awareness of their actions and experiences in learning to program. This focus is necessary in order to see if objectives of programming education are being met. Secondly, a major contribution of any research is the transferable theory that emerges from it (Miles & Huberman, 1984). Research cannot be prescriptive, but the explanatory power of a theory is useful. In this study, some of the important ideas of Vygotsky and Leont'ev have been applied and extended (Vygotsky, 1962; 1978; Leont'ev, 1978, 1981). The phenomenographic method is applied extensively; and the theory developed to understand a system of activity (Leont'ev, 1981; Engeström, 1993) encompassing teaching and learning programming within the social, cultural and historical environment of a traditional (both face-to-face and distance learning) university have been employed.

Section 9.1 summarizes and illustrates the theoretical ideas developed in this thesis. A simplified representation of an activity system is presented in section 9.1.1, which discusses aspects of the activity system, keeping in mind the dynamic and interactive

dimensions and transformations of learning programming. In section 9.1.2, teaching and learning programming is viewed in the light of the theoretical framework developed. Section 9.2 is a discussion of the results and methodology and an exploration of issues arising from these, which suggest directions for further research. Firstly, some important findings that emerged from the analysis are summarized in section 9.2.1. In 9.2.2, the insights provided by the research methods are reviewed. In 9.3 some important, broad issues and implications concerning programming education, in general, and teaching practice, are considered. Some limitations with respect to the methods and recommendations for further study are discussed in sections 9.4 and 9.5 respectively. In section 9.6, I review the research. That is, the research is analysed in terms of the dual aims described above. I conclude by raising questions as to what programming activities are appropriate in teacher education.

9.1 Overview of the research framework

The research framework presented in this investigation uses phenomeography to determine the categories of description of the phenomenon, learning to program, and activity theory was used to inform the decision to investigate the interactions between the learners and the learning context, a component of the activity system. Several settings within the institutional context were then identified on the basis of my teaching experience. Having selected these settings, a phenomenographic approach was then used to analyse the influence of the settings and the relations between the personal and physical contexts on learning to program for each setting.

In the study a diverse set of research methods were used to try and reveal the worlds of the students. The combination of qualitative description, phenomenographic analysis, elements of activity theory and quantitative summaries, provided different lenses with which to view the findings. The aims were:

- to explore patterns in the data as a whole, as well as to investigate collective perceptions and experiences.

- to view the data systemically, as is consistent with the theoretical framework building on the work of Vygotsky (1978) and Leont'ev (1981). For this reason, teachers as well as students were interviewed. This also involved paying attention to the dynamics of the relationship between individuals and context. In much research on programming education, the context, taken as the background (unchanging and uniform) is ignored.
- to use research methods acceptable to the research community, while acknowledging the role of my own beliefs, interpretations and values. Hence, the findings represent my interpretations of the data, rather than the output of a computer or an "objective reality". I concur with Anderberg (1973, p. 21) that:

> The analyst's research objectives permeate the entire investigation. They motivate the enterprise and effectively shape the evaluation of observations and explanation of facts as perceived.

The value of the methods of analysis is justified by its appropriateness for exploring the data, and by the value of the results brought out. The different methods and research tools used led to the discovery of different aspects of the phenomenon under investigation and suggested further avenues for exploration.

9.1.1 The activity system

The use of phenomenography is "extended from focusing mainly on the variations in the relations between the student and the object of his/her studies (in this case learning to program), to include also the variations in relations between the students and phenomena contextual to the study object" (Berglund, 2002b). While phenomenographic analysis allows the identification of aspects of the understanding of the phenomenon, from the students' perspective (thus helping educators to introduce students to new experiences of learning to program) elements from activity theory are used to guide the researcher's understanding of the learning situation.

A simplified representation of this system (adapted from Gordon, 1998) is shown on page 175 in Figure 9.1. This summarises the ideas raised throughout this study. The dynamic component of the system is the student's activity, that is, how he/she engages with the learning task. Activity is shaped by how a student orients him/herself with respect to the

learning task, his/her goals and needs and the tools and constraints accompanying the task. It also reforms these anew as the student's actions, with his/her ensuing outcomes, unfold. On one plane, a student's actions relate to his/her purposes and to the resources available, as well as to the constraints of the task. In this way, through activity, goals are linked to specific conditions. On another plane, through activity, orientations are connected to outcomes. For example, a pre-service student, S6 expressed a positive orientation to learning to program, indicated by the following quotation:

R: *What do you understand by the term programming? What do you perceive programming to be?*
S6: *The first thing that comes to my mind… is the idea of a problem and programming is the way in which we go about solving a particular problem…We use lots and lots of tools.. use various tools to do that.*
R: *Can you write a program that works?*
S6: *Yes I can.*
R: *How do you know?*
S6: *Very often we were given exercises to do on a regular basis. Lot of the work we do is on our own and (pause) so I do know that I can write because the exercises we do are similar.*
R: *Can you write a good program?*
S6: *I'd like to think so.*
R: *How do you know it's a good program?*
S6: *I suppose just looking back to where I've come from with other programs to now doing program which will work out various times that different vehicles will take to reach a particular destination. I suppose just looking at that it's a big step.*
R: *How do you see your current ability to program?*
S6: *I'm quite optimistic I suppose, especially considering the fact that we moving into a different style of programming. So whereas there are quite a few similarities there are few concepts I'm just starting to come to terms with.*

S6 achieved success both in subjective terms (personal triumph at overcoming difficulties) and by the criterion of performance in assessments. Activity is regulated within social and cultural contexts and also contributes to the transformations of these learning contexts. These links, between acting individuals and the world surrounding them, are organised through interactions, including verbal communication with other people, such as colleagues and teachers, and through objects, such as books and computers. Contextual elements are relatively stable aspects of the activity system, compared to individual actions.

Institutionalised practices, such as the writing of examinations at universities, seem to reproduce similar actions and outcomes in a persistent and seemingly unchanging practice.

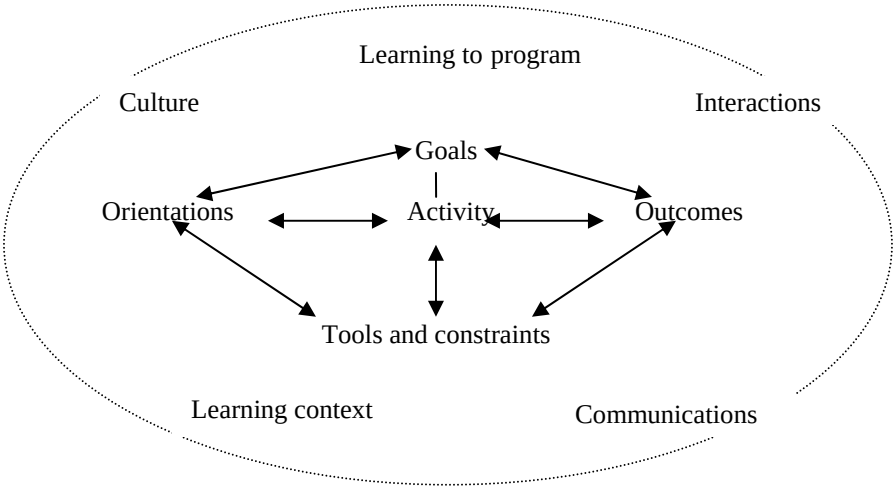


Figure 9.1 An activity system with the student who is learning to program as the focus

Only a frame or snapshot of the activity system can be diagrammatically represented. Figure 9.1 depicts relationships among orientations, activity, goals, tools and constraints, outcomes and contextual aspects. These contextual aspects include personal interactions and communications, the institutional setting, the surrounding culture and the task of learning to program itself. The relationships among all these elements are mobile and undergo transformations leading to the next phase of the activity system. The activity system encompassing teaching and learning to program is a complex and fluid formation. It may be transformed by the actions of individual lecturers or by groups of students, for example, acting through a class representative. It is transformed by new resources, such as the availability of technology, or by changes in departmental or institutional practices, or by new national education policies, such as the introduction of a new programming language. Further, it is both dependent on, and transforms, other activity systems in the social sphere. For example, the curriculum, teaching methods and assessment for programming are shaped by, and also shape, the activity system revolving around new demands and the new language.

9.1.2 The activity of Teaching and Learning to program

The specifics of one activity system are now considered: the system incorporating the teaching and learning of programming. Some perspectives on teaching and learning to program, suggested by the theoretical framework, are provided.

In teaching programming at university, traditional methods take for granted that students should learn certain topics at a specified level using prescribed techniques without questioning the framework within which this learning takes place. The student's attention, it is assumed, is focused on the content to be learned, rather than the learning situation. This study, drawing on aspects of activity theory, shows that, contrary to these suppositions, students' orientations to learning are mediated by their evaluations of the context and personal sense (Leont'ev, 1978) of the educational practices in which they take part. Further, the outcome of these evaluations may not concur with the objective of university education which is, surely, high quality learning.

Two fundamental principles of activity theory are relevant in addressing these issues. Firstly, each individual has a history of learning experiences which will influence how each learner acts in a given learning situation. Secondly, the learning task is situated in the university setting, with its own particular traditions and practices. The problems individuals have to solve are structured within the bounds of this setting; individuals become socialised as they act within this arena.

The evidence from this investigation suggests that some students behave in ways which educators do not value, but which are functions of practices in higher education. Students have diverse needs and constraints which both form, and are transformed, by their perceptions of the learning task and their actions in carrying out that task.

To many of the participants in the study, learning to program was evidently about accumulating knowledge for examination purposes (meeting the requirement) irrelevant to their personal interests and concerns. As lecturer T2 said:

> Like when I was explaining the solution, they wanted an explicit solution. I expected that when you explain the solution they would take it down, but they wanted an explicit answer. Kind of rote-learning, they are students who are going to become educators in about 4 years, they should be able to understand and adapt, but for them it is like wanting to get through the examination. They seem to have a short term goal.

Within this context, learning to program is reduced to performance on assessment tasks. These assessment tasks are generally imposed on the students with little or no room for negotiation.

Nevertheless, for some of the participants, experiencing learning to program as indicated by the categories of description, understanding and assimilation, and problem solving, appeared to be overwhelmingly concerned with mastering concepts, techniques and skills. For these participants grappling with the task of learning to program led to increased confidence, enhanced insight into scientific thinking and personal empowerment (see section 5.2).

If, as the activity framework suggests, understanding is constituted through personal actions, then the quality of a student's programming understanding is related to the nature of these actions and the goals directing them. The activity framework suggests that the process of learning is important in defining whether that learning will have a lasting impact on the students' views of the world. Assessments of the sort written by these students, however, measure a product, what students "know" as indicated by written tests and examinations. This knowledge product, furthermore, may not reflect a student's ability to communicate the knowledge, apply it in different situations or even remember it the following year. Activity theory therefore provides useful concepts to understand better the dynamics of students' learning and, hence, it provides directions for the improvement of teaching. From the perspective of activity theory, the role of the teacher is to establish an environment appropriate for educational activities. In this framework, teaching programming, rather than being the process of transferring a body of syntactical knowledge, is the guiding of students to view their learning as relevant and meaningful. For this guidance to be effective, the implicit assumptions underlying the content, presentation and evaluation of the course need to be examined. More specifically, the assessment

methods and tasks serve as an indication of what is expected of the students from the educator. However, while the intention of the educator may be to ensure that a deep interpretation of the material is achieved by most students, this intention may not necessarily be the same for students. Hence careful scrutiny of the assessment *from the perspective of the student* is needed to achieve this difficult aim.

Those currently organising and reforming, or teaching the programming curriculum, have inherited practices from their predecessors. Such practices are not lightly tossed aside. The research shows that most of us who teach continue to use tried-and-true methods. (The adage, "old habits die hard" applies in this case). Many teachers (in-service) and lecturers at higher educational institutions believe that because they learned procedural programming before they learned object-oriented programming that is the way in which everyone must learn it, as that is the way in which concepts are structured. Educators may also act in accordance with an academic system which does not always match their personal planes (Semenov, 1978), i.e. their evaluations of the meaning of teaching and learning programming.

9.2 Discussion

The discussion will focus on two aspects: a summary and discussion of the key findings, and insights provided by the research tools.

9.2.1 Summary and Discussion of the Key Findings

The research framework, which integrates phenomenographic methods and principles of activity theory, developed in this study, explains that students bring to a learning task perceptions, assumptions and strategies that are grounded in their previous experiences. In particular, prior programming experience provides the basis for learning a new language.
In chapters 5 and 6, the variation in the ways in which students experience learning to program and programming (problem solving) were described. The results of the experiences of learning to program were presented in an outcome space. Chapter 7 dealt

with aspects of the learning context (institutional, personal and teacher perceptions) that influence student learning, while chapter 8 shed light on some important issues (prior programming knowledge, performance in examination questions and nature of assessments), bearing in mind that learning an object-oriented language formed the backdrop to this entire study. In the overall investigation, it was found that there was no clear distinction between in-service and pre-service teachers; however, a clear distinction emerged between those who had previous programming knowledge and those without it. A summary of the findings is explicated below.

Finding #1

There are a "limited number of qualitatively different ways" (Marton & Marton, 1997) in which students experience learning to program.

These different ways of experiencing learning to program, which are referred to as categories of description, that were constituted through an "iteration" process can be regarded as a discovery and are unique in a number of ways. Firstly, the descriptions from the interview data and journal entries were obtained from a unique population of pre-service and in-service teachers in KwaZulu-Natal and South Africa respectively. Secondly, the results are unique in that they contribute to filling a gap in the research literature regarding students' conceptions of learning to program and programming using an object-oriented language.

Finding #2

There appeared to be a bigger difference between those with prior programming knowledge and those without than between pre- and in-service teachers, which was not expected. This was identified in investigating the effect of the setting of previous knowledge on learning to program as revealed in section 7.2.1. Hence, the limited number of ways of experiencing learning to program can be further distinguished between students with prior programming experience (in most cases in-service teachers), and those without prior programming experience (in most cases pre-service students) as indicated below:

Students with prior experience: Meeting the requirements, *learning by comparison*, understanding and assimilating, problem solving and programming in the large.

Students without prior experience: Meeting the requirements, *learning the syntax*, understanding and assimilating, problem solving and programming in the large (see table 5.1).

Finding #3

The set of categories of description constituted for the phenomenon, learning to program, formed an "outcome space" (see figure 5.2). This outcome space reflects an evolving awareness of learning to program. If students experienced programming at the lowest level (meeting the requirements) of the outcome space, it is hoped that they will move quickly to higher levels of experience, in particular the problem solving category.

Each category of description is associated with a deep or surface approach (see table 5.1). However, it must be stated that a student may not be classified as using either of the approaches exclusively. The research only describes the relative prominence of each approach in each category and in studying a student.

Finding #4

Programming is perceived by most students as a problem to be solved (even if they do not experience learning to program in the problem solving category of description, at the beginning of their learning paths). The literature concurs with this perception and suggests that programming inherently involves problem solving.

Finding #5

There are two ways in which students experience programming, namely approach A, which involves scanning, planning, translating and reinterpretation and approach B, which involves scanning and translating.

Approach A is characteristic of a deep approach to learning, while approach B is characteristic of a surface approach to learning (refer to table 3.1 for the characteristics of

deep and surface learning). However, Biggs (1993) asserts that what is specifically meant by deep and surface approaches in any instance depends on the context, the task and the individual's encoding of both. His line of reasoning is related to the association of rote learning with surface approach. In this respect, he distinguishes between the students' resolve to "reproduce without understanding" and the resolve to "ensure accurate recall of already understood information" (Biggs, 1993, p.7). While the former resolve is associated with the surface approach, the latter, depending on the context, could represent a deep approach. The approach A and approach B also exhibit features of expert and novice learners, as indicated in table 2.1.

The results of the analyses from chapters 5 and 6 confirmed that students experienced the phenomena of learning to program and solving problems in a limited number of qualitatively different ways (a fundamental assumption in phenomenography).
However, it must be noted that the set of variation in ways of experiencing the phenomena is "finite, but not closed" and that other studies could reveal further new variations. Nevertheless, previous phenomenographic studies show that such new variations are not likely to be extensive in any way (Marton & Booth, 1997).

Finding #6

The planning by students with prior procedural programming experience indicates a procedural way of solving a problem. The throwback to the procedural way of thinking influences their planning stages. This can have serious implications for teaching. The planning by those without prior programming experience indicates (even if at an inaccurate level) identification of objects and classes.

Finding #7

The learning context (institutional, personal and teacher perceptions) influences what students do when learning to program and solving problems. Their intentions and conceptions, as well as the various settings (such as study, lecture, previous experience, and testing) influence their learning processes.

In phenomenography, the categories of description are typically lifted out of their contexts. However, the learning context, in particular the settings of the learning situation are important if the categories of description are going to be meaningful to teachers and students. Therefore in order to extend the theory developed using phenomenography, which does not consider the learning context, activity theory was used to enable me to investigate the influence of the learning context. The way in which students use the learning context (institutional and personal) reflects their enrolment in the discipline. These intentions and conceptions, in turn, influence their processes of learning and solving problems.

Finding #8

Prior *procedural* programming experience is a predictor of success for learning the syntax and rules of an object-oriented l*anguage*; however, it is not necessarily a predictor of success for *programming* in an object-oriented way. A possible explanation for this result may be that students' problem solving experience of the introductory course in Java was procedural, rather than object-oriented.

Finding #9

Creating a program is more easily achieved than comprehension (tracing or correcting a program) only if the program is a one-class program. This is significant because, throughout the study, it has emerged that most teachers perceived that writing a program is synonymous to solving a problem, which seems to pose the most difficulty. This is consistent with the research on programming and solving problems (outlined in sections 2.8 and 2.9). A simple calculation that does not require the creation of objects is not really a problem but a classroom exercise.

The patterns that have emerged from the study are consistent with previous research. They may also match the intuition of some educators. However, one should be cautious in generalizing the results from the particular participants and setting.

9.2.2 Some Insights Provided by the Research Strategies

9.2.2.1 The Application of Phenomenographic Methods

Traditional phenomenography, in Marton's (1986) terms, has the aim of constructing categories of conception (explained in section 3.1). The questions asked required not only developing the categories themselves, but also finding dimensions of variation within each category, such as *learning approach, programming language* and *learning motivation*. This extension of phenomenographic methods led to interesting results. The findings showed a relationship between the categories of conception and themes, such as generating a program, comprehension of programs and prior programming experience (as highlighted in chapter 8). This suggested a link between conceptions of learning to program and approaches to learning it, a powerful result not available through direct observation.

Opportunities for finding and validating alternative and deeper explanations were also provided by quantitative analyses. This will be discussed below.

9.2.2.2 Interpretation of quantitative Results

Important quantitative data have been reported in this study, for example, the distributions of students' prior programming experiences and their performance in assessments, and descriptive statistics, such as means on the assessments of and relationships among variables. Analyses of these data are tools for suggestion and discovery of other results. They have added to the understanding of students' orientations to learn to program and their ways of experiencing it. In other words, I consider that the quantitative data complement the qualitative data in forming the hypotheses and heuristics on which the interpretations and explanations were founded. These techniques allowed for different planes of interpretation, rather than producing more absolute or definitive conclusions. For example, the cluster analysis suggested that there were different dimensions, such as prior programming experience and performance on assessment tasks.

9.2.2.3 Interviews and Conversations

I conducted a number of interviews with participants in this research. In each case I tried to establish a rapport with the person being interviewed, in order to get close to them which

would allow me to interpret correctly what they were saying to me. Nevertheless, not all interviews were on the same level. Some students were eager to express their perceptions. Conversations with lecturing staff also contributed to my understanding of their background.

9.3 Implication for teaching practice

The results of this study suggest several pedagogical issues to consider when teaching programming to novices. Programming, as suggested in the literature and observed in this study, is indeed what Dijkstra (1989) refers to as an "educational novelty" in which the students' tried and tested learning styles do not work when applied to programming (Jenkins, 2002). Dijkstra argues that learning is a gradual process of transforming the "novel into the familiar". An important feature of programming, one that reinforces Dijkstra's point, is that it is problem solving intensive and precision intensive, in that it requires a significant amount of effort for a small outcome (see discussion in section 2.8).

An important implication of the study of learning and teaching programming is that instructors must learn to see teaching as a process aimed at changing student conceptions. In order for this to happen, teachers must be aware of the different conceptions (categories of description) that students may have. This means that the approach to teaching must change. However, changing an approach to teaching requires firstly the knowledge that other approaches are possible and secondly, it requires reflective practitioners, i.e. an acknowledgement of the less useful strategies and a willingness to change from old habits.

Knowing that teachers teach as they were taught (even at a subconscious level), it became clear that if the object-oriented approach to programming is not infused in instruction during their practicum experiences, pre-service teachers will not graduate with the ability to create true object-oriented programs in the learning environment; and this cycle of teaching the way one was taught will be perpetuated. Invariably, the approach to programming in introductory courses is dependent on, and influenced by, the instructor's approach and the instructor, in turn, is influenced by his past programming and learning

experience. If the goal is to learn OOP then teachers should use appropriate teaching strategies to teach OOP (i.e. emphasis should be on identifying and creating classes and objects first) and avoid using a procedural approach to teach programming, even if students undergo a longer learning curve before they become competent programmers.

Distance education students (in-service teachers), in particular, often have problems as the difficulty of the course content is compounded by the problems of isolation from other students and their tutors. Hence, the teaching of distance students in introductory programming courses presents a special challenge. The reduced number of opportunities for support (both verbal and visual) for distance education students, as opposed to those receiving face-to-face instruction, makes it difficult for students to develop programming-specific problem solving skills. While the syntax of a language can be learnt from books and sample programs, the design (problem solving) skills are more difficult to acquire from written materials. There is strong evidence in the data to support face-to-face interaction when learning to program. This is an area that needs to be researched further.

The desirable categories of description of learning to program that should be developed, would surely be those that appear in the higher levels represented in the outcome space, i.e. those that encourage a deep approach to learning. Educators must make students aware of the different categories available to them and help them expand their horizons. The individual categories of description raise particular implications for teaching and learning. A few possibilities will be discussed.

Category 1- meeting the requirements

The way marks are allocated throughout the course may have an influence on student performance. Frequent and smaller assessment units may encourage students to learn and move from this conception to one that is more desirable. However, this movement is dependent on the nature of the assessment tasks. The assessment plays an important role in shaping the curriculum and methods of instruction. “The spirit and style of student assessment defines the de facto curriculum”, (Rowntree, 1987). Ironically, the assessment is generally in the teacher’s control and he/ she can ensure that this aspect of the learning

context points students clearly in the direction of the kind of changes in learning that are demanded. If object-oriented programming is the goal, then the assessment must reflect this goal.

Category 2 learning the syntax/ learning by comparison

It frequently happens that students see learning the syntax of a language as learning to program, which can be harmful. The teaching staff must encourage students to see the learning of syntax as only one component of the bigger picture, which is solving problems using the language.

In the case of the in-service teachers who had prior programming experience, learning by comparison is only effective up to a point. The limitation of this category (approach) became evident when solving problems in an OO way. Instructors can help by giving problems that are appropriate for OOP, so that identifying objects and making use of classes become the main focus.

Category 3 understanding and assimilating

This way of experiencing learning to program is a desirable way, one that provides a solid grounding in programming, which is really a prerequisite to solving problems using the language. Visual tools, such as memory diagrams, are invaluable aids in helping students grasp the meaning and execution of a program. This is clearly evidenced in the discussion in section 8.3. Reading code and understanding the execution of a program is surely an important technique to inculcate, even before the ability to write a good program. Many programming courses are taught in a way such that students start writing code without first (or, worst still without ever) reading any code. This parallels the teaching of writing before reading in preschoolers, or asking students to write essays without their ever having read one. Students will do well in studying programs written by others. It is important, however, that all examples students read are well-written and are worth being copied for style and cues.

Category 4 problem solving

When students see learning to program in this way, teachers need to develop the sequencing of topics according to problems given. Stated differently, a problem should be posed, discussed in order to clarify the problem, and students should be shown how the current programming constructs may be inadequate and then introduce new programming constructs to meet the need. This is in keeping with the pedagogic model (problem solving) suggested by Kaasbøll (1998) (see section 2.9), and is strongly supported by Rogalski and Samurcay (1990). In short, problem tasks should therefore be designed (or chosen) so that students progressively learn new constructs.

Category 5 programming in the large

To encourage this kind of experience of learning to program, teachers must create opportunities for students to create programs that are of value, possibly, to a school, computer science department, or to an individual's business. Bearing in mind that these students are introductory students, they will have to reach a certain level of maturity in programming in order to engage in production programming[1] and extreme programming (which might be ideal).

Given the many different experiences and learning styles that students adopt, enormous pressure is placed on the instructor, for effective teaching to take place. This study, which dealt with in-service and pre-service teachers, is particularly pertinent, because of the pedagogical implications it has. The content knowledge, which, in the discipline of programming, generally refers to the reasoning involved, problem solving and syntax, are only part of the picture for the in-service and pre-service teachers. Computer science (in particular programming), by its very nature, is less about facts, theorems, or theories, than about reasoning. The heart of the discipline, programming, therefore, amounts to modelling, structuring, and solving problems. These skills are difficult to teach and to learn. While it is common practice to make pre and in-service teachers aware of common and effective teaching strategies, referred to as pedagogical knowledge, it is still not

[1] Production programming involves creating or modifying a software product to meet the needs of real customers. To teach such a course a methodology called extreme programming is used to develop an open source project with real customers

enough. Shulman explains another kind of knowledge that is needed, pedagogical content knowledge, in the quotation below:

> Pedagogical content knowledge also includes an understanding of what makes the learning of specific topics easy or difficult: the conceptions and preconceptions that students of different ages and backgrounds bring with them to the learning of those most frequently taught topics and lessons. If preconceptions are misconceptions, which they so often are, teachers need knowledge of the strategies most likely to be fruitful in reorganizing the understanding of learners, because those learners are unlikely to appear before as blank slates. (Shulman, 1986)

9.4 Limitations of the study

Limitations of the methods used were identified. As for any qualitative interpretation, analysis of data was extremely painstaking and time-consuming. Moreover, this method assumes that students understand the questions and are able, and willing, to express their thoughts in writing. Some students gave too little information for their responses to be classified. Further, students may not have responded sincerely.

Diverse perceptions, evaluations and actions taken were evident in the journal entries of the students. However, the complex relationships among *individual* students' histories, needs, goals and actions, all of which are socioculturally embedded, were not captured directly. Hence, corresponding results in tests and examinations could not be followed for individual students. These limitations were unavoidable under the conditions in which the research was carried out and with the resources available to me.

Within the activity of learning and teaching programming, student activities, with their accompanying goals unfold continuously over time. The study, however, could only capture students' perceptions and actions at particular points in time. In particular, the study was a snapshot of an activity system and so cannot capture the movement and transformations that undoubtedly took place for individuals. I am therefore unable to assess the directions of these transformations.

9.5 Recommendations for further study

The motivation for this research was to determine and understand the ways that students experience learning introductory, object-oriented programming, given the major difficulties that students experience in learning to program in general (see chapter 1 and 2). The following points represent some of the key areas for potential future research.

- Conducting a longitudinal study by following the same cohort as they undertake successive units, and seeing how students change over time in their approach to programming, and how the outcomes of learning to program impinge on their practice as in-service teachers.
- Conducting pilot studies in secondary schools to investigate the teaching of programming using an object-oriented language.
- Investigating the variation in ways in which high school Information Technology (the new subject name for secondary schools) students experience learning to program.

9.6 Conclusion

The findings from this study were summarized in section 9.2. From the findings, it is apparent that most of the students, whether pre- or in-service teachers find programming difficult at the introductory level, which is consistent with the literature. Their style and approach to learning to program is a function of how they experience it. Prior programming knowledge, in particular of the procedural kind affects their learning true, object-oriented programming. The paradox is that the instructors themselves have had a difficult time teaching the new paradigm. This is possibly so because many instructors transfer to teaching Java from teaching procedural languages, such as Pascal. Their procedural background may have a negative influence on their teaching an OO language. This influence has been suggested in the results found in chapter 6. The implication of this influence is that the introductory module to programming in Java has a strong procedural slant. The transition to the object-oriented paradigm has been difficult, and continues to be

difficult for many instructors, because it is driven by a language choice and not by an understanding of problem solving with object-orientation (differences outlined in chapter 2, section 2.1). It may be that the distributed nature of OO programs and emphasis on abstraction places great demands on novices. Therefore, more time may need to be allotted for novices to reach a level of skill comparable to procedural novices.

It is natural that the teaching of object-oriented programming will be perceived as more difficult, because there is still a lack of experience with teaching OO to introductory students. *However, teacher educators must realize that using Java implies adopting a different programming paradigm from what they have been used to*. This aspect needs to be stressed in our teaching approach. Support materials, methodology and appropriate problems for OO are less mature than the equivalent for procedural programs. Development of suitable problems for OO and methods for teaching an objects–first approach need to be further researched.

From a pragmatic point of view, this research shifts the task of the programming instructor from a preoccupation with how best to present information to focusing on what students are actually doing and how their goals and conceptions relate to their actions. The investigation raises questions about the central premises behind teaching and learning to program at schools and university. That is, what aspects of programming should be taught and how are they experienced by students? The research has shown that studying students' experiences of learning to program and integrating the results with their personal and physical contexts may contribute to gaining a greater degree of clarity and conceptual understanding of learning to program. Educators may, therefore, be able to improve students' performance by explicitly teaching them techniques in the object-oriented approach to programming, and by helping them to learn the semantic constructs of the language. Stated differently, the emphasis in this study is the recognition that teachers must develop objected-oriented programming knowledge for teaching as well.

The main *theoretical implication of this study* lies in its blending of the perspectives of phenomenography and activity theory within the naturalistic paradigm. The emphasis of

phenomenography on the variation of the experience of learning has clear benefits, to the extent that it has enabled educational research to move beyond the psychology of the individual learner.

Activity theory shares with phenomenography its view of spatio-temporal distribution of knowing, but its systems of activity and interaction with the learning context are particularly useful in studies (such as the present one) concerned with the influence of the learning context. In this study, elements of activity theory have been used to complement phenomenography by overcoming phenomenography's exclusion of contextual factors, a perspective that has value for other similar studies.

Finally, learning and learning to teach OOP has been shown to be a complex task. The challenge, therefore for programming educators is to understand student conceptions of learning to program and problem solving, and the learning context as described in the study in order to develop pedagogical content knowledge for OOP. The importance of recognizing typical paths of student learning in designing instructional strategies cannot be over-emphasized. If we are to be successful in promoting greater conceptual understanding in our students, we must utilize the best ideas from research-based teaching strategies.

A common but important pedagogical philosophy is *the more we know about what students know, the better we can teach them.*

REFERENCES

Adawi, T., Berglund, A., Booth, S. & Ingerman, A. (2002). On Context in phenomenographic research on understanding heat and temperature. Revised paper presented at *EARLI 2001, Fribourg, Switzerland* and submitted to Learning and Instruction.

Adelson, B. & Soloway, E. (1985). The role of domain experience in software design, *IEEE Transactions on Software Engineering*, 11(11): 1351-1360.

Anderberg, M. (1973). *Cluster Analysis for Applications*. New York: Academic Press.

Ausubel, D. (1968). *Educational psychology*. New York: Holt, Rinehart & Winston.

Barker, J. (2005). *Beginning Java Objects. From Concepts to Code*. 2nd edition. New York. Springer-Verlag.

Bergin, J. & Winder, R. (2000). Understanding Object-Oriented Programming. Accessed: May, 2004. http://csis.pace.edu/~bergin/patterns/ppoop.html.

Bergin, S. & Reilly, R. (2005). Programming: Factors that influence Success. *SIGCSE '05* February 23-27. St.Louis, Missouri, USA.

Berglund, A. (2002b). Learning computer systems in a distributed course: Problematizing content and context. Paper presented at the *European Association for Research into Learning and Instruction, SIG, "Current issues in Phenomenography"*, Canberra, Australia.

Bettencourt, A. (1993). The construction of knowledge: A radical constructivist view. In K. Tobin (Ed.), *The Practice of Constructivism in Science Education*. Hillsdale, New Jersey: Lawrence Erlbaum.

Biggs, J. (1993). What do inventories of students' learning processes really measure? A theoretical review and clarification. *British Journal of Educational Psychology*, 63(1), 3-19.

Biggs, J. (1999). *Teaching for Quality outcomes at university: What the student does.* Buckingham, UK: Society for Research into Higher Education (SRHE) and Open University Press.

Bishop-Clark, C. (1995). Cognitive style, personality, and computer programming.

Computers in Human Behaviour, *11*, 241-260.

Bloom, B. S. (1971). *Taxonomy of educational objectives: the classification of educational goals: handbook 1: Cognitive domain*. New York: David Mckay Co.

Blumer, H. (1969). *Symbolic interactionism: Perspective and method*. Englewood Cliffs: Prentice Hall International.

Booth, S. (1992). *Learning to program: A phenomenographic perspective*. (Göteborg studies in educational sciences, 89). Göteborg: Acta Universitatis Gothoburgensis.

Booth, S. (1997). On phenomenography, learning and teaching. *Higher Education Research & Development*, *16*(2): 135-158.

Bowden, J. & Marton, F. (1998). *The University of Learning: Beyond quality and competence in higher education*. London: Kogan Page.

Bransford, J.D., Brown, A.L. & Cooking, R.R. (2000). *How people learn: brain, mind, expeience, and school*, expanded edition. Washington: National Academy Press.

Brooks, R.E. (1977). Towards a theory of the cognitive processes in computer Programming. *International Journal of Man-Machine Studies, 9*, 737-751.

Bruce, C., McMahon, C., Buckingham, J.H. & Roggenkamp, M. (2003). Ways of experiencing the act of learning to program. Accessed: March, 2004. http://eprints.qut.edu.au/archive/0000756/.

Bruner, J. (1973). *Going Beyond the Information Given*. New York: Norton.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A. & Miller, P. (1997). Mini-languages: a way to learn programming principles. *Education and Information Technologies*, 2(1): 65-83.

Buck, D. & Stucki, D.J. (2000). Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. In *Proceedings 31st SIGCSE Technical Symposium Computer Science Education,* ACM, 2000, 75-79.

Buck, D. & Stucki, D.J. (2001). JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum. In *Proceedings 32nd SIGCSE Technical Symposium Computer Science Education,* ACM, 2001, 16-20.

Caillot, M. & Dumas-Carre, A. (1989). Teaching decision making to solve textbook problems. In Mandl, H., De Corte, E., Bennett, N. and Friedrich, H. F. (Eds), *Learning and Instruction.* Oxford: Perganom Press.

Cantwell-Wilson, B. & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. In *Proceedings of the thirty-second*

SIGCSE technical symposium on Computer Science Education, 184-188.

Capra, F. (1997). *The Web of life*. London: Flamingo.

Carbone, A. & Kaasbøll, J. (1998). A survey of methods used to evaluate computer science teaching. In *Proceedings of the 6th Annual Conference on the Teaching of Computer*. Dublin City University, 41-45.

Carr, S. C. (2002). Assessing Learning Processes. *Intervention in School & Clinic*, 37(3):156, 7p, 2 charts.

Casey, P.J. (1997). Computer programming: a medium for teaching problem solving. *Computers in Schools*, 13 (1/2): 41-51.

Chang, C., Denning, P.J., *et al*. Final Report of the Joint ACM/IEEE-CS Task Force on Computing Curricula 2001 for Computer Science. Accessed: March, 2005. http://www.computer.org/education/cc2001/final/index.htm.

Choi, W. S. & Repman, J. (1993). Effects of Pascal and Fortran programming on the problem-solving abilities of college students. *Journal of research on computing in Education*, 25(3), 290, 13p, 4 charts, 1 graph.

Cooper, S., Dann, W. & Pausch, R. (2003). Teaching objects-first in Introductory Computer Science. *SIGCSE'03*. In ACM computing curricula.

Daniels, M.P. & Berglund, A. (1998). Building a rigorous research agenda into changes in teaching. Accessed: April, 2004. http://www.docs.uu.se/docs/cse/papers/brisbane98.html.

Dahlgren, L.O. (1984). Outcomes of Learning. In F. Marton, D. Hounsell and N. Entwistle (Eds**)**, *The experience of learning*. Edinburgh: Scotish Academic Press.

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

Deek F. P. (1999). The Software Process: A parallel approach through Problem Solving and program development. *Computer Science Education*, 9(1): 43-70.

Deep and Surface Learning- Learning and Teaching Theory- Engineering Subject Centre (n.d.). Accessed: August, 2005. http://www.engsc.ac.uk/er/theory/learning.asp.

Department of Education (2003). National Curriculum Statement Grades 10 - 12 (General), Information Technology. Accesssed: February, 2004. http://www.education.gov.za/Curriculum/Curriulum.asp.

Détienne, F. (1990). Expert programming knowledge: A schema based approach. In J.M.

Hoc, T.R.G. Green, R. Samurcay, & D.J. Gillmore (Eds), *Psychology of programming* (pp. 205-222). London: Academic Press.

Dijkstra, E.W. (1989). On the cruelty of really teaching computer science. *Communications of the ACM*, 32: 1398-1404.

Dreyfus, H. & Dreyfus, S. (1986). *Mind over machine: The power of human intuition and expertise in the era of the computer*. New York: Free Press.

du Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer* (pp. 283-299). Hillsdale, NJ: Lawrence Erlbaum.

Dufresne, R.J., Gerace, W.J., Thibodeau Hardiman, P. & Mestre, J.P. (1992). Constraining novices to perform expert-like problem solving analysis: Effects on schema acquisition. *Journal of the Learning Sciences*, 2 (3): 307-331.

Eck, D. J. (2004). *Introduction to Programming Using Java [Online]* Version 4.1, June 2004. Accesssed: February, 2005. http://math.hws.edu/javanotes/.

Eckerdal, A. & Berglund, A. (2005). What does it take to Learn 'Programming Thinking'? *ICER'05. ACM Communications.*

ECU (2002). Bachelor of Business: Information Systems. Accessed: November, 2003. http://www.business.ecu.edu.au/courses/undergrad/Bus/Information-systems.htm.

"Edward de Bono – Lateral Thinking & Parallel thinking". Accessed: February, 2005. http://www.edwdebono.com/debono/lateral.htm.

Engeström, Y. (1993). Developmental studies of work as a testbench of activity theory: The case of primary care medical practice. In S. Chaiklin & J. Lave (Eds), *Understanding Practice*: *Perspectives on Activity and Context*, (pp. 64-103). Cambridge: Cambridge University Press.

Entwistle, N. (1988). *Styles of Learning and Teaching: an integrated outline of educational psychology for students, teachers and lecturers*. London: Fulton.

Fincher, S. (1999). What are We Doing When We Teach Programming? *29th ASEE/IEEE Frontiers in Education Conference.* November 10-13, 1999 San Juan, Puerto Rico.

Gary, L., Gutschow, A., McCartney, R., Sanders, S. & Shinners-Kennedy, D. (2005). What Novice Programmers Don't Know. *ICER'05*, Seattle, Washington, USA. ACM.

Gibbs, G., Morgan, A. & Taylor, E. (1984). The world of the learner. In F. Marton, D. Hounsell and N. Entwistle (Eds), *The Experience of Learning*. Edinburgh: Scottish Academic Press.

Glaser, B.G. & Strauss, A.L. (1967). *The discovery of grounded theory*. Chicago: Aldine.

Gordon, S. (1995c). A theoretical approach to understanding learners of statistics. *Journal of Statistics Education* (Online), 3(3). Available e-mail by sending the one line message: send jse/v3n3/gordon to archive@jse.stat.ncsu.edu.

Gordon, S. (1998). *Understanding students learning statistics: an activity theory approach*. Unpublished PhD Thesis: University of Sydney.

Good, R. & Smith, M. (1987). How do we make students better problem solvers? *Science Teacher*, 54(4):31-36.

Govender, I. & Grayson, D. (2006). Learning to Program and learning to teach programming: Problem solving issues. Paper presented at *ED-MEDIA* Conference 2006, June 26-30.

Green, T.R.G. (1990). Programming languages as information structures. In J.M. Hoc, T.R.G. Green, R. Samurcay, & D.J. Gillmore (Eds), *Psychology of programming*, (pp. 117-137). London: Academic Press.

Guba, E. & Lincoln, Y. (1981). *Effective Evaluation*. San Francisco: Jossey- Bass.

Guindon, R. (1990). Knowledge exploited by experts during software systems design. *International Journal of Man-Machine Studies*, 33: 182-279.

Gurtwistch, A. (1964). *The field of consciousness*. Pittsburgh: Duquesne University Press.

Haberman, B., Lev, E. & Langley, D. (2003). Action Research as a tool for Promoting Teacher Awareness of Students' Conceptual Understanding. *ITiCSE'03*, June 30 – July2, 2003, Thessaloniki, Greece

Hammersley, M. & Atkinson, P. (1983). Ethnography principles in practice. London: Tavistock.

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

Hermans, H. & Kempen, H. (1995). Body, mind and culture: The dialogical nature of mediated action. *Culture & Psychology*, 1(1): 103-114.

Hewson, P.W. & Thorley, N.R. (1989). The conditions of conceptual change in the classroom. *International Journal of Science Education*, 11, Special issue: 541-553.

Holland, S., Griffiths, R. & Woodman, M. (1997). Avoiding Object Misconceptions. In *Proceedings of the 28th SIGCSE technical symposium on Computer Science Education.*

Hounsell, D. (1984). Understanding teaching and teaching for understanding. In F.Marton, D. Hounsell and N. Entwistle (Eds), *The Experience of Learning.* Edinburgh: Scottish Academic Press.

Huit, W. & Hummel, J. (2003). Piaget's theory of Cognitive Development. *Educational Psychology Interactive*. Valdosta, GA: Valdosta State University. Accessed: February, 2006. http://chiron.valdosta.edu/whuitt/col/cogsys/piaget.html.

Jenkins, T. (2001). The Motivation of Students of Programming. *Proceedings of ITiCSE 2001*, pp 53-56.

Jenkins, T. (2002). On the Difficulty of Learning to Program. Accessed: January, 2004. http://www.ics.ltsn.ac.uk//pub/conf2002/jenkins.html

Johansson, B., Marton, F. & Svensson, L. (1985). An approach to describing learning as change between qualitatively different conceptions. In A. C. Pines and L.H.T. West (Eds), *Cognitive structure and conceptual change*. New York: Academic Press.

Kaasbøll. J.J. (1998). Exploring didactic models for programming. *Norwegian Informatics Conference*, Tapir, Trondheim, 195-203.

Kahney, H. (1989). What do novice programmers know about recursion? In E. Soloway & J.C. Spohrer (Eds), *Studying the novice programmer* (pp. 209-228). Hillsdale, NJ: Lawrence Erlbaum.

Kessler, C.M. & Anderson, J.R. (1989). Learning flow of control: Recursive and iterative procedures. In E. Soloway & J.C. Spohrer (Eds), *Studying the novice programmer* (pp. 229-260). Hillsdale, NJ: Lawrence Erlbaum.

Kirk, J. & Miller, M. L. (1986). *Reliability and Validity in Qualitative Research*. Beverley Hills: Sage Publications.

Knuth, D.E. (1997). *The Art of Computer Programming. Volume 1/ Fundamental Algorithms*. 3rd edition Addison-Wesley.

Kölling, M. (1999). The Problem of teaching Object-Oriented Programming, Part 1: Languages. *Journal of Object-Oriented Programming*, 11(8):8-15.

Kölling, M. & Rosenberg, J. (2001). Guidelines for Teaching Object Orientation with Java. *Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE)*, Canterbury, 2001.

Lancy, D. (1993). *Qualitative Research in Education*. New York: Longman.

Laurillard, D. (1984). Learning from problem solving. In F. Marton, D. Hounsell and N. Entwistle (Eds), *The experience of learning*. Edinburgh: Scotish Academic Press.

Lave, J. (1988). *Cognition in Practice*. Cambridge: Cambridge University Press.

Leont'ev, A. N. (1978). *Activity, Consciousness, and Personality*. (M. J. Hall, Trans.). Englewood Cliffs, New Jersey: Prentice–Hall.

Leont'ev, A. N. (1981). The problem of activity in psychology. In J. V. Wertsch (Ed.), *The Concept of Activity in Soviet Psychology*, (pp. 37-71). New York: M. E. Sharpe.

Lewis, J. (2000). Myths about Object-Orientation and its Pedagogy. *SIGCSE 2000* 3/00 Austin TX, USA.

Liao, Y.K.C. & Bright, G.W. (1991). Effects of computer programming on cognitive outcomes: A Meta analysis. *Journal of Educational Computing Research*, 7: 251-268.

Lincoln, Y. S. & Guba, E.G. (1985). *Naturalistic Inquiry*. USA: Sage Publications, Inc.

Linder, C.J. & Marshall, D. (2001). The role of context in the characterizations of mindful conceptual dispersion and reflective learning. *Paper presented at the 9th conference of the European Association for Research on Learning and Instruction*, University of Fribourg, August 28- September 1, 2001.

Linn, M.C. & Clancy, M.J. (1992). Can experts' explanations help students develop program design skills? *International Journal of Man-Machine Studies*, 36: 511-551.

Linn, M.C. (1985). *The Cognitive Consequences of Programming Instruction in Classrooms*. University of California, Berkeley. Educational Researcher.

Linn, M.C. & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J.C. Spohrer (Eds), *Studying the novice programming* (pp.57-81). Hillsdale, NJ: Lawrence Erlbaum.

Lister, R. & Leaney, J. (2003). Introductory Programming, Criterion-Referencing, and Bloom. *Proceedings ACM SIGCSE'03*, 34th *Technical Symposium on Computer Science Education*, Reno, Nevada, USA.

Mahmoud, Q.H., Dobosiewicz, W. & Swayne, D. (2004). Redesigning Introductory Computer Programming with HTML, JavaScript and Java. In *Proceedings of the 35th ACM Technical Symposium on Computer Science Education* (*SIGCSE March 3-7, 2004*, pp.120-124), Norfolk, Virgina, USA.

Marton, F. & Booth, S. (1997). *Learning and Awareness.* Mahwah, New Jersey: Lawrence Erlbaum.

Marton, F. & Pang, M.F. (1999). Two faces of Variation. *Paper presented at 8th European*

Conference for Learning and Instruction. August 24-28. Göteborg University, Göteborg. Sewden.

Marton, F. (1981). Phenomenography: Describing conceptions of the world around us. *Instructional Science,* 10(2):177-200.

Marton, F. (1986). Phenomenography: A research approach to investigating different understandings of reality. *Journal of thought*, 21(3): 28-49.

Marton, F. (1988). Describing and Improving Learning. In R.R.Schmeck (Ed.), *Learning Strategies and Learning Styles,* (pp. 53-82). New York: Plenum Press.

Marton, F. (1992b). Notes on ontology, Manuscript published as Searching for pedagogy of awareness. *Forskning om utbildning*, 19(4): 28-40.

Marton, F. & Säljö, R. (1976a). On qualitative difference in learning I – Outcome and process. *British Journal of Educational Psychology*, 46: 4-11.

Marton, F., Carlsson, M.A. & Halász, L. (1992). Differences in understanding and the use of reflective variation in reading. *British Journal of Educational Psychology*, 62: 1-16.

Mayer, R.E., Dyck, J.L. & Vilberg, W. (1989). Learning to program and learning to think: what's the connection? In E. Soloway & Spohrer (Eds), *Studying the novice programmer* (pp.113-124). Hillsdale, NJ: Lawrence Erlbaum.

McCoy, L.P. (1990). Literature related to learning problem solving in mathematics and computer programming. *School Science and Mathematics*, 90 (1): 48-60.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.D., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first year CS students: Report by the *ITiCSE 2001* Working group on Assessment of Programming Skills of First-year CS students.

Merriam, S. (1988). *Case Study Research in Education — A Qualitative Approach*. San Francisco: Jossey-Bass Inc., Publishers.

Merriam, S. (1996). Case study research — reliability, validity, generalisability. *Paper presented at the Research Centre for Vocational Education and Training*, University of Technology, Sydney (UTS), Sydney, Australia.

Miles, M. B. & Huberman, A. M. (1984). *Qualitative Data Analysis*. Beverley Hills, CA: Sage Publications.

Miles, M. B. & Huberman, A. M. (1994). *Qualitative Data Analysis*. (Second Edition) Thousand Oaks, CA: Sage Publications.

Miller, G.A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63: 81-97.

Mitchell, W. (2001). A paradigm shift to OOP has occurred…implementation to follow. Accessed: May, 2004. http://www.ualr.edu/wmmitchell/rules.htm.

Morgan, Kenneth J. (1991). *Introduction to Structured Programming Using Turbo Pascal version 5.0.* New York: *M*acmillan Publishing Company.

Nisbet, J. (1980). Educational research: The state of the art. In W. B. Dockrell & D. Hamilton (Eds), *Rethinking Educational Research* (pp. 1-10). London: Hodder and Stoughton.

Palumbo, D.B. (1990a). Programming Language/problem-solving research: A review of relevant issues. *Review of Educational Research*, 60(1): 65-89.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.

Pea, R.D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1): 25-36.

Perkins, D.N., Schwartz, S. & Simmons, R. (1988). Instructional Strategies for the Problems of Novice Programmers. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming* (pp. 153-178). Lawrence Erlbaum.

Petre, M., *et al*. (2004). A large-scale elicitation of students' knowledge of programming constructs. Submitted to ITiCSE for publication.

Piaget, J. (1972). *The psychology of the child*. New York: Basic Books.

Polya, G. (1957). *How to solve it: A new aspect of mathematical method.* Princeton, NJ: Princeton University Press.

Pong, W.Y. (1999). The Dynamics of Awareness. *Paper presented at 8th European Conference for Learning and Instruction*. August 24-28. Göteborg University, Göteborg, Sweden.

Porter, R. & Calder, P. (2004). Patterns in Learning to Program – An Experiment? in R. Lister & A. Young, (Eds), Australasian Computing Education Conference (ACE2004)', Vol. 30 of C*onferences in Research and Practice in Information Technology*, ACS, Dunedin, New Zealand.

Prosser, M. & Millar, R. (1989). The "how" and "what" of learning physics. *European Journal of Psychology of Education*, IV(4): 513-528.

Prosser, M. & Webb, C. (1994). Relating the process of undergraduate essay writing to the

finished product. *Studies in Higher Education*, 19 (2): 125- 138.

Ramsden, P. (Ed.), (1988). *Improving Learning: new perspectives*. London: Kogan Page.

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

Ramsden, P. (1984). The context of learning. In F. Marton, D Hounsell and N. Entwistle (Eds), *The experience of learning*. Edinburgh: Scotish Academic Press.

Redish, E. F. & Steinberg, R.N. (1999). Teaching Physics: Figuring out what works. *Physics Today*, 52(1):24-30.

Reed, W. & Palumbo, D. (1991). The effect of Basic programming language Instruction on High school students' problem solving ability and computer anxiety. *Journal of Research on computing in Education*, 23(3): 343-372.

Rist, R.S. (1995). Program structure and design. *Cognitive Science*, 19: 507-562.

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

Rogalski, J. & Samurcay, R. (1990). Acquisition of programming knowledge and skills. In J.M. Hoc, T.R.G. Green, R. Samurcay, & D.j. Gillmore (Eds), *Psychology of programming* (pp. 157-174). London: Academic Press.

Rowntree**,** D. (2nd ed.), (1987). *Assessing Students: How shall we know them*. New York: Nichols Pub. Co London: Kogan Page.

Rucinski, T.T. (1991). Effects of computer programming on problem solving strategies. *International Journal of Instructional Media*, 18(4): 341, 11p, 4 charts, 2 graphs.

Saj-Nicole, A. J. & Soloway, E. (1986). But my program runs! Discourse rules for novice programmers. *Journal of Educational Computing Research*, 21: 95-125.

Säljö, R. (1988). Learning in Educational Settings: Methods of Inquiry. In P. Ramsden (Ed.), *Improving learning: new perspectives*. London: Kogan Page.

Samurcay, R. (1989). The concept of variable in programming: Its meaning and use in Problem solving by novice programmers. In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer* (pp. 161-178). Hillsdale, NJ: Lawrence Erlbaum.

Savitch,W. (2nd ed.), (2001). *JAVA An introduction to Computer Science & Programming.* Upper Saddle River, NJ: Prentice Hall.

Schoenfeld, A. H. (1985). *Mathematical Problem Solving*. School of Education, Department of Mathematics, University of California: Academic Press.Inc.

Schoenfeld, R. J. (1978). Can heuristics be taught? In: J. Lochhead and J. J. Clement, (Eds), *Cognitive process instruction.* Philadelphia: Franklin Institute Press.

Semenov, N. (1978). An empirical psychological study of thought processes in creative problem solving from the perspective of the theory of activity. *Soviet Psychology*, 16(1): 3-46.

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

Shneiderman, B. & Mayer, R. (1979). Syntactic/semantic interactions in programmer behaviour: a model and some results. *International Journal of Computer and Information Sciences*, 8(3): 219-238.

Shulman, L.S. (1986). Those Who Understand: Knowledge Growth in Teaching. Educational researcher, Stanford University, 15(2): 4-14.

Sieber, S. (1976). A synopsis and critique of guidelines for qualitative analysis contained in selected textbooks (Project on Social Architecture in Education). New York: Centre for Policy Research.

Slay, J. (2000). Implementing modern approaches to teaching computer science: a life-long learning perspective. *16th World Computer Congress*, Beijing, August 21-25.

Slotnick, H., Pelton, M., Fuller, M. L. & Tabor, L. (1993). *Adult Learners on Campus*. Washington D.C: The Falmer Press.

Soloway, E. & Spohrer, J.C. (Eds). (1989). *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum.

Spohrer, J.C., Soloway, E. & Pope, E. (1989). A goal/plan analysis of buggy Pascal programs. In E. Soloway & J.C. Spohrer (Eds), *Studying the novice programmer* (pp. 355-399). Hillsdale, NJ: Lawrence Erlbaum.

Spohrer, J.C. & Soloway, E. (1986). Novice Mistakes: Are the Folk Wisdoms Correct? *Communications of the ACM*. 29(7): 624-632.

Stodolsky, S.S. (1988). *The subject matters: classroom activity in math and social studies*. Chicago: University of Chicago Press.

Stoodley, I., Christine, R. & Bruce, C. (2004). Masters Students' Experiences of Learning to Program: An Empirical Model. In *Proceedings of QualIT2004: International conference on Qualitative Research in IT & IT in Qualitative research.*

Svensson, L. (1984). Skill in learning. In F. Marton, D. Hounsell, & N.J. Entwistle (Eds), *The experience of learning.* Edinburgh: Scotish Academic Press.

Thomas, T., Ratcliffe, M. & Thomasson, B. (2004). Scaffolding with Object Diagrams in First Year Programming Classes: Some unexpected Results. *ACM SIGCSE'04*, March 3-7.

Thomas, R.A. & Sylvester, U. C. (1996). Give programming instruction a chance. *Journal of Research on Computing in Education*, Fall 96, 29(1): 96, 13p, 5bw.

Thompson, E. (2003). Thinking in programming. In: Kaschek, R., Kinshuk, Schewe, K.-D. and Turull Torres, J.M., (Eds.) *PhD Workshop Communications*, Palmerston North: Department of Information Systems, Massy University

Tukiainen, M. & Mönkkönen, E. (2002). Programming aptitude testing as a prediction of learning to program. In J. Kuljis, L. Baldwin & R. Scoble (Eds), 14th Workshop of the Psychology of Programming Interest Group, Brunel University.

Van Maanen, J. (1983). Reclaiming qualitative methods for organizational research: A preface. In J. Van Maanen (Ed.), *Qualitative Methodology* (pp. 9-18). California: Sage Publications.

Varela, F., Thompson, E. & Rosch, E. (1991). *The Embodied Mind*. Cambridge, Massachusetts: the MIT Press.

Ventura, P.R. & Ramamurthy, B. (2004). Wanted: CS1 Students. No experience Required. *Proceedings of the 35th SIGCSE technical symposium on computer science education*, Reno, Nevada.

von Mayrhauser, A. & Vans, A. M. (1994). Program understanding – A survey (*Tech. Rep. CS- 94-120*). Department of Computer Science, Colorado State University.

Vygotsky, L. S. (1962). *Thought and Language*. Cambridge, Massachusetts: The M.I.T. Press.

Vygotsky, L. S. (1978). *Mind in Society*. Cambridge, MA: Harvard University Press.

Wells, G. W. (1981). The relationship between the processes involved in problem solving and the processes involved in computer programming. (Doctoral dissertation, University of Cincinnati). Dissertation Abstracts International, 42, 2009A-2010A.

Wertsch, J. V. (1981). The concept of activity in Soviet psychology: An introduction. In J. V. Wertsch (Ed.), *The Concept of Activity in Soviet Psychology* (pp. 3-35). New York: M. E. Sharpe, Inc.

Wertsch, J. V. (1985). *Vygotsky and the Social Formation of Mind*. Cambridge, Massachusetts: Harvard University Press.

Widowski, D. & Eyferth, K. (1986). Comprehending and recalling computer programs of

different structural and semantic complexity by experts and novices. In H.P. Willumeit (Ed.), *Human decision making and manual control* (pp. 267-275). Amsterdam: North-Holland Elsevier.

Wiedenbeck, S. & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International journal Human-Computer Studies* 51: 71-87.

Winrow, B. (1999). The Walden programmer analyst aptitude test. Dr. Dobb's Journal, Fall 1999, Software Careers. Accessed: February, 2005. http://www.ddj.com/documents/s=894/ddj9914b/9914b.htm).

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

Zhu, H. & Zhou, M. (2003). Methodology First and Language Second: a way to Teach Object-Oriented Programming. *OOPSLA '03*, October 26-30, 2003, Anaheim, California, USA.

APPENDICES

Appendix A1
Back ground questionnaire

(confidential**)**

Please complete this form in full marking the relevant choice with a tick.

Date: _____ name:_____

---

1. Your gender:
   Male ❍ Female ❍
2. How old will you be on 31.12.2004?
   18 ❍ 19 ❍ 20 ❍
   21 ❍ 22-25 ❍ 26 -30 ❍
   30-39❍ 40+ ❍
3. Your racial background?
   African ❍ Asian ❍
   Coloured ❍ White ❍
4. Is English your mother tongue?
   Yes ❍ No ❍

If not, state your mother tongue: _____

5. In what year did you complete high school?
   2004 ❍ 2003 ❍ 2002 ❍
   2001 ❍ 2000 ❍ Prior to 2000 ❍
6. Have you ever been employed before?
   Yes ❍ No ❍
7. Should your answer to question 6 be yes, state your profession/occupation:

   _____

8. Highest academic qualification

Matric	Diploma	Other

   If other state qualification and institution obtained from.

   _____

9. Do you have a PC at your disposal to work on away from campus?

   Yes ❍ No ❍

10. Did you learn computer programming before? _____.
    If yes, what programming language did you use _____.

11. What was your average grade during the final matric examination?

A ❍ B ❍ C ❍
D ❍ E ❍ F ❍

12. Do you have any other computing knowledge prior to registering at this university?

Yes ❍ No ❍

13. How would you rate yourself as a computer user?

Advanced ❍ Competent ❍ Struggling ❍

14. Did you obtain a matric exemption?

Yes ❍ No ❍

15. From what kind of high school did you graduate?

Private ❍ Former model C ❍ Former Indian ❍
Former African ❍ Former Coloured ❍ Home School ❍

16. Indicate the symbol and grade (HG/SG) you obtained for the following subjects:

English Afrikaans Zulu
Mathematics Physical Science Business Eco
Geography/History Accounting Biology
Computer Studies

17. Which phase of schooling are you studying towards?

Foundation ❍ Intermediate ❍ FET ❍

18. When deciding on a career, was teaching your?

First choice ❍ Second choice ❍ Third choice ❍

19. Any comments you want to make pertaining to any question or in general regarding the computer science education/ information Technology?

_____
_____

_____

20. What are your views/feelings regarding computer programming.

_____

_____

_____

_____

21. What aspect(s) of programming, if any, do you find difficult? Explain fully.

_____

_____

I agree for the above information to be used for academic research purposes.
Yes No

Appendix A2
Analysing the problem worked with

Complete with full explanations.

1. What concepts, formulas, and rules did I apply? _____
_____
_____
_____

2. What methods did I use?_____
_____
_____

3. How did I begin?_____
_____
_____
_____
_____

4. Have I seen this problem before?_____
_____

5. Is it similar or dissimilar to other problems I've done?_____
_____
_____

6. How does my solution compare with the examples from the book and class?_____
_____
_____
_____

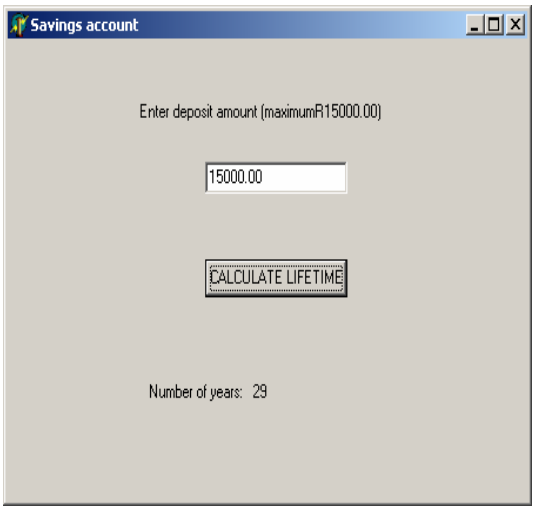
7. Could this problem be worked another way? Can I simplify what I did?_____
_____
_____
_____
_____

8. Next to each problem solving step, explain what you did and why._____
_____
_____
_____
_____
_____

9. What is the most difficult part of solving the problem?_____
_____
_____
_____
_____

Problem 1 (given to Pre-service teachers during the pilot study)

Write a Delphi program that reads an amount deposited into a savings account that pays 5% interest. The deposit amount may not exceed R15000.00. If R1 000.00 is withdrawn every year and nothing more is deposited, the program must calculate how many years it will take for the account to be depleted.



Problem 2 (given to grade 12 students)

Write a program that would generate a 2-D square array (4 x 4) and find the sum of the inner square of the array. The original array, the inner array and the sum of the inner array should be displayed

Appendix A3
Problem solving in computer programming

Name of teacher _____ Name of class_____ Name of subject _____

The word "problem" is used in these statements. It refers here to those computer program problems, and exercises you did this year and last year in class, for homework and in tests (including examples and worked problems done by the teacher).

Please circle the number which best represents your response to the statement
Strongly agree 1, agree 2, undecided 3, disagree 4, strongly disagree 5

	STATEMENT	**RESPONSE** Strongly agree 1 Agree 2 Undecided 3 Disagree 4 Strongly disagree 5
	How do you learn to solve problems?	
1	Reading and studying the worked examples in my notes or text book helps me to successfully solve problems.	1 2 3 4 5
2	I find that discussing how to solve a computer program problem with a friend is valuable.	1 2 3 4 5
3	I find that watching the teacher "go over" or "work out" a problem solution is valuable and I learn a lot.	1 2 3 4 5
4	I learn "off-by-heart" how to solve particular types of problems expected in the tests and examinations.	1 2 3 4 5
5	For me solving problems mostly involves matching previously done problem solutions with the current problem I have to solve.	1 2 3 4 5
6	I find that the most effective way to learn how to solve problems is to practise on many similar problems.	1 2 3 4 5
	What types of problem are you given?	
7	I am given a variety of interesting problems to solve.	1 2 3 4 5
8	Most problems I do have only one correct answer.	1 2 3 4 5
9	Most problems are numerical and require me to use formulae and equations to solve them.	1 2 3 4 5
10	Most problems I do, only have one acceptable method of solving them.	1 2 3 4 5
11	The problems I am given to do are about situations in life e.g. finding interest on account, determining the lowest and highest scores etc.	1 2 3 4 5
12	I find most of the programming problems difficult to solve	1 2 3 4 5

	STATEMENT	RESPONSE Strongly agree 1 Agree 2 Undecided 3 Disagree 4 Strongly disagree 5
	How does problem solving help you learn computer programming?	
13	If I can solve the problems, I think this indicates that I understand the programming.	1 2 3 4 5
14	Solving problems uncovers my misunderstandings of content i.e. theory, rules, concepts, principles etc.	1 2 3 4 5
15	Solving problems helps me to think logically i.e. understanding of theory, concepts, laws and principles etc.	1 2 3 4 5
16	I can only solve the problems successfully if I understand the knowledge involved i.e. theory, principles or concepts.	1 2 3 4 5
17	I have been taught general strategies (plans) for solving new problems e.g. what to do when I get stuck on a difficult problem	1 2 3 4 5
18	I have been taught a number of specific procedures to solve particular types of problems e.g. a method to solve sorting, searching, fibonac problems	1 2 3 4 5
	How relevant and useful are the problems?	
19	Computer programming helps me to recognise how to solve problems in real-life situations.	1 2 3 4 5
20	I find the problems I have to solve in Computer Studies relevant and useful.	1 2 3 4 5
21	Solving computer programs improves my problem solving skills in other subjects I take at school.	1 2 3 4 5
22	The problems I do give me skills which I can use now to solve problems that I encounter in my life outside school.	1 2 3 4 5
23	I think that what I have learnt from doing computer programming will assist me in my future everyday life.	1 2 3 4 5
24	My ability to solve these problems will be valuable for further study e.g. University, and technikon.	1 2 3 4 5

Any comments you would like to make about your experience of solving problems, using the computer.

Appendix B

Background questionnaire for in-service teachers

Centre for the Improvement of Mathematics, Science and Technology Education: Computing for Teachers Ib (Java 2004)

Please complete this form in full when you receive your material. Return the form either by enclosing it with your first assignment that you post to UNISA or by faxing it to us (Fax: (012) 429 8690).

Student and School Information

Student number: ……..Location: (urban/peri-urban/rural): ……..……

Surname: ……..Private or government: ……..……

Gender: ……..……..…Subjects currently teaching: ……..……..…….

Race: ……..……..……..…Province: ……..……..……..…

School: ……..……..

Education received

Type of Education	Courses completed, if any (indicate Secondary or Tertiary level and Institution)	Qualifications (if any)
Computing		
Mathematics		
Education		
Other		

Programming experience *(List the programming languages in which you have some experience, if any):*

Programming Language	Knowledge of the language *(tick one)*		How did you learn the language?	Number of years you have been teaching/using the language
	I have very limited knowledge			
	I know the basics			
	I know the language well			
	I have very limited knowledge			
	I know the basics			
	I know the language well			
	I have very limited knowledge			
	I know the basics			
	I know the language well			

Appendix C
Examination questions

Extract from the examination paper for the in-service course, Computing for teachers 1b.

Question 2 (12marks)

The Zoo charges the following entrance fees:

- Entrance fee per adult = R 8.50
- Entrance fee per child = R 4.50

2.1 Write a program that asks the user to enter the number of adults and the number of children. Your program should then calculate and print the total entrance fee. Marks will be given for showing planning in the form of comments in your program. (8)

2.2 Give test data in the form of normal, extreme and erroneous cases for testing your program above. Give at least one normal case, two extreme cases and one erroneous test case. You are not required to provide expected output values. (4)

Question 5 (16 marks)

Consider the following class Date and class UseDate written by a student:

```
public class Date
{
  private int year;
  private int month;
  private int day;

  public Date (int y, int m, int d)
  {
    year = y;
    month = m;
    day = d;
  }

  public boolean isLeap()
  {
    int year = 2004;
    if (year % 4 == 0)
      return true;
    else
      return false;
  }
}
```

```
import java.awt.*;
import hsa.Console ;

public class UseDate
{
  static Console c;           // The output console
  public static void main (String [] args)
  {
    c = new Console ();
    c.print ("Enter year, month, day (separated by spaces) :");
    int year = c.readInt();
    int month = c.readInt();
    int day = c.readInt();
    Date userDate = new Date (year, month, day);
    if (userDate.isLeap())
      c.println("Is leap year");
     else
      c.println("Is not leap year");
  } // main method
}    // The "UseDate" class.
```

Depending on the year entered by the user, the program should display “Is leap year” or “Is not a leap year”. The problem is that no matter what the user types in, the program always displays “Is leap year”.

5.1 Explain why the program is not working as it should and what can be done to fix the problem. (2)

5.2 Describe in detail how you could use variable box diagrams and arrows (if need be) to explain this problem to a class of learners. Your diagram must be accompanied by a textual description. Explain what happens with and without the change as described in 5.1. (8)

5.3 Write a second constructor for the Date class which takes a single parameter of type String: a date in the the form dd/mm/yyyy (for example, “24/09/1998”). Your constructor should extract the day, month and year parts from this string parameter and use them to initialise the data members of the Date class. You are not required to do any checking for invalid values. (Hint: to convert a String to an integer, use the method Integer.parseInt, see Appendix B). (6)

Extract from the examination paper for the pre-service course.

Question 4

```
public class LineTestExam
{
        private int size = 3;
       private char pattern = ‘*’;

public void setSize (int s)
{
          If (s>=0)
           Size = s;
}
```

```
Public void setPattern (char p)
{
        for (int i = 1; i<= size; i++)
         {
            for (int x =1; x<= i; x++)
                System.out.print(pattern);
          System.out.println( );
           }
 }
}
```

```
import Utilities.*;
public class LineTestE
{
Public static void main (String [] args)
{
    LineTestExam  line1 = new LineTestExam();
    Line1.draw( );

    System.out.println ("Enter the size of line you want");
    Int num = Keyboard.getInt( );
    System.out.println("Enter the pattern of line you want");
    Char pat = Keyboard.getChar( );
    Line1.setSize (num);
    Line1.setPattern(pat);
    Line1.draw( );
    LineTestExam line2 = new LineTestExam ( );
    Line2.setSize(5);
    Line2.draw ( );
    Line1.draw( );
  }
}
```

4.1 Trace through the program above and give the exact output when the main method is executed. Use as input; 4 for variable num and "%" for variable pat. [8]

4.2 If the line **"private int size =3"** was changed to **"private static int size = 3;"** what would the output be, given the same input as 10.1, and why? [4]

4.3 If the line **"public void draw( )"** was changed to **"private void draw( )"**, will the main method compile successfully? Explain your answer. [4]

Question 6

For the following program you must make use of classes, constructors, objects and methods. Write a Java program that will calculate the distance between 2 points on the Cartesian plane and determine the equation of the line y=mx + c that passes through those 2 points.

The points (x1, y1) and (x2, y2) are input via the keyboard.

Distance = $\sqrt{(x2-x1)^2+(y2-y1)^2}$

Gradient m = $\frac{y2-y1}{x2-x1}$

[40]

And the constant c = y1-m*x1

Appendix D

Interview with students about programming

Interviews with pre-service students learning JAVA

R: Is it the first time you are studying programming since you've come to university?
S1: I started programming at university.
R: Were you exposed to computers before?
S1: NO, Yes, using computers.
R: What do you understand by programming?
S1: Well I wouldn't say too much. Actually what programming is about?... I think it's the way things are made simpler and implemented …To find easier ways to do things.
R: Can you write a program that works?
S1: Yes. I have
R: How do you know?
S1: Yes I have written one
R: Can you write a good program?
S1: I wouldn't say my program is a good program...but I've seen it work.
R: Why do you say it is not good?
S1: Well since I 'm not very experienced in the field and I have been reading books… and I've come across books that say… even an experienced programmer does not write a good program sometimes. So I wouldn't feel that my program is perfect.
R: How do you see your current ability to program?
S1: I think it's stable but I need to work hard
R: Can you describe the process you go through when you write a program? What do you do first?
S1: Okay I think of a program generally. Like, how would I solve this problem. Just put it down roughly. Okay… If this happens its gona be because of this, so I got to do this because of this. Have a reason for everything that I do. Break the program down. Make sure it makes sense and then write it.
R: In other words do you write a psuedocode?
S1: Yes, I write an algorithm to solve it and then when I see that it makes sense I try to code it.
R: How have you learnt to program? How did you go about learning to program?
S1: I practiced a lot. And I just thought of a program generally. I try to solve the problem generally. For instance the car problem that we've been doing, just make sense of it generally and come up with a pseudocode ... I mean a program. I think of the problem generally.
R: But before you could come to the point of solving the problem how did you learn this language, was it purely by lectures, or notes or reading?
S1: My learning experience is mainly by lectures. I assisted my self a little bit.
R: How did it take place? How were you taught the subject programming, the language? Did you learn the syntax, or the objects?
S1: I was taught the syntax first that's what I came to grips with first…
R: Have you learnt about objects and classes yet?
S1: I have started but not confident yet about classes and object.
R: What would you say is the demanding or difficult aspect of programming in your experience?
S1: Actually programming itself and coming up with the sequence of steps to solve the problem.
R: Do you enjoy programming?
S1: Yes I do because it demands a lot of thinking and when you've thought of something and solved the problem you feel good about yourself. You feel that you're a great thinker.
R: So that's what makes it enjoyable?
S1: Yes
R: Is there anything else you'd like to say with regard to programming in general or objects?
S1: It is an interesting module and ... that is far as I can say…

R: Is this the first time you started to learn programming since you came to this university?
S2: Yes I only started programming last semester.
R: Have you done anything with regards to computers?
S2: I used the computer at school in literacy.
R: Were you competent in the use of the computer before you could come to university?
S2: Yes
R: In your own words what do you understand by the term programming to be, what comes to your mind?
S2: Basically its problem solving.
R: Can you write a program that works?
S2: Yes structured programming, yes I can.
R: Why do you say structured programming?
S2: Because last semester we learnt structured programming and this semester we learnt how to create objects and I have some difficulty with objects.
R: How do you know that you can write a program that works?
S2: I think if I can execute it and it works
R: Can you write a good program
S2: I'm not too confident, but with time I think I can.
R: What would you say is a good program?
S2: I think a program that does not have errors, a program that can be understood by other programmers.
R: How do you see your current ability to program?
S2: Well at this point I'm not too confident about myself but as time goes I keep on learning
R: Can you describe the process when you write the program
S2: First I look for the keywords in the problem see what's required the calculations etc.
R: What are the keywords you are talking about? Give an example.
S2: Like if they say write a program to calculate the average of a student then the average is one of the keywords, the other values would be the marks of the students so maybe the values as well..
R: How did you learn to program, is it from lectures, from the web, from textbooks?
S2: In my case it's been mostly from lectures. Last semester the more I practiced the more I understood.
R: What is the most demanding or difficult aspect of programming?
S2: Basically it's solving a complicated program, because it makes you think.
R: Do you enjoy programming?
S2: Yes I do
R: What makes it enjoyable?
S2: It's challenging, it tackles your mind, it makes you think. The fact that your program is running, makes you feel good it makes you think that you can do something, therefore I feel challenged.
R: Do you understand objects, classes and how to program using objects and classes?
S2: As I said I have difficulty with objects and classes now. I don't understand them completely.
R: But you are confident that you will get the hang of it.
S2: Yes, before the semester is over.

R: Have you done programming before?
S3: Yes
R: At school?
S3: Yes at grade 10, 11 and 12.
R: What language have you learnt?
S3: Turbo Pascal
R: How would you classify turbo Pascal?
S3: I don't know
R: How is TP similar or different to Java?
S3: We just started with Java. We did one semester of Java They are more or less similar. We are just working on the difference that is objects now. I can't actually say much.
R: How do you see programming?
S3: It's a solution to a problem.

R: Can you write a program that works?
S3: Yes
R: How do you know?
S3: Well I've done it before.
R: That's good. When you say that, are you referring to Pascal, Java or both?
S3: Both
R: Can you write a good program?
S3: I can write a good program in Pascal but Java… I'm not that good. Basic programs
R: What would you say is a good program?
S3: It must contain a bit of graphics, sound..that's in Pascal..solves the problem without any errors.
R: How do you see your current ability to program?
S3: Fairly good
R: Can you describe the process you go through when you write a program?
S3: First you got to plan your algorithm or flowchart. Then code
R: Remember I'm not talking about theoretically what must be done, I referring how do you proceed?
S3: That's how I do it.
R: How did you learn to program? Was it purely by the teacher teaching you, through lectures, or the web, textbooks, supplementary reading?
S3: Trial and error. The teacher teaches certain topic, and I go home and work on it
Make mistakes, and get the solutions, see why certain things happen
R: But it's first initiated by the teacher or lecture before you experiment with it
S3: Yes. In Pascal I use to do my learning, but now I wait for the lecture, because it's a bit different.
R: What is the most demanding or difficult aspect of programming?
S3: You have to first find out exactly what they want from you in the question and that sometime is a bit difficult.
R: So it's interpreting the question?
S3: Yes. To find out what variable you need to declare.
R: In object –oriented programming are you able to identify what the objects will be immediately, the classes?
S3: Now that's more of a problem now.
R: Now you've learnt a structured programming language, such as Pascal and Java is an object-oriented language, now that you are learning Java, do you find that you get confused with the two languages? Or is it a problem to shift over?
S3: I work with them separately. But I try to still remember what I did Pascal. I don't want to forget. But when I'm working in this code I only think of what will happen in this language. I don't put them together… It doesn't work.
R: Okay. That's as far as syntax is concerned, but the actual problem solving is a little bit different?
S3: I use the knowledge of the previous language.
R: Do you enjoy programming?
S3: yes
R: What makes it enjoyable?
S3: It's nice to have a problem and work it out that do something
R: What is difference between Pascal and Java, A quick review, what strikes you as different?
S3: The names sometimes are different, like Procedures and functions are called methods in Java. The way you start a program. In Pascal you give the programs name, but here you start with public classes and methods.

R: What do you understand by the term programming?
S4: Programming to me is just solving problems…and using different programs, programming languages
R: You see programming as problem solving?
S4: Yes, for me its just problem solving.
R: Can you write a program that works?
S4: Yes
R: How do you know?
S4: Because I tried it. I run the program and it works.

R: Do you get the results you wanted or does it work the first time round?
S4: No, I got to keep trying until I get the result.
R: Can you write a good program?
S4: Yes
R: How do you know? What makes you think that it is a good program as opposed to lets say a not so good program?
S4: I think when I run it I get ...compared to...like in my class...compared to the other students I get the results that I want, the output that I want.
R: When we think of good programs, do you think of the type of structures that are used or the efficiency of memory allocation that is used? Does that come to mind or just that because you get the results you feel it is a good program?
S4: No I think a good programmer solves the problem properly. I don't think a good programmer is someone wasting his time writing comments and all in the program…As long as they can solve the problem.
R: Okay. How do you see your current ability to program?
S4: Good
R: If you had to rate it on a scale from one to 5, what would you say were?
S4: I would say …er 3.5 maybe.
R: Can you describe the process you follow when you are given a problem to write a program? Think back of the steps you follow.
S4: Like what I start with?
R: Yes.
S4: I always write an algorithm first, then I write the program
R: How did you learn to program?
S4: In school from grade 10 to 12 and then now at university
R: Did you have to read up textbooks or did you get material from the teacher or lecturer?
S4: I think also of the material I got from high school. When I was at school I went for tuition.
R: What language did you study at school?
S4: Turbo Pascal
R: What would you consider is the most demanding or difficult aspect of programming?
S4: I think the syntax. Like now we are doing Java I can't remember some of the syntax ...not everything is on the data sheet provided and I can't remember that. But solving problems, I love solving problems, only if I can't solve it, then maybe... But I don't think it is difficult just… I just find that writing out the program using the proper language I think that's just ...(pause) irritating.
R: That's irritating.
S4: Yes
R: But would you say that that's the most demanding and difficult aspect, the syntax?
S4: Ja maybe
R: Do you enjoy programming?
S4: Yes
R: What makes it enjoyable?
S4: Solving the problems and getting it right. My favourite… I like working out sequences. Like when we get a number like 4, 16 , you work out the 12 in the sequence of numbers
R: You mean the pattern in a series of numbers?
S4: Yes.

R: Can you tell me what do you understand by programming?
S5: Programming is… a for example you got a question and you need to have a program in which you can get answer for the question…ja
R: So that is how you perceive programming, an answer to a problem
S5: Ja in which somebody can use it
R: Can you write a program that works?
S5: Yes I can.
R: How do you know that?

S5: When I write a program it does exactly what I want it to do?
R: Can you write good program?
S5: Ja, I think I can.
R: How would you know it's a good program?
S5: Because the style in which I write it and the languages and many other things
R: How do you see your current ability to program?
S5: Its good but as far I know for now ...but as time goes on I think it will improve. But its good for the time being with the information I know.
R: Did you do computers/programming before?
S5: I didn't do programming before we started last semester.
R: Have you been exposed to computers before?
S5: Yes but not in programming
R: How have you learned to program? Through lectures, web, textbooks?
S5: Only through lectures.
R: Have you not supplemented this learning with any other material?
S5: No
R: Just by attending lectures and listening you were able to come to this point of programming?
S5: Yes
R: What is the most demanding or difficult aspect of programming?
S5: Right now its methods and passing of parameters, the only thing I'm trying to get now…
R: Objects and classes do you understand the terminology?
S5: Yes
R: Can you describe the process you follow when you are given a problem to write a program? Think back of the steps you follow.
S5: I sit and look at the question and find out what I really need to do and then I find out the steps in which I need to get to the answer. And then I write an algorithm most of the time, not always and then I start programming.
R: You find that it works
S5: Yes
R: Do you enjoy programming?
S5: I do at times if I get to the answer then I'm happy but if I don't You get so confused if you don't get to the answer, because you don't know really where the problem is. You find that always you think your answer is right and if it doesn't run you get so confused.
R: Does it not motivate you to get down and find out what went wrong?
S5: Yes it does. Because for instance when we were doing the corrections its like oh I went wrong there and you can correct it for the next time.
R: What makes it enjoyable? You say you do enjoy it.
S5: It's challenging and it gets you to think...I don't know how to put it...to think .at that time this thing you want to do now.

R: What do you understand by the term programming? What do you perceive programming to be?
S6: The first thing that comes to my mind… is the idea of a problem and programming is the way in which we go about solving a particular problem…We use lots and lots of tools.. use various tools to do that.
R: Can you write a program that works?
S6: Yes I can.
R: How do you know?
S6: Very often we were given exercises to do on a regular basis. Lot of the work we do is on our own and er so I do know that I can write because the exercises we do are similar.
R: Can you write a good program?
S6: I'd like to think so.
R: How do you know it's a good program?
S6: I suppose just looking back to where I've come from with other programs to now doing program which will work out various times that different vehicles will take to reach a particular destination. I suppose just looking at that it's a big step.

R: How do you see your current ability to program?
S6: I'm quite optimistic I suppose, especially considering the fact that we moving into a different style of programming. So whereas there are quite a few similarities there are few concepts I'm just starting to come to terms with.
R: What are those concepts?
S6: Quite a few concepts with regard to objects…object- oriented programming and just getting my head around looking at a particular (end of tape)
R: You were trying to orient your thinking to objects and classes.
S6: Yes and just the whole idea that each object has the same characteristics of that particular class and the whole idea that you can have many, many objects and they basically all draw on the same memory location as the classes.
R: How have you learned to program? Through lectures, internet, textbooks?
S6: It was exclusively from lectures. I came into this course without very much...knowledge, without any knowledge as far as programming was concerned. So everything was new to me.
R: So you had not done this subject at school?
S6: No.
R: But were exposed to computers but not programming?
S6: Yes, in fact all the work I did on computers before that was on my own.
R: Can you describe the process you follow when you are given a problem to write a program? Think back of the steps you follow.
S6: Thankfully its been indoctrinated into my mind that the first thing I need to do is to understand the problem itself and …follow the formal steps which is to write an algorithm ….do some tests to see if it works on paper and the next step would be to ..once you figure it out how you can solve the problem then you could sort of outline your program and go from there. Do simple tests, run simple tests data..
R: What is most demanding or difficult part of programming?
S6: For me It's most definitely the language and the syntax. I won't say I'm struggling but I find it hard to come to terms with some of..well that particular aspect.
R: So actually solving the problem is not the problem?
S6: No In fact I like to think that I have a very.. logical mind.
R: Do you do Mathematics?
S6: No..
R: Do you enjoy programming?
S6: I do..I do, the most enjoyable part is to see it the program run
R: So that's what makes it enjoyable?
S6: It's when you click on the compiler, you hold your breadth for a couple of seconds
R: What made you choose this course? What motivated you?
S6: From the onset and I did speak to Mr…, I did not have the grades to do this course and no mathematics. I had to write a motivational letter. All because I wanted to do computer science education because the way things are going technologically speaking. Its going to be a must for kids to come. Somewhere along the line its changed now. We've become a little bit of a family now, those guys that made it from first to second year. Although it is a primary motivation, I keep going each day and each lesson and also I don't want to let the guys down or Mr … down, myself down.

R: Why did you choose to do this course computer science education?
S7: Honestly I had,..I was short of credits. And I needed to make up credits it was the most appealing subject left for me to do.
R: Presently, can you write a program that works?
S7: Yes
R: How do you know that?
S7: Because I can, it runs and I can credit my self given a problem, you can code it, execute it and it runs.
R: Can you write a good program?
S7: I don't know.. (laughs).. I hope so.
R: What would you consider to be a good program?

S7: One that is efficient and does not have too much information that is not needed, a program that is orderly and lacks and calls for things at the right places...
R: Okay. How do you see your current ability to program?
S7: I.. as I was saying to Mr… I was so fine with last years work and I was getting to know it properly. But now we started this object-oriented, I'm a bit lost with that. I don't...But I was like that at the beginning with the other one, so at the moment I'm a bit unstable…like ooh! Like I don't know what I'm doing, but I'll get there.
R: Do you enjoy programming?
S7: Yes, very much. I could have dropped it this year I have too many credits as it is but I decided to keep it.
R: Do you see it as a challenge?
S7: Yes, I enjoy it.
R: How did you go about learning to program?
S7: First we started with the theory of programming and then we started on flowchart and flow diagrams and how to pass information and we just given programs to type and we got used to the syntax. Then we started problem solving. And then coding
R: What do you use as your main resource? Is it just attendance at lectures, additional material from the web, from textbooks or just talking with others?
S7: I think I learnt most obviously in my lectures. Mr… does a lot with us in the lectures. If we miss any lecture we do miss a lot. My textbook is very helpful; it's got a lot of more theory than actual programs. But I think it has a lot to do with doing it by yourself and working through the problem and you know trying to do it properly.
R: Have you got Computer science or programming background?
S7: No, nothing.
R: So you are just computer literate?
S7: Yes, I did the ICDL at UNISA and that was as far as it went.
R: When we talk of programming, what comes to your mind as you are learning? How do see programming or learning to program?
S7: I feel that It's finding a niche, like a market in a programming atmosphere where somebody hasn't already developed a program to do a certain thing and that you will fill that...that gap and maybe design a program to do something that somebody needs; or even upgrading programs to make them… you know further with technology.
R: Do you think it has anything to do with problem solving?
S7: Oh sure, definitely, because you have to first, have a problem to solve it and then program it.
R: Is there anything you would like to tell me with regard to programming, Java how you see it
S7: Not really, I feel maybe the other students may have more of a programming back.. not background, but they have had more programming languages, no like various different languages maybe, because they've done Java, now they are doing Java this year and they did Delphi and I know that Mr … is trying to teach us principles that will fit over each programming language but I don't know what I will do given a program in Pascal. I feel unsure about that.

Interview of students after writing a test in programming

R: So Thami you've just written this test. How do you feel?
S1: Very scared
R: Very scared…(confirming)
S1: Very scared
R: Those are your feelings about the test?
S1: Yes
R: Why are you feeling scared?
S1: Okay…It was something that not what I expected...it was totally different from what I expected.
R: How was it different from what you expected?
S1: The way I studied wasn't what the way it was in the test.
R: Okay…You said that the problem in the test was different from what you expected?

S1: yes
R: So what were you expecting for the test?
S1: The actual problem was very different from what we usually do.
S1: I expected some problems like the ones we did in class.
R: What kind of problems did you do in class?
S1: It's actually the question… the way it was…
R: worded (completing the sentence)
S1: Yeah… it was worded
R: Did you understand the problem and what was required?
S1: Yes I did understand the problem. I had some problems interpreting it
R: Now that you say you had a few problems interpreting it.
Which part or aspect do you think you misinterpreted or had a problem with? Was it making use of classes, objects, designing the methods?
S1: Creating the objects… creating the objects actually.
R: That's where you had a problem.
S1: Yes.
R: Do you understand what an object is in programming?
S1: I do have an understanding of what it is but not the entire thing
R: Were you able to write the method in order to calculate the interest?
S1: Yes I did write the method
R: How did you go about solving the problem? Can you trace back your steps as soon as you got the test?
S1: Okay I thought about the methods that needed to be in the class and the input values that should be in the constructor and … I just laid it out I just made an outline of the...(inaudible)
R: So did your program work, run?
S1: The first one had two variables in the class and the main class…mm
R: You could not follow the logic?
R: What was the most difficult or challenging part of the problem?
S1: Creating the objects and calling the methods.
R: That seems to be the problem?
S1: Yes…
R: Is there anything else you'd like to say about programming or solving the problems?
S1: Well …you never say you know…you know …
R: Until you work the problem (completing the sentence)
S1: It's not something that you can work the problem and say you know. You really need to… know programming.
R: So you need to develop your problem solving skills on a regular basis.
S1: Yes…

R: What are your feelings about the test?
S2: I have mixed feelings about it
R: mixed feelings? Why?
S2: I think I was able to achieve some things and not able to achieve some other things.
R: When you say you achieved some things, can you be more specific and tell me what things were you able to achieve in this problem solving?
S2: problem solving I was able to apply my knowledge and there were other parts where I got stuck, I did not know what to do.
R: How did you prepare for the test?
S2: I did about 3 programs. I went over them,… the structure,… all of them were different problems.
R: You say you went over, you did 3 programs, did you actually solve, design them and key the program yourself or did you just look through the solution?
S2: Normally, I must confess, I never...don't type on the computer I just write it down in my book then maybe the following day when Mr… discusses it with us as he discusses then I correct my errors.

R: For the test you say you worked on 3 programs, when you say worked did you solve the problems using the computer or did you just look at the solution you got and went through the solution and say this is how it's done?

S2: Well I compared
R: So you did not actually sit and redo it on your own? You had a solution already
S2: Yes I looked at the solution.
R: How did you interpret the problem calculating the compound interest?
S2: Well you mean the way I did it?
R: What did you think about first as soon as you got the problem/test?
S2: First I read through and then after I finished I had an idea of what to do.
R: Did you plan it on paper
S2: First I designed two classes, then I decided to use one class. Different methods for the two interests
R: Did you use 2 different methods to calculate the compound interest?
S2: Yes the problem wanted an output for the two different interests.
R: What is the method? Isn't it to calculate the interest? So whether you use 15000 or 10000 the calculation should be the same method should calculate the interest?
R: Why did you use two different methods?
S2: That's how I solved it.
R: That's how you interpreted it.
S2: Yes
R: How many objects did you create?
S2: I tried to use two objects in the main method, but I couldn't
R: So you don't know whether the program worked?
S2: No
R: What is the most difficult or trying part to work out this problem?
S2: Trying to figure which was the better interest. That's where I spent most my time.
R: In general what do you think about programming and problem solving?
S2: It's not easy, but its challenging the more you do it the more learn.

R: You've just written a test. What did you think about the test?
S3: It was nice
R: What do you mean by nice?
S3: We covered the work in class.
R: How did you learn for the test?
S3: I looked at other examples from what we worked and got an idea of how to do it.…like specific order which you do, like what's needed
R: You mean the structure
S3: Yes
R: Was the problem anything like you did in the lectures. Was it similar?
S3: Similar, He took little bits of different questions and put them together.
R: How did you interpret the problem?
S3: I managed to interpret it.
R: Think back and trace back your steps. What was the first thing you did when you saw the problem?
S3: I read the question and saw what needed to be inputted, what's needed first what processes were needed and what will be the output.
R: Were you able to create the objects?
S3: Yes
R: And the constructor?
S3: Yes
R: Did your program work?
S3: I don't know, it had utility problems…So I couldn't say.
R: So you didn't run your program
S3: No didn't run ...., I ran it, but there were other problems

R: Logical problems?
S3: Yes
R: So you are pretty confident that what you did was right?
S3: I simply tried
R: When you say there were other problems, what problems were they?
S3: I don't know, t couldn't locate the utilities that means it was telling me keyboard class error. I didn't actually see the output
R: So you didn't see your output, but you know what you did was right?
S3: I know what I did.
R: What was the most difficult or challenging part of the question that you had to think really about here.
S3: The formula, and the other part was the output. I wasn't too sure about the … to get which one is higher from the two, the two objects
R: Were you able to determine the last part
S3: I think I was able to …(inaudible)
R: But if you think about it you got invest1 and invest2, you had to find the interest1 and interest2 and then compare them
S3: Yes, I found the interest for both but didn't do the comparing part, I 'm not sure of that part.

R: What were your feelings about the test?
S4: Okay
R: Just okay? Is there more that you can tell me?
S4: I expected something harder. Only one example came out. It's the only example I did from the learning.
R: Do you mean he gave you a list of examples to go through and this was one of them?
S4: Yes
R: How did you prepare for the test?
S4: Just looked at the examples and wrote out the solution
R: How did you interpret this particular problem?
S4: What do you mean?
R: What's the first thing you did when you saw this question that you needed to solve? Can you trace back your steps that you did?
S4: Okay, first I wrote down the… that calculating the interest part first. First I wrote how you write it in Java, and then I started the main program input, then call for all methods, and output, then I just started typing.
R: Did you make use of objects?
S4: Yes
R: So you did not have a problem of calling methods using the objects
S4: No, everything worked.
R: Everything worked out? So you ran the program.
R: Was there anything you found challenging, difficult, confusing that just bogged you for a little while.
S4: mm…I made lot mistakes, the only thing
R: Mistakes in which respect?
S4: Like missed out semicolons and brackets. The other thing was for the display, when we output it I think it was supposed to be static, I'm not sure, because not each object is not going to be output separately; we output in one so I made that method static, so I don't know if its right that's what I had problems with.
R: But it still worked?
S4: Yes it still works
R: What does static mean?
S4: Static is… the method does not belong to the object, it belongs to the class; in that problem you have two objects, but in the output you give which is the better of the two. That's why I choose the static method.

R: Can you tell me what your feelings are about the test?
S5: It wasn't that surprising but often I did get a problem with parameter passing.
R: How did you prepare for the test?
S5: I worked on examples, previous examples and examples the lecturer had given us And…
R: Mostly just examples the lecturer had given you…?
S5: Yeah
R: Was it similar to what you had in the test?
S5: Yes it was similar
R: Do you think if you were given a completely different problem from what you've been going through, would you be able solve it?
S5: I would try but not certain it would work.
R: How did you interpret this problem? Did you understand all aspects of what was required?
S5: I did because it was… compound interest. Ja...and it was quite logical
R: What steps you followed in order to solve the problem? Think back.
S5: I made my class and I was thinking of what type of methods I use for my objects, And I created those
R: Did your program work, run?
S5: It did run even though... I don't know how.
R: Did it give you the desired output?
S5: I think so
R: You think so?
R: So you were able to compare the two investments?
S5: Yes I was
R: Why are you not confident, you say that you think you did?
S5: Because I did a constructor first but I the constructor did not pass the values...(inaudible)
R: How did you get around it?
S5: I put two parameters on both I initialize a new object and the actual method
R: What was the most difficult part or was there any difficult part?
S5: There wasn't but I don't know why it did not work out, I think there was something to do with the constructor.
R: So you think the constructor was the problem. Did you check the number of parameters, the type and the order?
S5: yes

R: Good morning… You've written this test 2 days ago. I want you to cast your mind back and try to visualize what happened when you wrote the test.What were your feelings about the test?
S6: When I saw it I was surprised but happily surprised because we... it wasn't the same type of question…the examples we were going over last week,...that was good and...
R: Are you saying the test had similar examples to what you had gone over?
S6: No... it, that was a good thing it wasn't the same, the idea was the same.Luckily over the weekend I got the idea in my head, the concept, how this OOP system works. The good thing is it wasn't the exact same… or similar program as what we were doing last week ...but the idea was obviously the same. It was still an object-oriented program.
R: How did you prepare for the test?
S6: Well I went over 3 specific examples that we had done last week that Mr … had gone over with us as well. I broke it down over the weekend. I printed them all out and looked for similarities between the three. I managed to break down each of the programs into their structure. By doing that it finally clicked in my head how this whole object-oriented programming works.
R: What kind of problems did you go over? Can you recall?
S6: Yes. There were three. One was the split minutes and where we had to receive an input in minutes or seconds and then convert it into days, hours and minutes.
R: You receive one input from the user, like a date and time?

S6: You receive just one input from the user, either in minutes or seconds and from that break it down using methods.
R: And the other examples?
S6: The others were average where we received3 inputs, a test mark, an assignment mark and exam mark from the user and from that we had to determine whether the student passes or fails
R: Were you able to, how did you interpret the problem?
S6: Mmm,…The nice thing was even after speaking to Kirsti as well, I put all the programs out, after printing the programs out, I looked at them, wrote on them and looked at the structure of each program. In the class for eg. We initiated the class, and we then had a constructor, followed by each receiving method… and then in the main class we created the objects and passed our variables to the class and after that we called for each method that made sense to me.
R: Why did you follow this method?
S6: Because, well I tried everything. After trying everything… after speaking to the tutor and Mr…on numerous occasions, it just wasn't sinking in,… it really wasn't helping me , my grades wasn't good as well for the last few tests and somehow or the other after speaking to Kirsti and trying this new system out, it seemed to make sense..
R: Did your program run?
S6: No unfortunately and this is thanks to or no thanks to Mr …
After I understood the problem I decided to have nothing in the main method, but my object creation for instance initializing I was going to call for the different methods that I needed. In other words, in the class itself I had a method for display heading, a method for input request and a method for working out. The only thing that was going to be in the main method was the output statement and the calling of the methods. So I asked him, I can't remember what I had asked him about, and he came over and looked at my program. He said, what are you doing, why are you putting the display heading in the class it should be… or shouldn't be there.. So I realized it can be, normally it is in main method.
I also realized for neatness and my own sake I didn't mind it being in the method because I was calling for the input statements twice I had two separate objects. Subsequently I changed that and couldn't see it run… there wasn't enough time to make anymore changes?
R: So you didn't have enough time to see it through. Al right, now that you are going to go through the test with Mr… you will be able to see where you went wrong.
S6: Yes.

R: What were your feelings about the test?
S7: hmm… I was a bit confused with my objects, but… I didn't think it was too bad.
It wasn't a complicated program that he set for us to write.
R: How did you prepare for the test?
S7: Just did questions that Mr …suggested that …we did programs
that he suggested we try to do. We went over the ones that he done with us in class.
R: Was it anything similar to what you've done in the test?
S7: I hadn't done that one.
R: But the similar kind?
S7: Yes. Creating objects
R: How did you interpret the problem? How did you go about solving the problem?
S7: Oh well, I first figured out what kind of methods I needed, I needed to create two objects, then broke down the methods then just programmed it.
R: So you mean you did planning on the paper first.
S7: Yes.
R: Were you able to grasp the formula for compound interest? Did you understand how that works?
S7: Oh Yeh, because we did compound interest last year in one of our programs.
R: Why did you go about solving it the way you did? Is there any particular way or is it the only way?
S7: No, Yes it's what makes sense to me, it's what I did

R: I just heard that you could not actually get it to run.
S7: I couldn't get to run. I could not use the object to call one of the methods; it said that my syntax was wrong. So I'm not sure… something was wrong there.
R: So you couldn't actually solve that aspect. But you know how to call a method using an object.
S7: Ja, I just didn't have the syntax right. I don't know what I did wrong there.
R: You couldn't figure it out.
S7: No.

Interviews from the Pilot studies

S8: My biggest problem was... it says here it pays 5% interest whether it was 5% per annum or monthly? It just says it pays 5% interest. I just read it as is but later I asked Marcian and he said you should have used a repeat statement or whatever…but I did not understand what the question asked for. That's my biggest problem.
R: Do you find that that being a general problem when solving other problems or is it just in this particular situation?
S8: Well you know…I just been introduced to programming and I'm understanding what they asking for but a lot of the examples that are given to us I don't quite know what its asking…so I just feel if I do it this way is that going to get me the desired result and that's the basic thing. I'm not sure if I'm going in the right steps. I'm hesitant to start off because I'm not sure exactly what's required.
R: So you think the problem is reading the problem and understanding it, interpreting the problem, that's your main problem?
S8: Yes
R: But once you've interpreted the problem do you think you could actually solve it. You think you could actually implement it in Delphi code?
S8: Yes, I think I can. I've been working on it. I know I'm a little bit behind to what we are doing in class. I'm sure I can work this out. I thought I'd done this perfect,…and then I found out it was countdown that was because I did not understand what it was asking for.
R: But I think you have, according to what you've done in the test you've actually did what was required. It is this portion here where you had to say that if thousand rand is withdrawn every year, according to you you've divided by a thousand rand.
S8: You take off a thousand rand
R: But do you divide or subtract?
S8: Take that amount, you divide it by that.
R: Every year you are withdrawing?
S8: Yes, no I understand that but I was saying that in 15 ...you've got R15000
R: And at the end of the year you want to withdraw R1000, what will you do?
S8: You subtract it off, yes sure off-course, but the way I worked it out was I said okay, I've got R15000, you take 15 divided by how much you take out every month and you get how many months you have. Look, I realize I've done it wrong, but…
R: Okay,...What I'm saying generally have to go step by step to test and you've got to analyse the problem.
S8: Yes.

R: We are referring to this problem of calculating the number of years until the balance becomes zero. What is the major difficulty you had in solving the problem?
S9: I guess applying the rules of fixed deposit interest to get the… I understood the concept of finding the value until the sum was 0. But at first I was doing a fixed deposit and then I realized it needed compound interest. Because, yes I was confused

R: Do you think there was confusion with interpreting the problem or the problem wasn't clear enough?
S9: Oh the question?

R: Yes
S9: Can I read the question?
R: Yes
S9: I did say it was ambiguous because it doesn't say whether the 5% increases annually or monthly, but the thing is if you look at the diagram if anything has to tie in, it has to tie in with the number of years; that was crucial.
R: Would you say that whenever we talk of interest in everyday situations do they give you interest per month or is it by default interest per annum?
S9: I guess its interest per year.
R: Yes, per annum. Basically that was a little bit of confusion
S9: Yes, they should have added that in the question. Otherwise I thought the question R: was straight forward.

R: The purpose of this interview is to determine what problem you encountered in solving the problem and how did you go about solving the problem? Were there any major difficulty?
S10: The problem itself was pretty basic. But I…where I found it hard was I felt under pressure. I looked at the format where you had a real number and I was trying to figure out how to get …how to use a real number and that caused me to leave out things like using the if… then… statement to make sure that the input they put in is less than 15000. So in the end I still could not figure it out, so I left it as an integer. Other than that it was okay.
R: So you were able to solve the problem to an extent?
S10: Yes to an extent
R: You understood what was required?
S10: Yes
R: Is there any other aspects you have in general with regard to problem solving computer problems?
S10: No not really, in fact I enjoy it.

R: The purpose of this interview is to determine what problem you encountered in solving the problem and how did you go about solving the problem?
You had to calculate the number of years before the bank balance becomes 0.
You didn't seem to get it right completely.
S11: According to what the question says, the mathematical aspect of 5% of the deposit
R: You mean you didn't understand what that 5% meant?
S11: Yes I didn't understand.
R: The 5% means that every year you will earn interest on the amount of money you have in the bank. They will have to calculate a certain rate of that amount. So in this case they will have to calculate 5% of this amount
S11: Every year?
R: Yes, every year. What did you interpret it to be? What did you think the 5% was?
S11: I thought the 5% was just once, not every year, just once.
R: So that's what you did? Do you realize, it says here how many years, because nowhere in your program code have you got something to say count, because the question says: "Write a program that reads an amount deposited into a savings account that pays 5% interest. And then you have to calculate,… the program must calculate how many years it will take for the account to be depleted. Because every year they are also withdrawing R1000. Did you not see that part of the question or you just didn't understand?
S11: Mainly I didn't understand.
R: Okay, so it's mainly reading and understanding the problem is the major difficulty for you before you can actually implement it in or rather code it?
S11: Yes.

R: The purpose of this interview is to determine what problem you encountered in solving the problem and how did you go about solving the problem?

S12: Basically the problem was not clear
R: What aspect was not clear?
S12: Like the way we should be working and the final output or the runtime what should happen? …(inaudible…)
R: You mean you have no idea of what is required here?
S12: I see what is required but the way the question is worded
R: Okay, do you know what the input was supposed to be?
S12: The amount
R: Do you know what the output should be?
S12: I think it should be the number of years.
R: So you seem to know what input and the output are required, did but the processing is the problem?
S12: Yes I didn't know how to calculate the interest.
R: And how to get the new amount?
S12: Yes
R: What did you do?
S12: I used the spinedit, then I tried to multiply by the percent 5% and subtracted the actual amount. I couldn't go any further.

Appendix E

Guide to interview questions for teachers/lecturers in the study

How did you get involved with teaching programming?

How long have you been teaching it?

What o you see as the purpose of teaching programming to these students?

What do you hope the pre-service students will gain from this programming course?

How do you know if they have understood aspects of programming or programming in general? Can you give some concrete examples?

Do you see differences in the groups you teach?

Are there any constraints on you? In other words is everything ideal or are things done because of various constraints?

How much control do you have over the way the course is presented?

How do the assessments affect the learning?

What do you see as the different goals of the students you teach?
I'm interested in how this relates to what students think the programming course is about.

How do you feel about teaching programming?