LEARNING COMPUTER PROGRAMMING IN IRISH THIRD-LEVEL INSTITUTIONS: A STUDY OF FIRST YEAR STUDENTS’ EXPERIENCES

Thesis submitted to the University of Leicester for the degree of Doctor of Education

by

Enda Noel Duncan B.Sc. (Hons), M.Sc.

December 2006
Acknowledgements

There are many people I would like to thank for their help and support during this thesis. Firstly, I would like to thank all of the staff at the School of Education in Leicester, with special thanks to Julie Thompson and Leigh Blair. I would also like to thank the academic staff who gave me help and encouragement during my studies. To Prof. Paul Cooper, for acting as my initial supervisor and setting me off on a very enjoyable and educational journey. To Prof. Clive Dimmock for his expert help on Grounded Theory. To Dr. Roger Merry for his incessant enthusiasm and willingness to help all of us on the Learning and Teaching strand.

I would also like to thank my classmates Liane Purnell and Joan Fox for their friendship and help. I would particularly like to thank Liane for keeping me informed about everything that was important on the course and her infectious work ethic. I would also like to thank my friend Prof. Thomas Green for his help and encouragement since I have embarked upon this journey of studying novice programming. A sincere thanks must also go to Juliet Corbin, whose constant willingness to help me on the methodological issues surrounding grounded theory was of vital importance to me and has not been forgotten. I would like to thank all the participants in this study and the individuals at each institution who facilitated me in every way possible.

I am deeply indebted to my supervisor Prof Stan Tucker for his professionalism, patience and expertise. His expedient and insightful feedback on my papers and thesis was of immense help to me in completing this course of study. I could not have had a better supervisor.

Finally, the biggest thank you must go to my wife Paula, for her unconditional patience and support over the last three years. We can now look forward to life post Education Doctorate.
Abstract

Over the last three decades there has been a significant amount of research conducted in the area of novice computer programming. Within the Irish context however, there has been limited amount of research in this area with the vast majority of studies conducted being quantitative in nature. Given the fact that Ireland is becoming hugely dependent on its knowledge economy, the attrition and non-completion rates in computer science reported in both Irish and international studies is becoming a cause for concern. In this study, the actual experiences of Irish novice programming students are presented in the form of a theory and the implications of this theory for the teaching and learning of the subject are discussed.

This thesis presents findings from research conducted across four Irish higher education institutes with 31 participants studying first year computer programming as part of a formal course in computing. Data collection and analysis was conducted using the Strauss and Corbin (1988) variation of the Grounded Theory methodology. This model was chosen based on its suitability for inductive theory generation and learner type categorisation. The outcome of this study is a twofold. Firstly it generates a grounded theory containing the major phenomena experienced by first year programming students. Secondly, it uses this theory to identify a number of novice programming learner types presented in the form of a learning continuum.

This thesis analyses the various phenomena experienced by students as they progress through a programming syllabus and how the challenge posed by the subject may affect them in both positive and negative ways. Furthermore, it will discuss how a student may not only experience a number of learner type categorisations but may also oscillate between them.

The thesis concludes by discussing the implications the findings have for teaching and learning programming subject in the Irish context and identifies possible directions for future research.
Key Words

Novice Computer Programming, Learning, Grounded Theory, Categorisation.
Dedicated to my parents, Thomas and Esther Dunican
# Table of Contents

## Chapter 1 Introduction

1.1 Introduction 1

1.2 The Irish Educational System 2

1.3 Computer Science in the Irish Education System 3

1.4 Research Problem Being Addressed 4

1.5 Aims of the Research 5

1.6 Objectives of the Research 6

1.7 Motivation for the Research 8

1.8 Overview of the Thesis 9

## Chapter 2 Literature Review

2.1 Introduction 11

2.2 Literature Review in a Grounded Theory Study 12

2.3 Learning Computer Programming 16

2.4 Teaching Programming to Novices 20

2.5 Qualitative Studies in Novice Programming 24

2.6 Categorisation Studies in Novice Programming 36

2.7 Major Issues and Challenges 44

## Chapter 3 Methodology

3.1 Introduction 46

3.2 Suitability of Grounded Theory 47

3.3 Grounded Theory Overview 50

3.3.1 Open Coding 51

3.3.2 Theoretical Sampling 53

3.3.3 Theoretical Memoing 54

3.3.4 Axial Coding 55

3.3.5 Selective Coding 56

3.3.6 Category Development 57

3.4 Underlying Issues in Grounded Theory 60

3.5 Theory Forcing and the Paradigm Model 65

3.6 The Research Question 68
List of Figures

Fig 2.1 Categorisation Approach in Thesis Study 35
Fig 2.2 Progressive Categorisation in Phenomenographic Studies 35
Fig 2.3 Visualisation Using Jeliot Environment 43
Fig 3.1 Abstraction Levels in Grounded Theory 58
Fig 3.2 Categories and Subcategories 58
Fig 3.3 Abstract Categories 59
Fig 5.1 Open Coding 108
Fig 5.2 Theoretical Memo 108
Fig 5.3 Category Abstraction Using Association Diagrams 113
Fig 5.4 Class Diagram 114
Fig 6.1 Open Coding Example 120
Fig 6.2 Progression Strategies 123
Fig 6.3 Cross-Phenomena Association Diagram 124
Fig 6.4 Memo on 'Eureka' Moment 130
Fig 6.5 Mini-Framework 132
Fig 6.6 Avoidance Strategies 134
Fig 6.7 Perception of Programming 141
Fig 6.8 Programming Organisation 152
Fig 6.9 State of Mind 167
Fig 6.10 Central Category 175
Fig 6.11 Learning Continuum 190
Fig 6.12 Concept Generation – Micro Analysis 218
Fig 6.13 Positive Feedback Loop 224
List of Tables

Table 2.1 Learning to Program Categories 27
Table 2.2 Categories of Learning to Program 30
Table 3.1 Paradigm Model 56
Table 3.2 Paradigm for Abstracted Category 'Solution Strategies' 66
Table 3.3 Natural Unforced Questioning 67
Table 4.1 Data Collection Overview 74
Table 4.2 Data Collection Participation Summary 87
Table 4.3 Common Programming Course Structure 89
Table 5.1 Category Overview Table 111
Table 6.1 Learning Concepts 118
Table 6.2 Open Coding Categories 121
Table 6.3 Progression Strategies 133
Table 6.4 Avoidance Strategies 137
Table 6.5 Perception of Programming 142
Table 6.6 Programming Organisation 154
Table 6.7 State of Mind 166
Chapter 1

Introduction

• Introduction
• The Irish Educational System
• Computer Science in the Irish Education System
• Research Problem Being Addressed
• Aims of the Research
• Objectives of the Research
• Motivation for the Research
• Overview of the Thesis

1.1 INTRODUCTION

The purpose of this study is to examine in detail the learning experiences of Irish first year third-level computer programming students. In this context, the term ‘third-level’ relates to Irish higher education institutions, namely universities and Institutes of Technology. Computer programming is a subject that appears on practically all computer science courses in Irish third-level institutions. Research conducted within Ireland (Morgan et al., 2001), and internationally (Rountree et. al., 2004; Hagan and Sheard 1998) indicates that this subject has often experienced a higher attrition rate than is desirable. In particular, Beauboeuf and Mason (2005) report attrition rates in the 30-40% range whilst Wosczynski et al. (2005) highlight that in some cases it
Chapter 1 Introduction

exceeds 50%. In many cases, this has been attributed to a plethora of factors such as
difficulty with syntax, poor problem-solving skills and a host of others that will be
explored in this thesis.

With the exception of the recent study by Stamouli and Huggard (2006) novice
programming in Irish third level institutions is an area where qualitative research
appears to be lacking, with most published research to date being of a quantitative
nature (Bergin & Reilly 2005; Daly & Waldron 2004; Hyland & Clynch 2002)
reflecting a task-oriented approach where students fill in questionnaire-type forms or
engage in a programming test of some nature. This is consistent with the assertion of
Yuan (1997) who indicates that most end-user studies have focused on descriptions of
behaviour or performance on a particular set of tasks. Whilst this type of research is
very valuable in identifying what is actually happening within these educational
settings, it only presents a partial picture. In this regard, it seems logical that these
quantitative studies can be complemented by qualitative research that aims to
ascertain why certain issues actually occur in a given context. This study utilises the
grounded theory qualitative research method (Strauss and Corbin, 1988) in an attempt
gain an insight into the why question. This research method has not been used within
this context before and its implementation-specific issues and results are presented in
this thesis. In terms of importance, a project of this nature and scope has not been
conducted previously in Irish third-level institutions. In this regard, it helped to gain
an insight into both the common experiences shared by Irish students and their
international counterparts as well as the specific issues that exist within the Irish
context.

1.2 THE IRISH EDUCATIONAL SYSTEM

The Irish educational system comprises primary, secondary and tertiary (third-level)
education under the auspices of the Department of Education and Science. Formal
state examinations comprise Junior Certificate (equivalent to the English O-Level)
and Leaving Certificate (equivalent to the English A-Level) which are taken by
students at second-level. Third-level education is provided by both further education
and higher education institutes. Further education is mainly provided by Post-Leaving Certificate colleges attached to secondary schools. Higher education is provided by the Institutes of Technology and Universities. The primary difference between Institutes of Technology and Universities is that the Institutes offer higher certificates as well as undergraduate and postgraduate degree programmes. This research was conducted at four Irish higher education institutions, where all participants were either studying for a higher certificate that led to a degree or for a degree in computing.

1.3 COMPUTER SCIENCE IN THE IRISH EDUCATION SYSTEM

Computer science is not a provided as a formal subject in the Irish second-level curriculum. At the Junior Certificate level, the only subject that deals with computer science related material is Technology. This is a general subject that includes a diverse mix of topics such as ‘communications’, ‘craft and materials’, ‘energy and control’ and ‘technology and society’. Within the ‘communications’ topic there is a subsection that introduces computing and covers simple Information Technology issues such as input/output, saving files and use of graphics (where it is mentioned on the syllabus that graphics can be developed by either using software packages or programming). Within the ‘energy and control’ topic the syllabus refers to the fact that programming is used in robotics. These two instances are the only reference to computer programming in the Junior Certificate syllabus.

Currently, there is no formal computer science subject provided at Leaving Certificate level. In the recent past, the National Council for Curriculum and Assessment (NCCA) has developed a ‘Technology’ subject that will be offered to Leaving Certificate students from September 2007. Within this syllabus ‘Information Technology’ (IT) is a core module with ‘Information and Communications Technology’ (ICT) being offered as an optional module. The Information Technology module exposes the student to introductory material such as the basics functions of a computer system as well as application packages such as word processors and spreadsheets. The Information and Communications Technology module focuses more on computer science type material such as memory, hardware, software, and
networking. Whilst these topics provide the student with a good introduction to some of the subjects covered on a third-level computer science course they do not expose the student to any form of computer programming or problem solving. The potential ramifications of this situation will be discussed in this thesis, with particular reference to issues such as students not fully appreciating what a computer science course actually entails and the particular problems associated with being exposed to abstract or alien concepts for the first time for which mental schemes have not been developed.

Finally, recent research commissioned by the Council of Directors of the Irish Institutes of Technology (Flanagan et al., 2000) indicates that courses in the area of mathematics, science, engineering and technology have much higher rates of non-completion than other academic disciplines such as business and humanities. Furthermore, it concludes by indicating that this situation has important implications not only for the students taking the courses and the respective institutes but also for the Irish knowledge economy that is heavily reliant on skills and expertise developed in these areas. In this regard, it is clear that improvement in the retention rates of computer science courses is an issue of concern for Irish educational institutes and as such, research in the area of computing/technology may provide a key role in achieving this.

1.4 RESEARCH PROBLEM BEING ADDRESSED

The primary objective of this research was to get an in-depth understanding of the learning experiences of first year computer programming students in Irish third-level institutions. In this regard, all issues that impinged on the learning of programming were open for discussion with participants. This ranged from emotional issues like an overt awareness of the progression of their peers to specific strategies that facilitate accommodation such as seeking out code examples that clarify a particular programming concept e.g. parameter passing in methods. This differs from previous qualitative studies in computer programming to the extent that the research question is phrased in such an open fashion that any part of the learning experience could be
explored and subsequently developed into phenomena. This is contrasted to previous studies which have a slightly more focussed question such as 'what does it mean to learn programming?' This type of research question appears to be more concerned with illuminating the steps of programming concept accommodation over a number of identifiable phases. In this regard *programming concepts* refer to issues like input/output, memory variables and various programming control structures that determine the sequence of program execution. The phenomenographic studies that adopted this more focussed approach will be examined in chapter 2. Furthermore, the open nature of this research question and its scope for exploration of any issue that influences programming learning resulted in more straightforward sample selection. In particular, any participant engaged in a first year programming subject as part of a formal computing course who volunteered for the study was a theoretically rich candidate who would bring to the data collection process a unique and individual accumulation of programming learning experiences, each of which could be used to saturate emerging theory (Strauss and Corbin, 1998). The concept of saturation will be discussed in chapter 3. It should also be noted that all types of research question, irrespective of their particular focus, are important, with each one having the potential to provide computer programming educators with another part of the jigsaw in terms of understanding the issues encountered by students as they go about learning programming.

### 1.5 AIMS OF THE RESEARCH

This research project had two primary aims. Firstly, it endeavoured to develop a grounded theory based on the learning experiences of novice programming students using the grounded theory research method. In specific terms, a grounded theory study develops theory that is based on the experiences of the participants who took part in the study. It does not claim to develop a *formal theory* that is applicable to all contexts outside of that observed, but rather a *substantive theory* that is developed from work in a particular area (in this case, learning experiences of programming students in a selected number of Irish third-level institutions). In this research, a major aim was to develop substantive theory that is consistent with the 'parsimonious'
requirement described by Goulding (2002) where she indicates that the researcher should not endeavour to generalise with explanations of situations for which there are no data. This assertion is consistent with another grounded theory concept, i.e. that of ‘fit’, a notion which is seen as a desirable characteristic of GT where the theory generated ‘fits’ the actual data collected and is not derived from bias or other non-data-related sources (Glaser and Strauss, 1967). The resultant grounded theory will take the form of a central category that encapsulates phenomena uncovered in the data. This theory, if developed properly by paying close attention to the data will provide a detailed insight into the actual experiences and phenomena encountered by students. In addition, given that one of the main objectives of the grounded theory method is to not only uncover categories but also to examine the relationships between them, this study will attempt to show how the experience of learning programming can be viewed as a complex interaction of inter-related phenomena.

The second aim of this study was to develop categorisation/taxonomy of novice programmer learner types. This categorisation is facilitated in grounded theory by analysing the varying manifestations of phenomena and their sub-categories in terms of variance in their categories and dimensions. Specifically, each learner category on a learner type continuum will exhibit a certain set of attributes that is manifested in the form of dimensional variation relating to properties of inter-related categories. Whilst these categories may exhibit congruence with categories developed in other studies uncovered in the literature, they are solely based on the data. In this regard, one can see an overlap in the two primary aims of this research whereby the learner type classification is also substantive in nature.

1.6 OBJECTIVES OF THE RESEARCH

Achieving the aims set out in the previous section was predicated on a number of specific objectives:

- To gain, through rigorous data collection, an insight into the learning experiences of Irish third-level programming students, in terms of how they actually
experience it i.e. in their own voices. This insight will take the form of a grounded theory comprising a central category that encapsulates phenomena.

- To identify re-emerging patterns of learning experience that can be categorised and used in theory generation and learner type classification.

- To develop categorisation/classification of theoretically saturated student learner types based on the theory (by examining variations in the sub-categories of phenomena in terms of their properties and dimensions uncovered). This categorisation will encapsulate the varying learning profiles encountered in the data.

- To develop and present the categorisation in such a way that will enable educators to deal with actual learning experiences of students as opposed to relying on anecdotal evidence or perceived experiences.

It must be noted that this research endeavoured to look at the different learning types/profiles experienced in a number of first year programming cohorts. In the thesis proposal of this study it was explicitly stated that this research would not be suggesting that a student's learning profile cannot change over time, but rather its purpose was to isolate and identify the various learning types and their attributes. These attributes will be represented by a set of interacting categories that vary in terms of properties and dimensions. In fact, chapter 6 supports this stance where it will show that not only can a student experience a number of varying learning categories throughout their first year studies but also that they can oscillate between these varying states. It is hoped that gaining an in-depth understanding of saturated learning types will provide computer science educators with invaluable insights in terms of suitable course provision and delivery. Whilst the research question was open and left room for any programming experiences to be analysed, there were a few key questions that the author was interested in addressing:

- What are the actual learning experiences of Irish first year programming students (as opposed to those perceived by educators)?
- How do students manage the programming subject throughout the academic year?
- How do they approach their learning of the programming subject?
• Are there clear recognisable differences between highly performing and poorly performing programming students? If so what does the categorisation tell us about it?

Whilst this list poses a series of initial questions of interest, subsequent chapters will show that many other interesting issues emerged out of this study. For instance, as well as the phenomena experienced by students such as their perception of programming and how they deal with the programming subject, low-level strategies such as how they deal with programming tasks also emerged. This proved to be one of many satisfactory aspects of the grounded theory methodology in this domain of study whereby many different issues (both expected and unexpected) emerged and were subsequently analysed through successive levels of data collection and analysis.

1.7 MOTIVATION FOR THE RESEARCH

The author has worked for over a decade as a programming lecturer with first year computer science students in Irish third-level institutions. This professional experience has witnessed an unsatisfactory progression rate in the programming subject that is consistent with that found in other institutes both in Ireland and worldwide. In addition, this chapter has already highlighted the fact that there is no provision for any formal computer programming education in the Irish primary and secondary school curricula. This has resulted in many students entering a computing course under the misguided assumption that it was application-based i.e. a direct follow-on from the software application focus of the second-level Information Technology (IT) modules. In this regard, the author is hoping to ascertain if this lack of computer programming provision impacts on the students’ progression through a computer science course.

Secondly, in recent years the Irish Department of Education has introduced a quality assurance program whereby each lecturer is required to distribute subject assessment forms to students. In these forms, students are given an opportunity to give feedback (in an anonymous fashion) in terms of their satisfaction with a particular subject. This
feedback form contains a number of sections that look at areas such as standard of teaching, quality of lecture notes, sequencing of topics etc. On receipt of these forms the lecturer concerned summarises them and returns the summary to the relevant school. In some cases, when analysing these feedback forms, a lecturer may see considerable variance between how he/she perceives the delivery and progression of the subject and how the students actually see it. Whilst engaged in the delivery of the course, the lecturer may be oblivious to many issues actually affecting students' experience of a subject given the fact that they may view it from an observer or 'etic' perspective. It was then decided that if one was to obtain the actual experience, in-depth qualitative data collection would be required to obtain the insider or 'emic' perspective. In order to attain this 'emic' insight, grounded theory was deemed a suitable research method whereby it facilitates participants to recount their personal experiences in their own voices (Strauss and Corbin, 1998). In this regard, the desire to understand the actual experiences of students (as opposed to those perceived by the educator) was a major catalyst for this research undertaking. This point is made quite clearly and succinctly by Milne and Rowe (2002) who indicate that 'the lack of these types of studies may be explained by current lecturers of programming simply relying on experience and intuition to inform them of what is difficult and what is not' (p.55). Finally, it is hoped that a deeper understanding of the actual issues experienced by students will pave the way for more student-focussed teaching practices and material provision

1.8 OVERVIEW OF THE THESIS

The remaining chapters in this thesis are arranged in a logical progression format whereby existing material relating to literature, methodology and analysis is presented followed by the actual analysis conducted in the study. Chapter two presents the reader with a literature review that discusses issues relating to the teaching and learning of novice computer programming as well as the previous studies conducted in this area. Chapter 3 provides a detailed discussion of the grounded theory methodology and makes particular reference to the suitability of this method to the research domain. Chapter 4 entails a detailed discussion of the data collection process
implemented in this study across four separate data collection sites. It will describe how data was collected using interviews, focus group discussions and process diaries/journals. Chapter 5 provides a detailed overview of the analysis process used in this study. It will describe how the author devised a number of techniques to facilitate axial coding, namely conglomereration and category association diagramming. Chapter 6 presents a detailed account of the analysis conducted in this study. It illustrates how the process of open, axial and selective coding resulted in the generation of a theory that comprises a central category that encapsulates a number of phenomena that emanated from data analysis. It then uses this theory to generate a programming learner categorisation continuum that identifies the different learner types that emerged from the data. Finally, it shows how the study conformed to the rigours of validity and reliability that are prescribed for this type of study. Chapter 7 will provide the reader with the major conclusions of this study and the recommendations for future work in this area.
Chapter 2

Literature Review

- Introduction
- Literature Review in a Grounded Theory Study
- Learning Computer Programming
- Teaching Programming to Novices
- Qualitative Studies in Novice Programming
- Categorisation Studies in Novice Programming
- Major Issues and Challenges
- Conclusions

2.1 INTRODUCTION

This chapter introduces the reader to the research area of this thesis, namely novice computer programming. It begins by emphasising an important issue pertaining to the positioning of a literature review on the research life cycle. In particular, it will show that there are essentially two schools of thought on this positioning in terms of before and after substantive data collection and analysis has taken place. It will then present the reader with many of the core issues in learning programming that have emerged in the literature over the past twenty years. It will then go on to present the major issues that have emerged in research in terms of teaching novice programming and highlight the views from many commentators pertaining to the inextricable link between teaching and learning. Given the qualitative nature of this study, it will then go on to
present the reader with a number of qualitative studies that have been conducted in the area of novice programming. Given the fact that one of the aims of this project was to develop a taxonomy of learner types, it then goes on to present an overview of previous studies conducted in the area of novice programmer categorisation. It will conclude by providing the reader with a clear summary of the major issues and challenges that emerged from the literature study. Finally, it was decided that the literature in this research be divided into two sections. The first section relating to fundamental novice programming issues and research are presented in this chapter (as described above). The second provision of literature material is interspersed in the analysis chapter (chapter 6) in terms of synthesising the emergent theory of this thesis study with that existing in the literature. This approach is taken so that the reader can see an immediate overlap between emergent theory and existing literature as opposed to having each covered in a disjointed fashion in separate chapters.

2.2 LITERATURE REVIEW IN A GROUNDED THEORY STUDY

The engagement a literature review, in terms of when and how it should be done is an important issue in grounded theory literature. Essentially there are two schools of thought on the positioning of the literature review in a grounded theory study and these emerge out of the two main variants of grounded theory, namely Glaser (1992) and Strauss and Corbin (1998). A detailed discussion of these variants will be presented in the next chapter on methodology. Having analysed a vast amount of grounded theory literature, this research has found that there are two main purposes of a literature study. Firstly, in the case where the researcher is not familiar with the problem domain, a detailed literature analysis will bring them up to the required level of theoretical sensitivity so that they can understand the various terminology and nuances in the data. Secondly, the literature review can be used to synthesise findings with those of previous studies as well as providing a mechanism for external validity or ‘transferability’ (as will be discussed in chapter 6). The remainder of this section will look at the various perspectives on the positioning of a literature review in a grounded theory study.
In methodological terms, literature suggests mixed views in terms of when a literature review should actually take place. Glaser (1978) advocates against collecting data in advance of engaging in a literature review in order that 'the theory will not be preconceived by pre-empting concepts' (p.31). In his book on 'emergence vs. forcing' Glaser (1992) recommends the optimal approach is to engage in a 'modicum of literature' (p.32). In this regard he postulates that:

The researcher should not worry about covering the literature in the same field before the research begins, since it will always be there. It does not go away! And there will be plenty of time during the grounded theory process to integrate this literature with the emergent theory during saturation, densifying and sorting. Especially during sorting and then writing, the researcher-analyst by constant comparison reconciles differences, shows similarities in concepts and patterns, and imbues his work with data and concepts in the literature. (p.33)

Glaser (1992) claims that the benefit of this approach is that the researcher can subsequently look through the literature quickly for what relates to emergent theory in the study, where all literature is not of equal importance and recommends the use of the 'skip and dip' approach (p.33) when reading which results in a greater breadth of coverage based on the emergent data in the study. Smith and Biley (1997) take a modified position to that of Glaser by suggesting that a general reading of the literature is acceptable as long as 'the reading is not too extensive' (p.20).

Strauss and Corbin (1998) suggest that a literature review can be conducted in a prudent fashion in advance of a study. They indicate that 'familiarity with relevant literature can enhance sensitivity to subtle nuances in the data just as it can block creativity' (p.49). They suggest that concepts derived from the literature can provide a source for making comparisons to the data at a dimensional level in terms of properties of categories. This enables the analyst to present variations and give

---

1 This was a technique used by the author where engulfed with a large amount of literature the researcher 'dips' into various sections of literature that pertain to emergent theory and 'skips' those that don't. This allows more time to analyse a wider breadth of literature.
specificity to an emerging concept. They also suggest (in direct variation to Glaser) that the literature can be used to formulate initial questions in the form of a 'stepping off point' (p.51). Subsequently after the first interview(s) the researcher can turn to questions that emerge from the data in subsequent data collection. Furthermore, where there appear to be discrepancies between the literature and emergent data the researcher can ask questions like 'what is going on?' or 'am I overlooking something important?' or 'are conditions different in this study?'. In addition, they suggest that literature can direct the researcher towards places where certain relevant concepts to emerging theory can be found. By this they mean that the literature can lead the researcher to places that he or she might not have considered. As regards overlap with Glaser (1992) they suggest that literature can be used to confirm findings or can be used to identify where the literature is incorrect or at variance with the study under investigation. In essence, Strauss and Corbin (1998) recommend prudent use of literature and advise the prospective researcher that 'literature can hinder creativity if allowed to stand between the researcher and the data' (p.53). On the other hand, 'if it is used as an analytical tool it can foster conceptualisation' (p.53). In this regard, they can see both the positive and negative aspects of literature reviews conducted in advance of initial data collection.

In terms of computer programming education research analysis of the literature indicates that researchers more often than not possess a priori knowledge of computer programming acquired from both literature and professional experience, and therefore it is incumbent upon him/her to adopt the aforementioned prudent approach so that the identification of new phenomena is not stifled by this pre-existent knowledge. This was the case in this study. The author has not only worked as a programming lecturer for over a decade but as part of his active participation in The Psychology of Programming Interest Group\(^2\) (PPIG) has amassed a considerable amount of programming education-specific literature over the years. In order to overcome this potential pitfall some commentators (Franchuk, 2004; Backman, 1999; Rennie, 1998) document the use of bracketing, a technique whereby the researcher attempts to suspend or set aside their prior knowledge, attitudes and beliefs about the phenomenon under study.

\(^2\) Interested readers can find details about the work of this group at www.ppig.org.
Given this background in terms of professional and literature experience, the author decided on two particular issues. Firstly, given many years of teaching experience in the area of novice programming engaging in reading of the literature in advance to acquire ‘theoretical sensitivity’ (a grounded theory term described in the next chapter) was not required. Secondly, given prior exposure to much novice programming literature, the author took a deliberate decision not to engage in any formal reading of the literature in advance of the study. This is consistent with the assertions of Glaser (1992) in terms of theory forcing where the concepts developed in the study might be construed as being preconceived or forced. In effect, it might have had the effect of looking for data and saturation only in certain places as prompted by the literature.

In this regard, accumulating a detailed number of concepts in a detailed literature review may have resulted in looking only for certain concepts, perhaps ‘fishing’ for them in interviews and only asking follow-up questions pertaining to emergent data that was consistent with the literature (and thus may leave the theory open to the criticism of having been ‘forced’). This decision is also consistent with one of the issues raised by Strauss and Corbin (1998) where they indicate that in some cases the literature can stand between the researcher and the data and stifle creativity. In this case, the author refrained from detailed analysis of the literature for fear that it would have lead to a pseudo-deductive process, which would fly in the face of inductive qualitative enquiry as prescribed by grounded theory. Finally, as data analysis was progressing the author engaged in a detailed literature review to synthesise the emergent data with that found in the literature using the ‘skip and dip’ approach described by Glaser (1992). This literature analysis uncovered considerable overlap with many concepts that emerged in this study and served as a form of verification. Strauss and Corbin (1998) also see benefits in this type of approach where they indicate that ‘later, in writing up their findings they can make comparisons describing how their conceptualisations of data extend or fit with the existing literature’ (p.156).
This can be seen in chapter 6 where synthesis of literature material with emergent concepts is interspersed at the appropriate places where the individual concepts and phenomena are being presented.
2.3 LEARNING COMPUTER PROGRAMMING

The learning of computer programming by novice programmers is an area of research that has been undertaken over the last three decades. Programming as a learning activity differs to other academic undertakings given the specificity of the demands it imposes on the novice learner. Samurcay (1989), gave a definition of this specificity that holds true today:

The specific nature of programming as a problem-solving situation can be summarised in a few points: the goal of the activity is not only to produce a solution as in classical problem solving situations, but also to make explicit the procedure producing the solution; the activity is highly mediated by a technological tool for which the subject has to construct functional representations; the elaboration and the expression of the procedure in an “action code system” necessitates the acquisition and use of specific programming concepts. (p.162)

Samurcay (1989) goes on to indicate that this specific nature of computer programming as a learning activity has important effects on the learning process. Specifically, learning programming cannot be considered only in terms of acquiring syntax of a language. In particular, it requires a level of mental representation of a problem to be solved and its solution. These mental representations and the sequence of actions required to put them into an appropriate programming solution are referred to as ‘programming plans’ (discussed later in this section).

The learning of computer programming is well documented over many years as being a difficult and complex task. Rogalaski and Samurcay (1990) highlight that ‘acquiring and developing knowledge about programming is a highly complex process’ (p.157). The more recent research conducted by Pane et al. (2001) concurs with this assertion:

Programming is a very difficult activity. Some of the difficulty is intrinsic to programming, but this research is based on the
observation that programming languages make the task more
difficult than necessary because they have been designed without
careful attention to human-computer interaction. (p.237)

Perkins et al. (1988) back up this viewpoint by contrasting programming to other
academic subjects (many of which are interspersed with so-called ‘trouble spots’) by
indicating that programming is a subject that ‘is almost all trouble spot’ (p.154). In
another project, Perkins and Martin (1986) refer to the ‘challenge’ of computer
programming as an academic subject:

Consider for a moment what a challenge programming is.
Unlike most school subjects, programming is problem solving
intensive. One cannot come close to getting by just by knowing
answers. (p.226)

Furthermore, Perkins et al. (1989) indicate that learning to program with a degree of
competence in computer languages ‘poses a daunting challenge to many youngsters’
(p.261). In addition they indicate that programming calls for such a high degree of
precision that is not present in other subjects. For example, a student who spells 90% of
words correctly in a spelling test will achieve an ‘A’ grade. This is contrasted to a
computer program where 90% of the program statements are coded correctly. In this
case the program is likely not to do anything like what it is supposed to do. This is
compounded by the fact that:

The 10% shortfall in the program may introduce several interacting
factors that will make the debugging process exceedingly
demanding and frustrating, so that achieving a workable program
becomes a difficult task indeed. (p.155)

In his considerable experience of teaching programming Jenkins (2002) asserts that
‘few computing educators of any experience would argue that students find learning
to program easy’ (p.53). Bonar and Soloway (1983) indicate that ‘it is widely known
that programming even at a simple level is a difficult activity to learn’ (p.10). This
suggests that there is considerable agreement amongst programming academics and
researchers over a sustained period of time on the difficulty of programming as a subject for first year students. Furthermore, despite all of these problems, literature suggests that there doesn’t seem to be any universal solutions. Garner (2005) et al. highlight that ‘fundamental problems relating to program design are very persistent’ and ‘the average student does not make much progress in an introductory programming course’ (p.6). Linn and Dalbey (1989) highlight that it is ‘unreasonable to expect progress in general problem solving from a first course in programming’ (p.78) and that efforts to teach ‘complex’ problem solving skills to novices ‘have generally met with failure’ (p.78).

In terms of this difficulty, one encounters the notion of ‘fragile knowledge’ pertaining to learning computer programming as described in the seminal study by Perkins and Martin (1986):

> However, common experience testifies that often a person does not simply “know” or “not know” something. Rather, the person sort of knows, has fragments, can make some moves, has a notion, without being able to marshall enough knowledge with sufficient precision to carry a problem through to a clean solution. One might say that learners in such a state have fragile knowledge. (p.214)

In a later paper Perkins et al. (1988) refers to this ‘fragile knowledge’ as the first category of difficulty in learning computer programming for novices (this earlier study on ‘fragile knowledge’ will be described in detail subsequently in this chapter in the categorisation section). The second category of difficulty relates to ‘problem strategies and strategic shortfall’. Here they suggest that ‘many students of programming suffer from a shortfall in elementary problem-solving strategies’ (p.158). In particular, they fail to prompt themselves with the appropriate self-questions at the appropriate time e.g. ‘do I know a command that does that sort of thing?’. The third category of difficulty relates to ‘affect and attitude: problems of confidence and control’ (p.158) where deficiencies may be attributed to attitude and confidence. In this regard, they describe a student who disengaged from a task and
moved onto the next without attempting to understand what was wrong. When pressed for an answer, the student eventually offered a suggestion. The experimenter then urged the student to implement the suggestion and it subsequently worked, thus suggesting their disengagement was a result of lack of confidence in their perceived ability.

Fincher (2006) in a recent publication indicates that this difficulty experienced on novice programming courses can not only be exacerbated by the amount of material on a syllabus but also the interlinked nature of the concepts:

You should expect to go through some periods where the amount of new material seems overwhelming, and you appear to be being expected to learn it before you have had much chance to master earlier stuff. Sometimes this ties into the nature of computer programming as a unified skill. (p.46)

In terms of a student possessing requisite strategies to deal with programming, the notion of a programming plan/scheme emerges in the literature on learning programming. Samurcay (1989) defines a ‘plan’ as:

A catalogue of stereotypic action sequences that expert programmers have and select and use by adapting them to the need of the current situation. (p.163)

This clear and succinct definition illustrates that programming plans are something possessed by expert programmers and something that all novice programmers need to acquire to a certain degree if they are to progress to a satisfactory level in programming. Mayer (1989) agrees with this assertion and suggests that it may be possible to ‘teach the major “chunks” or “schemata” involved in computer programming’ (p.155) and indicates that explicit naming and teaching of basic schemata should become part of programming curricula.
2.4 TEACHING PROGRAMMING TO NOVICES

In the material already presented in this chapter one can see that there appears to be a consensus that given the idiosyncratic nature of the programming subject, it can be a difficult subject to learn. One might suggest that the corollary of this could be that it is a difficult subject to teach. In some of the literature already discussed, the authors refer to the outcomes of their research in terms of the implications they have on programming teaching/instruction. Perkins et al. (1989) highlight that one prevailing argument as to why programming is so hard to learn is because it is not well taught by instructors. However, they go on to state that they do not subscribe to this assertion given the fact that programming presents special problems that are not found in any other school subject. Some researchers e.g. Mayer (1989), Linn and Dalbey (1989) propose techniques for teaching that might improve learning. This section will look at some of these issues and recommended techniques.

Some researchers suggest that the type of teaching engaged upon has a significant impact on learning. According to Robins et al. (2003) teaching standards can influence the outcomes of student progress in programming. In this regard, Linn and Dalbey (1985) propose a ‘chain of cognitive accomplishments’ that should arise from ideal computer programming instruction. This chain starts with the features of the language being taught. The second link is design skills, including templates (schemas/plans), and the procedural skills of planning, testing and reformulating code. The third link is problem-solving skills, knowledge and strategies (including the use of the procedural skills) abstracted from the specific language taught that can be applied to new languages and situations. In a later paper, Linn and Dalbey (1989) indicate that ‘in order for more effective programming instruction to be developed, curriculum materials and teacher education programs are required’ (p.75). They go on to state that the main finding of their 1989 study was that instruction (i.e. teaching) strongly influences outcomes in introductory programming classes. They conclude that even the most able students learn significantly more in exemplary classes than in typical classes, where ‘exemplary’ refers to exemplary teaching in terms of the ideal chain of cognitive accomplishments. In this regard, they refer to exemplary teaching methods as those that focus on mastering program comprehension, reformulation of
existing programs into new ones that solve similar types of problems, design of programming solutions to new problems and acquisition of programming skills that are transferable to other programming languages.

Perkins et al. (1988) suggested the development of a ‘metacourse’ to enhance the learning of programming. This course comprised nine lessons to assist student in developing thinking skills and heuristics that were specifically designed to moderate the problem of fragile knowledge. In addition, they presented students with mental models of the computer, a systematic approach to breaking problems down into sub-problems and a framework for learning new commands with a large focus on drawing their attention to identifying re-emergent patterns in programming e.g. a counting pattern. The study entailed comparing the implementation of the metacourse with a series of experimental groups across four separate sites and comparing the results of programming task completion against those of control groups who were engaged in traditional programming instruction (i.e. one that uses a combination of lecture and laboratory work). The authors presented two important outcomes of this study. Firstly, whilst teachers found the metacourse teachable, they highlighted that they were more comfortable with presenting their own lessons. The authors attributed this to first time implementation of the course on behalf of the teacher which was likely to diminish as familiarity with the course increased. Secondly, in terms of impact on programming performance, the experimental group performed slightly better on each task with the largest difference relating to manual design (i.e. where the student writes out a manual solution on paper) of programming problems. The authors suggest that despite the fact that their study highlighted marginal improvements by using the metacourse it did suggest that developing ‘a more powerful pedagogy of programming is a possibility that deserves pursuit’ (p.175). In this regard any improvements, however minor they may be, are a move in the right direction. In addition, it appears that moving away from traditional means of instruction to ones that can improve problem solving strategies could result in incremental benefits as teaching methods are refined over time.

Mayer (1989) proposed a number of techniques such as ‘advanced organisers’ and ‘concrete models’ and ‘putting it in your own words’. The advanced organiser comprised a short expository prior to the course material, containing no specific
content to the material but providing general concepts and ideas that can be used to subsume key information in course material. Concrete models are developed either using ‘black box’ or ‘glass box approaches’. In the former, the student is presented with the model of the computer as a black box that takes input and data and outputs results. In the latter, the student is given a more elaborate model that comprises ‘input windows’ ‘output pad’, and ‘memory scorecard’. These terms like ‘pad’ and ‘scorecard’ are used to make analogies between the internal workings of the computer and items used in everyday life. Finally, he suggests a technique where getting students to put programming statements into their own words ‘can enhance the breadth of learning’ (p.147) and this can ‘encourage the activation of existing knowledge that is relevant for presenting the newly presented material’ (p.147).

Mayer (1989) contends that the primary reason for this type of approach lies in the fact that novices lack domain-specific knowledge. This is consistent with the approach of using metaphor or analogy to explain programming concepts to novices. Analogy consists of two key concepts i.e. the source and the target. The source represents a piece of knowledge that one is familiar with. The target refers to the less familiar piece of knowledge. When an analogy is made the features of the source are mapped onto the target (Blanchette and Dunbar, 2000). In this study conducted by Mayer (1989) experiments comparing performance of groups using these techniques against that of control groups were conducted with the experimental group exhibiting marginally improved results. The failure to achieve major improvements in this study can be linked to the assertions of Du Boulay (1989) who indicates that:

> Very often an analogy introduced at one point does not fit later on, so producing extra confusion in addition to any misapplication of the analogy at the point where it was appropriate. (p.298)

In this regard, Halasz and Moran (1982) predicted the failure of using analogy in computer science education:

> An analogical model is, by definition, a partial mapping to the computer system it is supposed to explain. No simple analogical model is sufficient to completely explain the operation a computer system. Computer systems are too different from familiar,
everyday non-computational systems. This forces the use of baroque or multiple analogical models. Furthermore, analogical models explain too much; there are always aspects of the analogical models that are irrelevant to the system being modelled. To make effective use of analogical models, the new user is faced with the confusing task of sorting out the relevant inferences from among the many possible irrelevant or incorrect inferences suggested by the analogy. (p.384)

Furthermore, Du Boualy (1989) suggests that association of programming language commands with English statements as suggested by Mayer (1989) may be risky:

They may derive from chance associations with the meaning of English words used in the programming or command language, from overgeneralisations from one part of the language to another, or from application of inappropriate analogies. (p.298)

In this regard, one can see that programming educators and researchers disagree over the potential usefulness of teaching techniques and methods. It seems plausible to suggest that because only marginal improvements emerged from these studies, no one method can be considered a panacea. This issue will be discussed in the summary.

In a related issue to the notion of a model of computer memory, Milne and Rowe (2002) suggest many problems are linked to student’s lack of understanding of what goes on in computer memory. They also state that ‘students will struggle in their understanding until they gain a clear mental model of how their program is working’ (p.63). This will require knowledge of how their program and its data is stored in memory and how the objects in memory relate to each other. They conclude by suggesting that the use of visualisation tools that enhance the programmers understanding of memory as the program executes might go in some way to alleviate these difficulties. An example of a visualisation environment is illustrated in fig 2.3. Future research can only determine the potential effectiveness of these visualisation tools on novice learning.
In summary, this section has presented an overview of studies implementing varying approaches to teaching programming. Whilst some of them did report marginal success rates it was noticeable that none provided a panacea. After concluding their significant literature review in the area of teaching and learning programming, Robins et al. (2003) concur with this viewpoint. In particular, they indicate that whilst many of these approaches show promise 'it is also the case that none of them has come to dominate the theory or practice of programming pedagogy' (p.161). It is also possible that marginal improvements in various studies may be examined in more detail and tested in various sites as to their generalisability e.g. at Irish third-level institutions. Finally, one issue put forward by Du Boulay (1989) is that in many cases, teachers have complete control over syllabi and teaching methods. In this regard, it may be possible to achieve incremental progress by subsuming best practice into teaching practice that is consistent with the notion of attaining exemplary teaching as demanded by Linn and Dalbey (1989). Perhaps this can only be achieved through cross-fertilisation of research in both teaching and learning.

2.5 QUALITATIVE STUDIES IN NOVICE PROGRAMMING

While much attention has been paid to the study of novices versus experts, it is clear that it is also useful to explore the topic of novices versus novices. A group of novices learning to program will typically contain a huge range of backgrounds, abilities, and levels of motivation, and also typically result in a huge range of unsuccessful to successful outcomes. (Robins et al. 2003, p.155)

This section examines the previous qualitative studies conducted in the area of novice computer programming. A cursory glance at these indicates that they have all been conducted within the last five years, with all but one being conducted in the last two years. This is an indication of the increasing popularity of qualitative methods in this research domain which has historically seen the vast majority of research projects being quantitative in nature. Of the studies uncovered in this literature survey, three were undertaken with the phenomenography method, one in the area of grounded
theory and one not reporting any formal qualitative research method. The discussion that follows looks at each of these studies in turn.

**Phenomenographic Study of Bruce et al. (2004)**

Bruce et al. (2004) conducted a similar study to this thesis study which looked at revealing differences in how first year programming students go about learning programming. This study used the phenomenography qualitative research method. It was conducted with 13 participants all of whom used the Java programming language in an Australian university. The nature of the open-ended questions proved useful in the author’s study, where questions such as ‘How do you go about learning to program?’ were utilised. Furthermore, recommendations about asking the background of students led to the question in this study which asked each participant how they got into computing in the first place (as highlighted in chapter 4).

Analysis of this study uncovers a number of similarities with the author’s study. In terms of motivation for the project, the authors clearly highlight that:

> Our study also opens the little explored territory of understanding learning to program from the perspective of university students undertaking introductory courses. This is an empirical study, which attempts to take the guesswork out of understanding students’ views. (p.145)

This assertion has considerable resonance with the study described in this thesis. One of the motivations for choosing grounded theory as a research method was that it allowed students to speak about their personal experiences in their own voices. Further analysis of this phenomenographic study shows conceptual overlap with this thesis study where one attempts to seek variations in meanings as well as the simultaneous contrasting of differences and clustering of similarities. In this regard, we see overlap with the methods of constant comparison and theoretical saturation in grounded theory.
The net outcome of this phenomenographic study was a set of categories that described the act of learning to program and secondly an outcome space that describes the relationships between the categories. This has conceptual overlap with the phenomena developed in axial coding and the category association diagram that shows the relationship between the central category and the various phenomena (described in detail in chapter 6).

The net outcome of this study is a set of five categories pertaining to the learning of programming by first year students (i.e. five different ways of going about learning programming. Firstly, there is ‘following’ where the student experiences programming as ‘getting through’ the unit. Secondly, ‘coding’ relates to where programming is simply experienced as learning to code. Thirdly, ‘understanding and integrating’ where learning to program is viewed as understanding and integrating concepts. Fourthly, ‘problem solving’ where learning to program is viewed as solving problems. Finally, ‘participating or enculturation’ where it is experienced as discovering what it means to become a programmer. Whilst these abstract categories do not appear to have much overlap with the abstract phenomena developed at a macro level in the author’s study, some of the micro-level issues uncovered in the form of lower level codes do overlap e.g. issues relating to ‘persistence’, ‘prior knowledge’ and ‘trial-and-error’ (p.149, Bruce et al., 2004). Finally, this study concludes by looking at the implications of the study for teaching practice and contends that an understanding of the theory emanating from the research is integral to future teaching practice. In this regard, the actual learning issues experienced by programming students may be at variance with those perceived by the educator. Thus, areas where these variations exist can be looked at for improvement in future teaching practice. An example of this variance described in chapter 6 is the amount of time it takes for programming concepts to sink in with students. Here it appears that this sink-in time is longer than that presumed by educators. In this regard, the lecturer may have moved to a new topic that is built upon a previous topic e.g. from one-dimensional arrays to two-dimensional arrays. Here, the student may not have been given sufficient time for the one-dimensional material to sink in (i.e. be accommodated) so that they can build upon it in terms of the more complicated two-dimensional array material. In terms of issues like this, the implications this study has for teaching practice will be discussed in chapter 7.
**Phenomenographic Study of Eckerdal and Berglund (2005)**

Eckerdal and Berglund (2005) describe a similar type of phenomenographic study conducted with 14 participants in a Swedish University. This study differs from the authors' study and that of Bruce et al. (2004) in that the students were taking an aquatic and environmental engineering course of which a programming module was compulsory. Another key difference in this study was the nature of the questions e.g.

1. **Q What do you think learning means on this course?**
2. **Q Why has this course been good for you?**
3. **Q What has been most important to you in the course?**
4. **Q What has been difficult in the course?**

In terms of a grounded theory study, some of the questions above could be construed as forcing where one asks the students about what has been good and what has been easy without first of all ascertaining whether the student has actually found the course difficult or easy. In a grounded theory study, one would ask the initial question of how they have found the course. If the answer to this question is ‘it has been difficult’, the researcher would follow up with questions relating to *what* they have found difficult and *why* they have found it difficult. Assuming an experience of difficulty by including it in a question without the participant first of all recounting it in an open-ended question could be construed in a grounded theory study as theory forcing.

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

Table 2.1 Learning to Program Categories (Eckerdal and Berglund, 2005)
In terms of similarity, the programming language used in this course was Java and the course was mandatory in the first year of their chosen course. Table 2.1 is taken from Eckerdal and Berglund (2005) and summarises the five main categories pertaining to ‘what it means to learn to program’ (p.137). The first category relates to understanding the programming language itself. Here, they isolate two issues, the first relates to learning to understand the syntax of the language and the second pertains to being able to write short pieces of programs or ‘program chunks’. The second category relates to learning a new way of thinking that is ‘difficult to catch’ (p.137). Here, the authors highlight how the participants articulated ‘the special thinking that is required’ (p.137) by programming where you have to try and think how the logic behind the language works. In the third category, participants try think of programming tasks in terms of metaphors or analogies in everyday life such as the steps involved in an ATM transaction where money is withdrawn using a specific set of steps e.g. enter pin, select cash amount, if there is enough money in the account etc. The fourth category entails the student seeing that problem solving is a crucial element of programming. In the final category, the student experiences programming as a skill that can be used outside of the programming and transferred to other domains that. In particular, it is seen as something that will provide them with a useful tool from which benefits may accrue in later life e.g. where it refines problem solving techniques useful in other domains. Analysis of this study highlights a progressive set of categories, i.e. those that have reached category level three have already experienced categories one and two and this is the same for the other two phenomenographic studies presented in this section. In addition, as well as overlap in progressive categorisation amongst these studies, there also seems to be considerable congruence in terms of emergent categories. This will be discussed further in the summary of this section.

Phenomenographic Study of Stamouli and Huggard (2005)
The most recent phenomenographic study conducted by Stamouli and Huggard (2006) is, to the best of the author’s knowledge the only other qualitative study conducted on novice programming in the Irish Republic. Here, 16 participants were interviewed on a longitudinal basis on four occasions. Some selection criteria were used based on students filling a questionnaire. These criteria took the form of variation in prior
programming experience, motivation in choosing their degree choice and previous academic performance. Selected participants were then invited for interview. As in the study presented in this thesis, the participants were first year computer science students studying the Java language. The interviews were semi-structured with a number of questions prepared in advance. The purpose of these predetermined questions was two-fold. Firstly, they were asked two open-ended questions about learning to program in general:

1. 'What do you think learning to program is all about?'
2. 'What do you think it takes to learn how to program?'.

The second segment of questions was more closed in nature with the aim of ascertaining students' understanding of 'program correctness':

1. 'When do you think a program is correct?'
2. 'What is your idea of a correct program?'

In terms of learning to program, the categories that emerged from this phenomenographic study are depicted in table 2.2. Analysis of this table indicates a progressive set of categories experienced by participants (as described earlier). For example, a student falling into the third category will have experienced the first three categories of learning experience.

This thesis study differs from the one conducted by Stamouli and Huggard (2006) in a number of ways. Firstly, this study was conducted across four separate sites, whilst the phenomenographic study was based in one site with 16 participants. Secondly, the phenomenographic study focused on both learning programming and program correctness and thus, predetermined questions relating to both were formulated and presented to participants (albeit the predetermined questions pertaining to learning were open and in a similar nature to those used in this thesis study). In a grounded theory study, the researcher would ask open-ended questions and allow an issue like 'program correctness' to emerge from the data before asking questions about it (and thus saturate it as a category). Finally, in the conclusions the authors describe a link between learning to program and the perception of programming correctness. Once
again, in a grounded theory study concluding with a link between two predetermined concepts might be construed as theory-forcing. Both approaches have obvious merit where the phenomenographic approach described here is useful where specific issues are identified in advance and required to be analysed using qualitative enquiry. In contrast, the grounded theory approach of this study is more suited to situations where the aim is to determine the main concepts of interest as they emerge from initial data collection and analysis and to subsequently focus on them using theoretical sampling.

<table>
<thead>
<tr>
<th>Category Label</th>
<th>Category Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Learning the syntax of the language</td>
<td>Learning to program is experienced as learning the syntax of the language.</td>
</tr>
<tr>
<td>2. Learning and understanding the programming constructs</td>
<td>Apart from learning the syntax of the language, the focus here also includes learning and understanding the constructs involved in programming in general.</td>
</tr>
<tr>
<td>3. Learning to write programs</td>
<td>As above, but also utilising all these to write programs by having a structure in them.</td>
</tr>
<tr>
<td>4. Learning a way of Thinking</td>
<td>In addition to the above, learning to program is also experienced as learning how to think logically in general.</td>
</tr>
<tr>
<td>5. Learning to ‘problem solve’</td>
<td>As above, and also utilising this way of thinking to solve programming problems.</td>
</tr>
<tr>
<td>6. Acquiring a new skill</td>
<td>The whole process is experienced as learning a new skill that affects the way one thinks in real life.</td>
</tr>
</tbody>
</table>

Table 2.2 Categories of Learning to Program (Stamouli and Huggard, 2006)

Qualitative Study of Carbone and Mitchell (2001)

Carbone and Mitchell (2001) present an Australian novice programming qualitative study which looked at the characteristics of certain programming exercises that lead to poor programming tendencies. Data were gathered from three sources namely student interviews, tutor feedback and in-depth cases written by students about their engagement in a task (similar to the diary/journal used in this study). Initially, a small group of students were interviewed on a weekly basis as they worked through their tasks. Later in the study, another group of students provided deeper insights into their learning experiences and documented their engagements in the tasks. In addition, tutors also reported on how students progressed on the tasks as they assisted them.
each week in the labs and tutorials. The net result of this study was the unearthing of three problematic areas. Firstly there is ‘non-retrieval’ (p.94) which occurs when no attempt is made to retrieve one's own existing views and understandings which are relevant to the knowledge being presented. The learner is unsure of conflicts between the school knowledge and their personal views. Secondly, ‘lack of internal reflective thinking’ (p.94) where the learner is not thinking reflectively about the subject content as presented. Each lesson, activity or even instruction is seen as isolated from each the others. The word ‘internal’ here means within the boundaries of the subject. Finally, they describe ‘lack of external reflective thinking’ (p.94) where the learner makes no attempt to link the content of the subject with the outside world or other subjects. As in the previous problem, the word ‘external’ relates to outside the boundaries of the subject. It should be noted that whilst the data collection methods reported in this study clearly indicate that it was qualitative in nature, no mention of the specific research method used to reduce the data into the above categories was presented.

**Grounded Theory Study of Fitzgerald et al. (2005)**

The only grounded theory study uncovered in the area of novice programming was that undertaken by Fitzgerald et al. (2005). The particular focus of this study was to identify strategies that novice students use to trace programming code. The authors of this study define ‘tracing’ as ‘the overall process of trying to emulate, at some level of detail and/or accuracy the process of executing code’ (p.71). One of the main differences between this study and the study presented in this thesis was that it was a task-oriented project where, during an interview session, participants were given a series of Java code segments and they were asked to ‘think aloud’ by talking their way through each of the code segments before committing to an answer on a multiple choice sheet. These think aloud sessions were recorded and the transcripts subsequently analysed to identify strategies that students use to trace code. Furthermore, 37 participants were interviewed across 12 sites, and as was the case in this thesis study, those students who volunteered to participate were invited to an interview. The analysis of the data followed a similar approach to this thesis study where ‘the researchers underlined and noted student phrases and words that indicated a strategy was being employed’ (p.71) and this is consistent with the approach of ‘micro-analysis’ used in this thesis study and will be described in chapter 3. One key
observation that emerged during this detailed transcript analysis was that different strategies were used for different questions and that many strategies were used successfully by some students and unsuccessfully by other students. The net output of this study was a list of 19 codes that were subsumed into two logical emergent theories namely, 'temporal groupings' and 'syntax, semantics and pragmatics' respectively. The former theory subsumes issues like familiarisation, recognition of data structures and walking through the code before they commit to an answer on the multiple choice sheet. The latter entailed identifying data structures, understanding the semantics (meaning) of new programming concepts and pattern recognition of re-emerging concepts in the code on a pragmatic basis. Analysis of the tracing strategy codes highlight overlap with a number of codes that emerged in this thesis study, e.g. 'guessing' and 'coming back to questions later' which will be identified in chapter 6.

In the author's experience, this type of task-oriented grounded theory study represents a departure from traditional grounded theory studies. Here, the researchers are looking for optimal variation in a highly focussed area of computer programming learning, namely code tracing. In this thesis study, this would have conceptual overlap with the 'micro-level' strategies that emerged in data collection and analysis (as will be described in chapter 6). However the main difference being that this thesis study was much more open-ended and not only looked at micro-level issues but also those of a macro-level. For example, as well as looking at high-level macro issues like accommodation and students' perception of programming it also analysed macro-level issues like seeking clarifying programming examples or implementing known bits of code at the beginning of a task in order to gain some initial confidence. Finally, despite representing a departure in the form of a task-oriented study, it illustrates the versatility of grounded theory where it can be applied to many different types of research question.

In summary, analysis of these various studies highlights overlap in terms if issues encountered. Whilst at a macro level one may not see these overlaps initially (whereby the abstracted phenomena developed in this thesis study may not appear to exhibit congruence with the phenomena in these reviewed studies), analysis of the data at the micro level does show overlapping concepts e.g. overlap between 'program chunks' (Eckerdal and Berglund (2005), p. 137) and 'known bits' [KB] a low-level (micro) strategy uncovered in this study and explained in chapter 6 and in the conglomeration
document of appendix 2. A simple reason for this may lie in the fact that high level categories are abstractions of the concepts found in the data. Given the constant element of subjectivity in qualitative research, these abstractions generated by one researcher may not describe the data in a way that would be done by another researcher. In cases like this, it is prudent to look at the underlying data that make up these higher level categories. In this regard, instances of overlap between micro-level concepts in these studies and the thesis study will be discussed in the relevant locations of chapter 6.

As mentioned above, the majority of qualitative research studies uncovered were conducted using the phenomenography method. Analysis of these three studies uncovers a number of issues. Firstly, there seems to be a very strong overlap in the actual categories developed by each study. In fact, one cannot help but notice the existence of categories pertaining to syntax, problem solving, learning a new way of thinking and acquiring a new life skill that can be used outside of programming across all three studies. This suggests that there seems to be a considerable number of concepts that are generalisable across programming populations on an international basis. This may be due to the fact that given the universality of programming languages e.g. Java they present similar types of issues to all novice students (given that the language and its problem solving demands stay constant irrespective of the spoken language or educational setting of the student). Furthermore, in the cases where the abstracted phenomena in the various studies had different labels, the researcher could analyse the lower level categories for actual overlap (this was the approach taken in this thesis where previous studies were trawled for the existence of categories that emerged in data analysis). Secondly, the approach of progressive sets of categories differs from the learner categorisation developed in this study (and will be presented in chapter 6). In particular, fig 2.1 illustrates the nature of category development in this thesis study. Here, one can see that a certain learner categorisation can reflect a complex network of interacting categories. In this diagram the shaded areas represent the subset of categories being experienced by the learner at a certain point on the learning continuum. This indicates that a learner can not only experience various permutations of sub-categories but may also experience varying dimensions of the properties of these categories e.g. they may experience a low level of the 'difficulty' category and a medium-high level of the 'enjoy' category (these categories
Chapter 2  

Literature Review

will be explained in detail in chapter 6). Here, one can see that a learner can experience fragments of the various high-level categories (which in grounded theory are referred to as phenomena). This is contrasted to the nature of the phenomenographic studies where one is presented with progressive categorisation as illustrated in fig 2.2. In this type of categorisation scheme a student who experiences category three for example, will also have experienced the first two categories (this is represented in tables 2.1 and 2.2 by the term ‘as above’). In this regard, one cannot determine if this is a fixed chronological ordering of experiences or even if a student can partially experience a category. For example, can they experience part of category one and part of category four and none of categories two and three? Bruce et al. (2004) who present this type of scheme as illustrated in fig 2.3 hint at this potential limitation where they indicate that:

While our categories of description are discrete, students themselves may adopt different ways of experiencing at different times during their learning experience. (p.154)

In this regard, it may be the case that the methods developed in this thesis study that represent a series of learning experiences on a continuum (in a continuous manner) with each point representing varying subsets of experiences denoted by categories, is a more intuitive means of category illustration.
Chapter 2 Literature Review

Fig 2.1 Categorisation Approach in Thesis Study

Fig 2.2 Progressive Categorisation in Phenomenographic Studies
2.6 CATEGORISATION STUDIES IN NOVICE PROGRAMMING

This section presents an overview of the previous studies conducted in the area of categorisation in novice programming. It will show that categorisation of learners and learner strategies have been conducted over the last twenty years and many of the issues uncovered are still as prevalent in today's educational environment despite the proliferation of modern teaching tools and techniques.

Perkins and Martin (1986)

This study was conducted on a task-oriented basis where 20 participating novice programmers were asked to complete a series of programming tasks during which they interacted with the experimenter. The data collected during these sessions were in the form of field notes taken by the researcher, code written by the student and an audio taped transcript of the interaction between the researcher and the participant. This data was accumulated to generate a verbatim protocol of the session. Subsequently case studies were drawn from the protocols and used in knowledge categorisation. The purpose of the study was to investigate student programming difficulties. In this regard, the researchers developed the study with the aim of revealing whether particular difficulties reflected high-level problem management skills or a poor understanding of particular commands in the programming language. The net outcome of this study was that programming knowledge is 'fragile' i.e. 'knowledge that is partial, hard to access and often misused' (p.213). Perkins and Martin (1986) go on to elaborate on four different dimensions of fragile knowledge namely, 'missing knowledge', 'inert knowledge', 'misplaced knowledge' and 'conglomerated knowledge'. Missing knowledge relates to a situation where student progress has halted because of knowledge that the student has either not retained or never learned in the first place. Inert knowledge refers to situations where the student fails to retrieve programming knowledge but in fact possess it, as revealed by a probe during the programming session. Misplaced knowledge describes a situation where the student uses program structures in a context where they do not belong. Conglomerated knowledge describes a situation where a student produces a piece of programming code that mashes together several disparate elements in a syntactically
or semantically anomalous way in an attempt to provide the computer with the information it needs. Perkins and Martin (1986) conclude their study by discussing the implications their findings have on the teaching of programming. Firstly they postulate that programming should be taught in a way that reduces where possible the different types of fragile knowledge. In this regard, they suggest that the instructor needs to convey exactly what each command does. As an example, they suggest that instruction on a programming ‘for loop’ should convey that this type of loop is not used in all looping applications but rather only in situations where there are a fixed number of iterations. Secondly, they recommend that educators teach programming in a way that preserves the exploratory use of a language. Here, students should be encouraged to experiment with a language, make conjectures, make mistakes and learn why there are mistakes and subsequently fix them. Finally, educators are recommended to encourage the use of elementary problem solving strategies for example constant use of ‘strategic questioning’ e.g. ‘what does this program need to do next?’ or ‘what will I have written really do?’ which are consistent with self-regulated learning strategies.

Perkins et al. (1989)
Perkins et al. (1989) conducted a study to look at the conditions of learning in novice programmers. In this study, the researcher sat with the participants as they attempted to complete a programming task whilst occasionally asking them to explain how and why they did certain tasks. The net output of this study was what the authors describe as a ‘stoppers’ and ‘movers’ continuum. They describe stoppers as students when ‘lacking a ready answer to the difficulty, the student not only feels at a complete loss, but is unwilling to explore the problem any further’ (p.266) and simply stops trying. On the other end of the spectrum, movers ‘consistently try one idea after another, writing or modifying their code, testing it, never stopping enough to appear stuck’ (p.266). In this categorisation they also describe ‘tinkerers’ who represent students who write some code and make small changes to it in the hope of getting it to work. They go on to indicate that ‘stoppers’ may not be necessarily inept nor movers especially able and in many cases their categorisation relates issues of confidence. They go on to indicate that such patterns of behaviour may relate to tacit models of learning and perceptions of self efficacy. The authors suggest that many of the
problems experienced by ‘stoppers’ could be alleviated by proper instruction. In particular, they suggest that instruction could put emphasis on certain practices, i.e. being a ‘mover’ rather than a ‘stopper’, tracking closely the programs they write and breaking programs down into more manageable chunks.


Jenkins and Davy (2000), describe a categorisation project which was used to separate a class group of students for the purposes of diverting them into a stream where specific teaching methods could be devised to address their level of proficiency. This categorisation is as follows:

- **Rocket Scientists** – who are already highly proficient programmers, and who would learn little new from the module.
- **Strugglers** – students who will find the course challenging, but who would be expected to pass reasonably well.
- **Serious Strugglers** – students who will find the course extremely difficult, and who will not pass without significant additional support and encouragement.
- **Averages** – those who remain; they will pass the module well, and will need only an occasional word of advice or help with debugging.

This classification was based on three data collection activities, namely informal interview, a formal test and a background information assessment. The formal test assessed the understanding of variables and loops, mathematical expressions, programming syntax and sequences. In terms of this task-oriented approach to categorisation Jenkins and Davy (2000) highlight the following:

‘Using such a test was not without risk. Even though it was constantly emphasised that there was no expectation that every student would be able to answer every question well, there was concern that failure to do well would have a strong de-motivating effect on some students. Happily, there was no evidence that this was the case. (p.3)
Whilst the assertion that no de-motivation took place may be in fact true, engagement in the grounded theory study of this thesis suggests that this could only be accurately obtained through follow-up interviews with students. As this study will present in chapter 6, any one of a number of factors such as exposure to an ineffective example may cause the student to disengage from learning programming. In fact, this situation is even more tenuous if one takes on board the notion of the fragility of programming knowledge as referred to by Perkins and Martin (1986) in a previous section of this chapter.

**Dehnadi and Bornat (2006)**

Some researchers propose a simpler type of categorisation such as Dehnadi and Bornat (2006a). In this quantitative study, students without any prior programming knowledge are given a Java programming test consisting of rudimentary assignment statements. As a result of this test, the authors identify three categories of student:

- **Consistent** - students use the same mental model to answer almost all of the questions
- **Inconsistent** – students use different mental models for different questions
- **Blank** – students refused to answer almost all of the questions

On completion of this test the authors correlate it with the final results the students attained in their programming subject. The net outcome of this was in the authors' opinion a clear separation of populations. The consistent group attained significantly higher marks than their inconsistent/blank counterparts. The results at this stage indicate that the majority of inconsistent students do not perform well, whilst the majority of consistent students perform well, with a significant number performing well in the 70-100 grade category. Dehnadi and Bornat (2006b) go on to describe the various mental models the students participants actually used in the tests. The net outcome of this research is the assertion that the probable success of a given student can be predicted in the first week in an aptitude assessment type manner. Robins et al. (2003) also lend weight to this argument of early prediction but differ to the extent that they describe a study which indicates that the success prediction can come from
within the students themselves in terms of their own expectations having taken a programming course for two weeks:

Rountree et al. (2002) found that from a survey (covering factors such as background, intended major, expected workload and so on) of students in an introductory programming paper, the most reliable predictor of success was the grade that the student expected to achieve. This and other results showed that students in general have a reasonably accurate sense of how they are likely to do within the first 2 weeks of the course. (p.155)

The net outcome of this study is that the authors take an extreme position on novice programmer categorisation. In particular, there are two simple categories, those who can (consistent) and those who cannot (inconsistent) program. Furthermore, they assert that no approach to teaching will make inconsistent students consistent and that those who do not possess natural aptitude will not succeed as programmers. It should be noted that this study by Dehnadi and Bornat (2006) is in the process of being implemented with larger populations and this may serve to assess the generalisability of the initial studies which are still unknown.

Robins et al. (2003)
Robins et al. (2003) in their comprehensive literature review on teaching and learning computer programming also identify two categories of novice programmer, notably effective and ineffective. They contend that effective novices are those that learn, without excessive effort or assistance. Ineffective novices on the other hand, are those that do not learn, or do so only after inordinate effort and personal attention. They then suggest that it may be productive to explicitly focus on trying to create and foster effective novices. In light of this they contend that 'rather than focusing exclusively on the difficult end product of programming knowledge, it may be useful to focus at least in part on the enabling step of functioning as an effective novice’ (p.165). They go on to highlight that strategies will vary along the effective-ineffective continuum:
Differences in initial strategies will interact with other factors, such as motivation and the capacity to acquire language related knowledge, to rapidly separate novices along the effective-ineffective continuum. (p.166)

This categorisation emanates from a detailed and elaborate review of relevant literature and the considerable teaching experience of the authors. They suggest that future work in the area of research into the area of teaching computer programming might aim to try and gain an understanding of what strategies are employed by effective novices, as well as understanding their mental models or schemes as they program. If these can be ascertained, subsequent research might look into whether or not these mental models or schemes can actually be taught to ineffective novices.

Naps et al. (2003)
The approach taken in the previous example of using literature to develop categorisation was also taken by Naps et al. (2003) who look at the area of program visualisation and visualisation technology. This project was undertaken by a large working group. They describe visualisation as a technique used to graphically illustrate various concepts in computer science. Using this approach programming code can be written in the normal way and afterwards, graphical illustrations, very often in the form of animation (as in the Jeliot\(^3\) tool illustrated in fig 2.3), are used to simulate in a step-by-step manner how the program works in terms of variable initialisations and updates, arithmetic expressions, loop iterations etc. For example, it may show how each line of code is executed in sequence or how an array data structure is populated with data elements e.g. an array containing seven locations each holding the average temperature for each day in a week using a ‘for’ loop. In this working group project the authors propose a taxonomy relating to different forms of ‘learner engagement’ in terms of utilization of visualisation technology:

\(^3\) Interested readers should access http://cs.joensuu.fi/jeliot/description.php to see a detailed description of this tool.
• No viewing
• Viewing,
• Responding
• Changing,
• Constructing
• Presenting.

In the first category, ‘no viewing’ the authors assert that since it is possible to understand a piece of programming code without the use of a visualization tool it must be included as a viable category (even when the visualization tool is at their disposal). The second category, ‘viewing’, including step-by-step execution and multiple views, is a basic requirement for all the higher levels where the student may control the pace of the animation/simulation or rewind to re-simulate the code again. In the third category, ‘responding’ the student answers questions relating to the visualization, e.g. ‘what is the value stored at index position 3 in the temperature array’. In the fourth category ‘changing’ the student is given the opportunity to change the behaviour of the program, e.g. with different input. This can also be combined to the responding task, e.g. asking for an input to produce a certain output or to cover all possible execution paths. In effect, this facilitates learning by implementing a what-if scenario e.g. ‘what happens if I set the value of this numeric variable to zero?’. The fifth category, ‘constructing’ relates to where the student is provided with mechanisms to construct their own visualizations. Finally, the sixth category, ‘presenting’ entails a student presenting their visualizations to an audience e.g. their peers for subsequent feedback and discussion. In addition, the authors stress the fact that this set of categories is not an ordinal scale, but rather there are ‘overlapping possibilities the last five of these engagement categories’ (p.142) and they go on to illustrate this in the form of a ‘Venn diagram’. The net outcome of this study is a set of hypotheses that can be tested on student programmers e.g. ‘viewing vs. no viewing’ or ‘constructing vs. changing’. In terms of each hypothesis the authors describe examples of each one that can be tested on student groups.
In summary, this section has described previous studies in the area of novice programmer categorisation. At the highest level of abstraction one can see two approaches to categorisation. Firstly there is the notion of a binary categorisation, which in its crudest sense identifies two simple categories i.e. those who can and those who can’t. This extreme position is taken by Dehnadi and Bornat (2006a) who indicate that only those who fall into the ‘consistent’ category should proceed in engaging in a programming career and those who don’t should not. Secondly, there are less extreme representations of this type of categorisation where Perkins et al. (1989) and Robins et al. (2003) represent a continuum between the categories of ‘stoppers’ versus ‘movers’ and ‘effective’ versus ‘ineffective’ respectively. Other authors (Naps et al., 2003; Jenkins and Davy, 2000) identify a number of discernable categorisation points on this continuum. These studies also suggest that it is quite difficult to identify formal categories with the boundaries between categories often blurred or fuzzy. In addition, the literature suggests that in many cases, the categories may actually overlap, sometimes in a complicated mesh-like fashion. Furthermore, it
appears likely that a programming student may encounter different types of learner categorisation over a period of time. Chapter 6 will present the approach to categorisation in this study and will make particular reference to where it overlaps with the studies presented in this section.

2.7 MAJOR ISSUES AND CHALLENGES

This literature review has raised a number of issues that are of interest to this thesis study. As regards learning, many commentators are of the opinion that programming, with its unique demands of exactness in terms of syntax and semantics presents a difficult challenge for novice students. This difficulty is exacerbated by the fact that an ability to problem solve is thrown into what is already a complicated mix. Given these unique demands, programming is considered a more different and difficult undertaking than other subjects computing students encounter in their first year. Perkins (1986) has described this situation as resulting in first year programming knowledge being ‘fragile’ and as such needs to be dealt with teaching strategies that take cognisance of this fragility. In addition to this, rather than being a sequence of discrete non-related topics, Fincher (2006) has indicated that programming is a continuous series of ‘interlinked’ topics that this author refers to as a ‘building block’ approach. In this regard, if the foundation-level knowledge is weak then all subsequent programming learning can suffer as a consequence. In addition, we are told that in order to come to terms with syntax, semantics and problem solving the novice student must develop programming ‘plans’ or ‘schemes’ which are action sequences that can be called upon to solve or complete programming tasks. In addition, many commentators highlight that given this uniqueness, specific teaching methods need to be devised to deal with fragility and the demands of problem solving as well as programming plan acquisition. However given all of this, no major success has been uncovered with any teaching method to date. Here, Perkins (1986) makes particular reference to lack of problem solving strategies and the impact confidence and attitude has on student performance.
Chapter 2

The majority of qualitative studies uncovered utilised the phenomenography research method. These studies appear to have a number of conceptually identical categories in terms of what it means to learn programming (surprisingly all more or less use the same terminology to describe these categories). These relate to ‘learning to code’, ‘learning a new way of thinking’ and ‘learning to problem solve’. It will be interesting to see how these overlap (if at all with the outcomes of this thesis study). Whilst these studies describe what it means to learn to program at a higher macro-level, Fitzgerald et al. (2005) illustrate that detailed data collection and analysis can result in uncovering micro-level activities that represent low-level strategies employed by students such as guessing or beginning with known chunks of code.

In terms of categorisation of novice programming learners some commentators suggest that there are essentially two types namely those who can and those who can’t (e.g. consistent vs. inconsistent, or effective vs. ineffective). In representing the extreme viewpoint, Dehnadi and Borat (2006b) indicate that these are discrete categories and no approach to teaching will actually change this situation. Other commentators such as Jenkins and Davy (2000) and Robins et al. (2003) suggest filtering these students into different classes that use teaching approaches to address deficiencies pertaining to a category. Other researchers such as Perkins et al. (1989) and Naps et al. (2003) put forward the notion of a learning continuum where a given learner may experience many different points on the continuum depending on exposure to adequate/inadequate teaching methods.

In summary, this section has presented a number of issues and challenges to the novice programming researcher. The subsequent chapters in this thesis will illustrate how these have or have not manifested in this thesis study as well as presenting the new knowledge uncovered in this research.
Chapter 3

Methodology

- Introduction
- Grounded Theory Overview
- Suitability of Grounded Theory
- Underlying Issues in Grounded Theory
- Theory Forcing and The Paradigm Model
- The Research Question
- The Resultant Theory
- Methodological Issues Encountered
- Conclusion

3.1 INTRODUCTION

This chapter describes the grounded theory research method used in this study. To this end, it presents to the reader a number of issues. Firstly, it gives a detailed description of the Strauss and Corbin (1998) variant of grounded theory actually utilised in the project. This is followed by a detailed discussion of why this variant was chosen and a discussion of its suitability to the research question. In addition methodological issues
encountered in the data collection and analysis phases are described in detail with reference to how this experience tallies with experience of other grounded theory researchers in the literature.

3.2 SUITABILITY OF GROUNDED THEORY

It seems logical that the selection of any research method should be based on the nature of the research question (Brown et al. 2002). Smith and Biley (1997) ask the obvious question of any research method, ‘does the grounded theory method of investigation lend itself to the phenomena of interest?’ (p.25). According to Pidgeon and Henwood (1996) grounded theory is ‘particularly suited to the study of local interactions and meanings as related to the social context in which they actually occur’ (p.75). As mentioned earlier in this thesis, the majority of research conducted to date in novice computer programming in Irish institutions has been of a quantitative nature. In this regard, it may happen that quantitative research conducted to date may not explore, in a profound way the individual issues experienced by students in their particular educational setting. In determining the suitability of grounded theory to address this situation, Pidgeon (1996) postulates that ‘grounded theory places great emphasis upon the attention to participants’ own accounts of social and psychological events and of their associated local phenomenal and social worlds’ (p.76). Douglas (2003) affirms this view by indicating that ‘rather than commencing with a theory that he or she attempts to verify, commences with an area of study that allows relevant theoretical conceptual constructs to emerge’ (p.47). This in may ways reflects the status of Irish novice computer programming research, an area that is not short of valuable research (albeit mostly of a quantitative nature), yet to a large extent, devoid of context-specific substantive theory and constructs that is based on in-depth interviews with students (where they are given an opportunity to recount their experiences in an open manner). Given this lack of theory Smith and Biley (1997) assert that ‘grounded theory is particularly useful when little is known about the phenomenon under study’ (p.29). Goulding (1998) reports that grounded theory is particularly useful in studying the nature of ‘experiences which are not easily quantified’ (p.56). Brown et al. (2002) inform the prospective educational
researcher that it can be ‘an effective tool in conceptualising complex phenomena, providing language to describe it, detailing how it occurs, and ultimately, student affairs educators’ contributions to this process’ (p.10). Goulding (2002) essentially encapsulates many of the reasons why grounded theory has been applied to the specific research question:

Usually researchers adopt grounded theory when the topic of interest has been relatively ignored in the literature or has been given only superficial attention. Consequently, the researcher’s mission is to build his/her own theory from the ground. However most researchers will have their own disciplinary background which will provide a perspective from which to investigate the problem. (p.55)

In terms of this assertion by Goulding (2002), given the nature of the research question outlined in this research project, grounded theory was found to be a particularly suitable method in addressing the learning experiences of Irish novice programmers, an area of study where the author could not find any previously documented qualitative research. Furthermore, from the perspective of disciplinary background, the author has worked for 11 years in the area of novice programming education and given his academic background in computer science, is familiar with abstraction, an approach that permeates the computer science discipline and an essential requirement in theory building.

Furthermore, in terms of the participants themselves, applying grounded theory to this type of research context enables them to describe their experiences and problems in their own voices and this can allow the researcher to ‘discover the issues that are important or problematic in the respondents’ lives’ (Strauss and Corbin, 1998, p.38). Douglas (2003) supports this type of research approach where ‘theory emerges from the researcher grappling with not only his/her own analytical perceptions, but from empathising the ways in which the respondents themselves construct their reality, their world’ (p.53). Research uncovered in the literature review indicates that most research in novice programming is conducted by researchers who have experience in teaching programming. In this regard, they can often empathise with the problems
described by participants as the will have experienced their own students witnessing similar problems in both laboratory and classroom environments. This prior awareness and empathy can be useful in asking pertinent ‘how’ and ‘why’ questions of the participants. Data collection conducted during this project reaffirmed the suitability of grounded theory to the context of the study in terms of the way it allows participants to describe their own personal experiences in their own voices (Strauss and Corbin, 1998). Borgatti (2004) describes this as being concerned with emic perspectives of the world where categories are drawn from participants descriptions with the aim of what Borgatti (2004) describes as ‘making implicit belief systems explicit’ (p.1). In fact, this emic perspective is actually facilitated in grounded theory analysis where the term in-vivo is ascribed to a category that is taken verbatim from the data e.g. if the data contains a sentence such as ‘it takes ages for new programming concepts to sink in‘ then ‘sink in’ is an example of an in-vivo category. This study has found that not only does grounded theory allow participants to recount their experiences but it also enables the analyst to gain in-depth insights into how they deal with those experiences. This is consistent with the assertion of Goulding (1998), where she indicates that ‘grounded theory as a methodology was developed for, and is particularly suited to, the study of behaviour’ (p.57). Finally, the reasons for selecting grounded theory as the chosen research method for this study is aptly summed up by Allan (2003) where he indicates that grounded theory ‘investigates actualities in the real world’ (p.7). Brown et al. (2002) indicate that grounded theory resonates with both the people who experience the phenomenon (in this case the students) and those educators who have a professional interest in it. From the participants’ perspective, Pidgeon (1996) tells us that it places great focus on their ‘own accounts of social and psychological events and of their associated local phenomenal social worlds’ (p.76). These points illuminate in a clear fashion the rationale behind the choice of grounded theory in this research project, which is, if we want to know about Irish novice students’ experience in computer programming, let us ask them and use what they tell us as the actualities in the real world. Pidgeon (1996) refers to these actualities as ‘common-sense realities’ (p.77) of the participant.

In summary, this section has discussed the suitability of grounded theory to the research question. It will be shown in subsequent chapters in this thesis that grounded theory not only enables the researcher to classify the experiences of students engaged
Chapter 3 Methodology

in novice computer programming, but it also sheds light on how students actually deal with these experiences.

3.3 GROUNDED THEORY OVERVIEW

Grounded theory is an inductive qualitative research method. With this approach rather than starting with a hypothesis and testing it, the grounded theory researcher begins by collecting data in the field and lets the theory emerge or emanate from the data (Strauss and Corbin, 1998; Goulding, 2002). In this regard, it is postulated that the theory is actually grounded in the data. Data is usually in the form of interview transcripts or observational notes. Research subjects are chosen using theoretical sampling a sampling mechanism that is based on their potential for contribution to the development of theory.

Conducting grounded theory research entails a number of levels of coding and analysis. Open coding examines the text for items of interest, with the ultimate aim of accumulating codes into categories. Here, the researcher uses the constant comparative approach where they constantly compare new instances of the category with those already encountered until he/she saturates the category. Theoretical saturation is reached when no new insights into the category in terms of its properties or its relationships with other categories can be gained from gathering more data (Locke, 2001). Axial coding entails relating categories to their sub-categories around the axis of a central category, based on linkages between their properties. Selective coding entails identifying a central phenomenon and relating central categories to it using statements of relationships. Very often, in selective coding, a ‘storyline’ is generated that narrates the categories and their relationships (Strauss and Corbin, 1998).

The net outcome of grounded theory research is a theory that contains a central phenomenon, its causal conditions, its intervening conditions and its consequences (Creswell, 1998). According to Charmaz (1990) the strategies of grounded theory ‘can yield rich data, elaborated categories, and dense analyses with applications
Chapter 3 Methodology

across substantive fields' (p.1163). Given these potential outcomes, grounded theory was considered to be a suitable methodology in terms of addressing the research question. This suitability will be further discussed in various sections throughout this chapter.

3.3.1 Open Coding

The technique used in the open coding process was that of microanalysis as described by Strauss and Corbin (1998). Here the researcher analyses the interview transcripts on a line-by-line basis and inserts initial codes into the margins of the text (examples of codes might be ‘difficulty’, ‘enjoyment’ ‘pressure’). Strauss and Corbin (1998) point out the researcher must analyse the data as opposed to reading it in a general way. In this research it was found that the researcher must engage in a very concentrated analytical mode when they conduct this process, and the derived codes must reflect the data-intensive process that necessitates multiple passes through the interview transcripts. Pidgeon and Henwood (1996) describe this as a tentative process where the researcher labels concepts in the text that he/she ‘considers to be of potential relevance to the problem being studied’ (p.92). They go on to highlight an important issue that ‘judgement is always involved in this labelling process’ (p.92). In this regard, the researcher must possess theoretical sensitivity i.e. they must be sensitive to the meanings embedded in the data within the problem/phenomenon domain. This sensitivity may entail the researcher using their judgment when analyzing the data. Douglas (2003) addresses the judgement issue from the stance that in order to do proper analysis the researcher must suspend judgment on possible outcomes. Rennie (1998) and Thorne (2000) document the use of bracketing, a technique originally used in phenomenology where the researcher attempts to suspend or set aside their prior knowledge, attitudes and beliefs about the phenomenon under study. Backman (1999) warns that ‘this detachment, may, however, be quite difficult for the novice researcher’ (p.148). This was found to be the case in this research where on hearing a certain issue being raised by a student e.g. ‘difficulty with parameter passing’ the educator has to refrain from reverting to their own professional experiences of this topic and listen astutely to the student’s actual experience, rather than documenting their own slant on it. In this regard, the
experience of this gained in interviewing during the pilot study proved to be invaluable. Furthermore, according to Strauss and Corbin (1998) theoretical sensitivity cannot happen on a haphazard basis, but rather happens to 'prepared minds during interplay with data' (p.47). The pilot study conducted revealed this to be a slow, laborious process and the time taken to do it cannot be underestimated. In practical terms, open coding in this manner uncovers many codes, some of which may overlap. As a result of this issue, this research has found that the researcher must conduct a fine balancing act between eliminating their bias and possessing the required level of sensitivity to the data they are collecting. The potential danger here is that if the researcher fails in this aforementioned fine balancing act, the emergent theory may be based on assumptions, bias or speculation. Furthermore, on the other end of the spectrum, there is also a danger of the researcher being over-sensitive where they have such a profound knowledge of the nuances of the subject material, in this case computer programming, that they may gloss over the subjects' responses and present their own. Deep understanding of the subject material may cause the researcher to disengage analytically as soon as they hear a certain term being used by the respondent and fail to gain an insight into how and why a student internalises a certain concept. This issue is addressed by Backman (1999) who states that in some cases 'theory is not generated exclusively based on actual data, but it may also be based on previous knowledge about the research phenomenon (p.150). This situation was guarded against in two ways during this research. Firstly, a detailed analysis of the literature was not engaged upon until analysis had begun. Secondly, interviews were taped so that transcription was based on actual data and not on field notes.

Open coding entails the isolation of codes that were contained in the data, abstracting them into categories and analyzing them in terms of their properties and dimensions. Once again this is a labour-intensive process that necessitates multiple passes through the interview transcripts (Pidgeon and Henwood, 1996). Category development is aided by using the constant comparative analysis method. Here, the researcher compares new instances of a category and compares it to the existing categories with the aim of ascertaining what is the same and what is different. This term was derived because it reflects how the researcher compares new instances in the data against existing ones on a constant basis until saturation is reached. Open coding is also aided by the use of theoretical memos where the researcher looks at the codes/categories
they have developed in a reflective mode and documents their thoughts and interpretations. This process aids category development and further data collection and is described in the following section.

3.3.2 Theoretical Sampling

Strauss and Corbin (1998) describe the purpose of theoretical sampling as one whose purpose is to go to people, places or events that will maximise the researcher’s opportunities to uncover variations amongst concepts with the ultimate aim of densifying categories in terms of their properties and dimensions. Charmaz (1990) differentiates this from other types of sampling (e.g. purposeful sampling) in the sense that subsequent data collection is shaped by emerging theoretical categories in the field. Charmaz (1990) goes on to indicate that the purpose of theoretical sampling is to collect new data in a way that checks, fills out and extends theoretical categories. Purposeful sampling selects information rich cases for in-depth study and therefore can be considered selective (Mugo, 2005). Theoretical sampling, in its purest sense, pertains to sampling that is driven by emerging theory as described by Charmaz (1990). Becker (1993) highlights this purist standpoint by indicating that it is an 'ongoing process of data collection that is determined by emerging theory and therefore cannot be pre-determined' (p.256). However, Strauss and Corbin (1998) draw the researchers attention to pragmatism in terms of sampling and indicate to the researcher the practical constraints they may encounter in the field, such as access, available resources (in the case of this research, an resource example was an available room that was suitable in terms of absence of noise and suitability in terms of confidentiality), and time limitations (in this research, time limitations in terms of limiting class time missed by participants were encountered). In this regard, Strauss and Corbin (1998) advise that the researcher should never become upset by not being able to choose a site or obtain access to theoretically rich sites or persons. Instead they recommend that the researcher should ‘make the most of what is available to him or her’ (p.210). In the case of this research, this proved to be valuable advice in the sense that given the open nature of the research question, the experience of any participant engaged in a first year programming course was theoretically relevant to the research question irrespective of whether it further saturated a category or provided an example
of a negative case, i.e. data provided by a participant that was at odds with emergent theory. For example, if the emergent theory was suggesting a strong link between enjoying programming and performing well at it, a negative case would be uncovering a student who didn’t enjoy programming but who was achieving a very high mark in assessments. Here, negative cases have to be investigated to see if they are isolated incidents or a new dimension of an emerging category.

3.3.3 Theoretical Memoing

Grounded theory researchers are encouraged to use memos during the open coding process. Memos are written notes taken by the researcher during coding and may vary in form from written text to diagrams. Essentially they allow the researcher to document the theoretical insights that occur to them whilst analysing the data. Memoing of this nature is useful on many levels especially to direct the nature of future data collection (where it may highlight new issues to be investigated) as well as simply ensuring that important issues that emerged in the context of data collection and analysis are documented immediately lest they are forgotten in time. Miles and Huberman (1994) highlight the importance of memoing when they state:

> Always give priority to meoming. When an idea strikes, stop whatever else you are doing and write the memo. Your audience is yourself. Get it down. Don’t worry about prose, elegance or even grammar. Include your musings of all sorts even the fuzzy and foggy ones. Give yourself the freedom to think. Don’t self censor. (p.74)

In the pilot study the utilisation of memos was seen as an integral part of the analysis process and subsequently this experience proved to be invaluable in the full study. Furthermore, the author was guided by the advice of Strauss and Corbin (1998) who indicated that memoing should be done after each round of data collection. They indicate that the memos need not be long, but rather, capture an important issue at a given point in time. In this study, memoing was used intensively during open coding where the author’s illuminative thoughts were documented at the actual point in
Chapter 3 Methodology

which they occurred. This was seen as an essential part of the process as documenting them at a later date might have had the effect of losing the exact details of the thought and the context in which it was originally formulated. An example of a memo generated as part of open coding in the pilot study is represented as follows:

| Code - Persistence | This seems to me to be a recurring theme from emanating from all participants. When they come across a difficulty their instinct is to, as they say themselves, 'go over it again and again' or 'keep at it'. Also the participants seem to be stating that each successive attempt at the issue makes it a little easier and intimate that over a period of time the concept 'became easier'. This is a solution strategy that seems to be engrained within the participants for some reason. Whilst they take time to answer some of the questions posed, their answer to the question of what they do when they encounter a difficult concept is immediate and uniform in terms of persisting at it until it becomes clearer. |

Charmaz (1990) describes memo writing as a tool for engaging in an extended dialogue with oneself and allows the exploration of what is implicit and explicit in the data. In this research, memo writing was very useful as both a category development and reflective process. In particular, reflecting on the data and emerging categories often provided new insights that were documented at the time and explored in subsequent data collection, which is in harmony with the requirement of theoretical sampling.

3.3.4 Axial Coding
Axial coding is a process of relating categories to their subcategories and is given the term 'axial' because coding takes place around the axis of a particular category (Strauss and Corbin 1998). This enables the researcher to relate categories to each other. Walker and Myrick (2006) indicate that the purpose here is to extricate relationships on which the axis of the category is being focused. According to Goulding (2002), axial coding entails moving to a higher level of abstraction by
Chapter 3 Methodology

identifying relationships between categories and identification of a core category around which other concepts revolve and these higher level categories (phenomena) form the basis for the construction of theory. According to Spiggle (1994):

Abstract concepts encompass a number of more concrete instances found in the data. The theoretical significance of a concept springs from its relationships to other concepts or its connection to a broader gestalt of an individual’s experience’. (p.494)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description (Memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenomenon</td>
<td></td>
</tr>
<tr>
<td>Causal Conditions</td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td></td>
</tr>
<tr>
<td>Intervening Conditions</td>
<td></td>
</tr>
<tr>
<td>Action/Interaction Strategies</td>
<td></td>
</tr>
<tr>
<td>Outcome/Consequences</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1 Paradigm Model

To aid this process the coding paradigm developed by Strauss and Corbin (1998) is used. This paradigm enables the researcher to analyse a phenomenon (i.e. higher level category) from a number of perspectives in terms of its context and its relationship with its sub-categories. Once again the use of memoing is an essential part of this process. Borgatti (2004) advises the researcher to generate a frame (see table 3.1) comprising the various elements of the paradigm model and associated descriptions in the form of memos.

3.3.5 Selective Coding

Selective coding is described as the process of integrating and refining theory (Strauss and Corbin 1998). Essentially, categories developed during open and axial coding represent descriptions of data and do not yet form a theory (Goede and Villiers 2003). To this end, various categories need to be integrated to form a theory. This can be achieved by analysing the higher-order categories and subsequently deciding on a central category that is representative of the theme of the research. According to Strauss and Corbin (1998) ‘a central category has analytic power … what gives it that
power is its ability to pull the other categories together to form an explanatory whole' (p146). As an example of a grounded theory on teenage drug taking they present the central category ‘Teen Drug Taking: A Phase of Experimentation’ (p.146). They stress that a central category represents the analyst’s interpretation of what the data is all about and represents the salient issues in the lives of the participants.

The second part of selective coding entails the generation of a ‘storyline’ (Strauss and Corbin 1998) that refines and illustrates the theory. According to Brown et al (2002), the story should be told at a conceptual level, relating sub-categories to the central or core category. According to Borgatti (2004), ‘selective coding is about finding the driver that impels the story forward’ (p.3). In simple terms, the researcher identifies a storyline and writes a story that integrates the categories identified in the axial coding model (Creswell, 1998).

3.3.6 Category Development

Theory building in grounded theory is essentially focussed around the notion of a category. Here, the grounded theory researcher is concerned with category identification and classification. Strauss and Corbin (1998) describe the purpose of this grouping or classification. :

The purpose behind naming phenomena is to enable researchers to group similar events, happenings and objects under a common heading or classification. Although events or happenings might be discrete elements, the fact that they share common characteristics or related meanings enables them to be grouped. (p.103)
Theory building in grounded theory is focused around the notion of a *category*. Here, the grounded theory researcher is concerned with category identification and classification. In simple terms, the researcher analyses the relevant narrative for the existence of codes. On completion of this, the codes are analysed and those that conform to a common theme are grouped together. This higher order commonality is referred to as a concept (Allan, 2003). Concepts are then grouped into areas of commonality to form the highest-level abstract notion, namely that of a category (as illustrated in fig 3.1). Furthermore, in relation to category abstraction, the notion of the ‘subcategory’ emerges. Strauss and Corbin (1998) define a subcategory as ‘concepts that pertain to a category giving it further classification and specification’ (p.101). It should be noted here that the direction of the arrows in fig 3.1 illustrate the bottom-up nature of this process.
As well as the classification of identified categories, grounded theorists are also concerned with the properties and dimensions associated with each category. Properties represent the attributes or characteristics of a category, whilst dimensions refer the range of variability of a given property. The notion of behaviour is also important in category development. Strauss and Corbin (1998) draw the researchers attention to this by indicating that ‘what is less apparent when we classify objects is that a classification implies either explicitly or implicitly, action is taken with regard to the classified object...a flight consists of taking off and landing’ (p104). In this regard, they are articulating the fact that whilst properties e.g. height and weight and their dimensions short/tall or light/heavy may be easy to identify in the data, the sequences of actions, contextual conditions and action strategies that affect these dimension values may not be as explicit in the data. Therefore, they must be extracted with the use of questions such as ‘how’, ‘when’ and ‘why’. Therefore, the researcher must be aware of all aspects of a category i.e. in terms of how it encapsulates properties, dimensions and behaviour. For example, in the case of this thesis study we might want to know ‘what’ situations (causal conditions) cause a student to experience a ‘high’ level of ‘difficulty’ and how the student might deal with this situation order to reduce it to an acceptable level (action strategy).

With the Strauss and Corbin approach, the central or core category (as depicted in fig 3.2) is usually an abstract category (a higher order category that describes a number of concrete categories uncovered in the data e.g. fig 3.3 depicts the higher order abstract category ‘solutions strategies’ which in turn, subsumes the lower level categories ‘illustrative examples’, ‘assistance’ and ‘persistence’ all of which were uncovered in micro-analysis of the data during the pilot study. Codes presented in category development may often take the form of in-vivo codes. An in-vivo code is a code.
Chapter 3

Methodology

based on a verbatim term uncovered in one or more data sources, referred to by Pidgeon and Henwood (1996) as 'member codes' (p.93). For example, an in-vivo code that might replace 'assistance' might be 'look for help'. It should be noted that in fig 3.3 abstracted categories like 'solution strategies' are described by Pidgeon and Henwood (1996) as 'researcher categories' that 'refer to more theoretical ideas that have not been raised directly by the participants' (p.93).

3.4 UNDERLYING ISSUES IN GROUNDED THEORY

As well as understanding the methodological issues involved in grounded theory, it is also incumbent on the researcher to be aware of the existing variants of the method. The original version of grounded theory was developed (Glaser and Strauss, 1967) as an inductive, theory-generating method that differed from conventional deductive, hypothesis testing approaches. Since the original version of the method was developed, two main variants have emerged that are based on different directions taken by its originators Glaser (Glaser 1978) and Strauss (Strauss and Corbin, 1998). Babchuck (1996), in a clear and succinct manner, summarises the essential differences between these variants. Essentially, the Glaser approach takes the view that the informants world should emerge naturally from both data collection and analysis with little or no detail being paid to process on behalf of the researcher. Babchuck (1996) describes this as very much a 'laissez-faire' approach to process. In particular, it espouses total flexibility and advises the researcher against unnecessary constraints like tape-recorders for interviews or prior reading of relevant literature (for fear that it would cloud their analysis with preconceived constructs). On the other hand, the Strauss and Corbin (1998) method has a more prescribed approach that recommends more formal models and procedures to generate theory. According to Babchuck (1996), this approach appears to be 'concerned with producing a detailed description of the cultural scene' (p.3). In fact, Glaser (1992, 2002) refers to this as a 'contextual description'. This detailed description is achieved by using the paradigm model (Strauss and Corbin, 1998) where the researcher attempts to describe a phenomenon in terms of its causal conditions, context etc. Glaser (1992) is vociferous in his vehement objection to this paradigm model:
In actuality it teaches the analyst to make a full conceptual description on data with no questions about whether the links are relevant to any emerging theory that really explains how the participants process their main concerns. And the more the analyst practices the use of this model, the more he will exclude forever his ability to respond to any theoretical code that may emerge and become relevant. He will always just see a condition or consequence irrespective of relevance and stake his professional identity on it. (p.61)

Glaser goes on to assert that the Strauss and Corbin approach is based on Strauss forcing the data into his own ‘pet framework’ (p.64). Furthermore, Glaser (2002) describes what he means by ‘contextual description’:

One concept is generated and then the researcher spends the rest of his time describing it and describing it with incident after incident. There is little or no constant comparative work to generate conceptual properties of the category based on the inter-changeability of indices and conceptual saturation. (p.21)

In this regard, Glaser presents the reader with his distinction between conceptual description and conceptual analysis. In particular, he asserts that using the Strauss and Corbin model the researcher continually over-describes categories with a distinct absence of analysis and ends up with story after story being forced into the concept. In this regard, he refers to theory forcing as opposed to theory emergence.

The issue of the research question is also a cause of divergence between the two approaches. According to Glaser (1992) ‘the research question in a grounded theory study is not a statement that identifies the phenomenon to be studied’ (p.25) but rather emerges out of open coding and sampling based on data that wasn’t forced with preconceived theory forcing questions whereby the researcher begins his/her study ‘with the abject wonderment of what is going on that is an issue and how it is handled’ (p.22). Strauss and Corbin (1998) on the other hand, assert that having a research question allows the researcher to stay focussed in the midst of masses of data.
and in a qualitative study this question will be broad and open-ended. More importantly, they contend that whatever the source of the research question that it is ‘important that the researcher have enthusiasm for the subject because he or she will have to live with it for some time’ (p.53). Charmaz (1990) indicates that grounded theorists begin with a general research question as opposed to pre-conceived hypotheses. Douglas (2003) succinctly summarises the approach taken in this research study, where the researcher ‘commences with an area of study and allows relevant theoretical conceptual constructs to emerge from the process’ (p.47). Whilst the research question doesn’t get adjusted along the way, the researcher may decide to focus and follow up specifically on certain phenomena that emerge in the study and theoretically sample based on emergent theory. This approach was not taken in this study where all voluntary participants were interviewed and their experiences categorised into various phenomena. In this regard, selectivity was not used i.e. this study did not for example seek out only those students experiencing extreme difficulty with programming.

Given these two distinct variants of grounded theory it is necessary for the researcher to make it clear at the outset which version they are using and to remain aligned with the tenets of that model. According to Smith and Biley (1997) ‘it is not sufficient for the researcher to report in a methodology section that a grounded theory approach was utilised’ (p.24) and therefore must make explicit the actual processes that were undertaken in the study. Goulding (1998) recommends that ‘preferences regarding the version adopted should be stated to avoid confusion over terminology and procedures’ (p.57). Furthermore, in a later paper, Goulding (2002) contends choice of grounded theory variant is a crucial issue and states that ‘care should be taken to decide which method best suits the researcher’s personality and preferred modes of working, before embarking on the research’ (p.48). Whilst this contention holds merit, the experience garnered in this study suggests that it doesn’t go far enough. In particular, choice of method should be determined by the nature of the research question and the context of the research (context here refers to the research setting, previous work (if any), characteristics of participants engaging in the study etc.). This issue is discussed in more detail in this chapter in the section on the suitability of grounded theory to the research project.
Whilst it may appear obvious which version of grounded theory is being used i.e. utilisation of the paradigm model demonstrates alignment to the Strauss and Corbin approach, some researchers for whatever reason, may engage in what has been described in other research as 'method slurring' (Cutcliffe, 2000) whereby techniques from differing qualitative research methods are combined in a non-formal manner. If this is undertaken without due regard to the core principles of the respective technique the result may be a 'sloppy mismatch' (Morse, 1991, p.15). In this particular context the differing methods are actually variants of one method. Method slurring, in this context would entail cherry-picking the would-be appealing or 'nice parts' of each variant (or even the easy parts) as opposed to whether or not they were necessary in answering, or contributing to, in some necessary way, the research question. In this context, Smith and Biley (1997) refer to this criss-cross of grounded theory variants as 'muddling methods' (p.28) and Goulding (1998) describes it as 'methodological transgression' (p.56).

In the context of the discussion above, it is important that this thesis defines clearly which variant of grounded theory is being used and clarifies why it is being used. This research project has utilised the Strauss and Corbin (1998) variant of grounded theory. After careful consideration, this variant was chosen because it was more aligned with both the research question and the context of the research. The reasons for this decision will now be outlined.

Firstly, as Babchuck (1997) suggests, whilst in the Glaser approach 'the informant's world should emerge naturally from the analysis with little or no attention to process' (p.3) the Strauss and Corbin approach provides insights into the realities of the participants it also provides 'a detailed description of the cultural scene' (p.3). In this regard, the Strauss and Corbin approach may be viewed as more aligned to answering the research question where the outputs of the research containing both theory and detailed cultural description may prove invaluable for improving programming teaching in the future (which happens to be one of the main objectives of this research). In particular, the paradigm model presented in table 3.1 will enable educators to not only see what phenomena are at play in learning computer programming but will also give them a detailed insight into the context and reported causal conditions of these phenomena (i.e. as reported by the participants in an open-
ended interview). This is considered by the author as an important part of improving and refining programming education. Secondly, the detailed attention to process in this method seemed a good starting point for a novice grounded theory researcher and also in this regard the author received procedural advice from Juliet Corbin, co-author of Strauss and Corbin (1998). Thirdly, within the context of a an Education Doctorate study where the researcher is required to possess a specific research question at the thesis proposal stage, once again one witnesses greater alignment with the Strauss and Corbin variant. Within this method, the researcher is recommended to have a clear research question that provides the focus and clarity of endeavour. Smith and Biley (1997) agree with this stance and recommend that the research question ‘must be flexible and open-ended to allow the theory to develop’ (p.19). In contrast, within the context of this doctoral study, whilst the Glaser (1992) approach of standing back in abject wonderment and letting the research question emerge undoubtedly has pure inductive qualities, it was not feasible for this project given the limited time-window within an academic year to interview students and other non-avoidable logistical constraints. This time window pertained to a time in the year where the participants had exposure to most programming concepts (i.e. the end of the Spring term) as well as not being too close to their examinations.

Finally, the Glaser (1992) approach requires the researcher not to review any literature in the substantive area in advance of the study for fear that it would ‘inhibit, stifle or otherwise impede the researcher’s effort to generate categories’ (p.30). On the other hand, the Strauss and Corbin (1998) approach espouses that if used incorrectly, literature can hinder creativity, but if it is used properly ‘as an analytic tool it can foster conceptualisation’ (p.53). In terms of this research, the author, through professional practice (working as a lecturer in the substantive area for the last 11 years), publication of papers at previous programming conferences, production of papers for the Education Doctorate course has amassed considerable literature in computer programming education. In this regard, using the literature in combination with professional practice to enhance theoretical sensitivity, is aligned with the Strauss and Corbin (1998) approach.

In summary, this section has highlighted the differences between the two main variants of grounded theory methodology. It has shown that the Strauss and Corbin
(1998) approach was chosen for its conceptual alignment with both the research question and the context within which the research took place.

3.5 THEORY FORCING AND THE PARADIGM MODEL

Practical experience gained in this research has found that some common aspects of each variant can be served whilst diligently adhering to the methodological requirements of one. Walker and Myrick (2006) agree with this assertion:

Both Glaser’s and Strauss’s versions of grounded theory use coding, the constant comparison, questions, theoretical sampling and memos in the process of generating theory. Moreover, both versions adhere to the same basic research process: gather data, code, compare, categorize, theoretically sample, develop a core category and generate a theory. (p.550)

In addition, both variants espouse the fundamental issue of inductive theory generation whereby the theory generated ‘fits’ the actual data collected and is not derived from bias or other non-data-related sources. Despite the fact that Glaser harshly criticises the paradigm model of Strauss and Corbin, the work conducted in this research suggests that once the data presented in the paradigm model emanates from the data and is not a result of forced interview questioning then it conforms to the inductive requirement. As an illustration of this issue table 3.2 highlights a paradigm model representation of an abstracted category ‘solution strategies’ uncovered in the pilot study.

It should be noted that the memos noted in the description/memo column of table 3.2 reflect the actual issues described by the participants. For example the context that all participants referred to was the manifestation of a difficulty when it came to doing a project (at that point a lack of understanding of a programming concept became a real issue for them). In this context, the lack of understanding of a concept was seen to
impede the students' progress in a project and therefore to get around this barrier a 'solution strategy' of some sort was required.

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Description (Memo)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenomenon</td>
<td>Solution Strategies</td>
</tr>
<tr>
<td>Causal Conditions</td>
<td>Student encounters a difficult issue in programming that doesn't make sense straight away.</td>
</tr>
<tr>
<td>Context</td>
<td>Understanding a programming concept like this is a prerequisite for progression in a project/assignment. For example the ability to reference a cell position in a two-dimensional array.</td>
</tr>
<tr>
<td>Intervening Conditions</td>
<td>Lecture materials, i.e. notes and code examples will been given to the student illustrating how the concept works, accompanied by a relevant explanation by the lecturer.</td>
</tr>
<tr>
<td>Action/Interaction Strategies</td>
<td>Look back through notes to find an illustrative example. Persist in analysing the examples on an iterative basis until the concept is understood. Seek assistance from another individual who understands the concept.</td>
</tr>
<tr>
<td>Outcome/Consequences</td>
<td>Student either understands the concept or decides it is too difficult and looks to use another concept in its place if possible (i.e. a concept that they do understand).</td>
</tr>
</tbody>
</table>

Table 3.2 Paradigm for Abstracted Category 'Solution Strategies'

Table 3.2 reflects the presentation of data where participants discussed their approach to dealing with the computer programming subject. In an open-ended interview it is normal for the researcher to ask questions on *when* and *how* a certain situation manifests itself, e.g. 'when do you find this?' or 'how do you deal with this?'. Answers to these types of questions can be presented in the paradigm frame as illustrated in table 3.3. Whilst being aware of Glaser's (1992) vehement objection to this paradigm model, this research has found that presenting categories uncovered in the data in this format gives educators an interesting insight into certain phenomena in terms of *how*, *when* and *why* they occur. This can serve as useful material in programming education reform in terms of subject matter delivered and delivery format. Furthermore, once the information presented in the paradigm frame is
extracted from the data (as a result of how and when questions being asked) then it cannot be construed that it is forced. If the structure of the paradigm model forced the researcher into asking specific paradigm-specific questions to conform to its structure then one could argue the forcing issue.

Given the above discussion, analysis of the last three elements in the paradigm frame (illustrated in table 3.2) conducted in this research suggests that their contents can be a product of natural unforced questioning in an in-depth interview as illustrated in table 3.3. This table presents an example of methodological clarity, i.e. that of the researcher having a clear and precise understanding of their chosen variant and confidence in the fact that they are not deviating from inductive theory generation. As is clear from an inspection of table 3.3, none of the questions listed could be construed as leading or forcing, but rather stem out of a desire to get a more in-depth understanding of the participant’s experience in learning computer programming.

<table>
<thead>
<tr>
<th>Intervening Conditions</th>
<th>When do you find this is the case?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action/Interaction Strategies</td>
<td>When in this situation, how do you deal with it?</td>
</tr>
<tr>
<td>Outcome/Consequences</td>
<td>How do you find this works for you?</td>
</tr>
</tbody>
</table>

Table 3.3 Natural Unforced Questioning

This finding is consistent with the assertion of Priest et al. (2002) who tell us by using the paradigm model, the analyst is ‘guided to think about what caused the phenomenon to occur’ (p.34) as well as investigating present conditions at the time and what actions or consequences emerged as a result.

Another Glaser (1992) criticism of the paradigm model pertains to the development of properties and dimensions of categories uncovered in open coding. According to Walker and Myrick (2006) this criticism alleges that automatically developing the dimensions of a property is a jump too far in the coding process. However, Walker and Myrick (2006) go on to state that this dimensionalisation is helpful as it ‘breaks the data down and assists in the development of relationships among categories’ (p.552). In this regard, this research has found that looking to artificially create dimensions of properties that do not exist would lend weight to the theory forcing
argument, and therefore it was avoided in this research. Despite using the Strauss and Corbin approach, this approach was taken because it was decided that artificially creating any component be it category, property or dimension that did not emanate from the data would fly in the face of induction which is the over-arching principle in both variants of the grounded theory method.

3.6 THE RESEARCH QUESTION

As mentioned previously in this chapter, the Strauss and Corbin variant of grounded theory advises the researcher to have decided on a research question in advance. This contrasts to the Glaser approach of letting both the research question and its subsequent theory emerge from open-ended interviewing. In this study, the research question was decided upon in advance but its nature was such that whilst it was specific, it was open enough to allow unforced theory to emerge and facilitated any experiences of programming to be uncovered using open-ended questioning. This approach is consistent with the recommendation of Backman (1999) who indicates that ‘the questions are formulated so that they give the researcher the flexibility and freedom to explore the phenomenon in depth’ (p.149). This detailed consideration of the nature of the research question paid off for two main reasons. Firstly, the initial aim of the author was to seek out participants who are experiencing significant difficulty with programming. In hindsight, this might have had the effect of categorising participants in advance and to an extent pre-determining the theory in a forced manner. In fact, actual data collection found that all students experienced difficulty, albeit of differing levels and extents and it was the way they dealt with that difficulty which emerged as of significance to the study. Secondly, when visiting potential research sites and talking to students, it was important to be able to convey to them the exact nature of what the research was about. In this regard, given the open nature of the research question, potential participants were informed that each one of them had an individual experience of programming, each of which was equally valuable for this study. In addition, explaining the research question in this way emphasised the fact that the data collection wasn’t aiming to ascertain how much
programming they did or did not know but rather what their learning experience was like on their first year course.

As Smith and Biley (1997) indicate ‘if the researcher has not made the initial thoughts driving the study explicit, it is impossible for the reader to evaluate how successful or valid the grounded theory method as been’ (p.25). In this regard, Pidgeon and Henwood (1996) clearly state that ‘one aspect of the grounded theory approach that we want to highlight is the requirement to document the analytical process fully’ (p.86). In light of this, the process undertaken in this research is made explicit in this thesis.

3.7 THE RESULTANT THEORY

According to Charmaz (1990) the emergent categories from a grounded theory study both explain and conceptualise ‘(1) the data, (2) common sense understandings of these data, and, likely, (3) other theoretical interpretations’ (p.1162). Charmaz goes on to state, ‘in my view a theory explicates a phenomena, specifies concepts which categories the relevant phenomena, explains relationships between concepts and provides a framework for making predictions’ (p.1164). Brown et al. (2002) describe another desirable characteristic of a grounded theory which is that of elegance. An elegant grounded theory study produces a theory where the fewest possible concepts are generated with the widest possible scope. Glaser (1978) refers to elegance by highlighting the need for developing concepts with ‘as much variations as possible in the behaviour and problem under study’ (p.125). Essentially, elegant grounded theory develops theory where multiple manifestations of a concept are identified and described. Pidgeon and Henwood (1996) describe this as a major aim of grounded theory where the researcher should aim to seek similarities and diversities, collecting a range of indicators that point to the multiple qualitative facets of a potentially significant concept. This concept of elegance is consistent with the aims of theoretical saturation and constant comparison.
Chapter 3

As may be evident from this chapter, generating a theory using this process is not a simple task. In order to generate theory that most accurately reflects the data, the researcher must possess the ability to disengage from the data to create theory (Backman 1999). This requires abstraction skills that if not present, the resultant theory may be ‘naïve, concrete and written in the same terms as the data’ (Backman, 1999, p.151). Borgatti (2004) reinforces this stance by indicating that ‘it is important to have fairly abstract categories in addition to fairly concrete ones, as the abstract ones help to generate general theory’ (p.2). This is similar in concept to the criticism of full contextual description as presented earlier in this chapter. Goulding (2002) describes this as the requirement to have ‘conceptual density and meaningful variation which should go beyond thick description’ (p.45). In light of these issues it is important that the researcher possess not only theoretical knowledge of the problem domain but also possesses the ability to stand back from the data and produce abstracted categories as illustrated in the ‘solution strategies’ category in table 3.2.

3.8 METHODOLOGICAL ISSUES ENCOUNTERED

Whilst engaging in grounded theory implementation in this research, it was found that a number of methodological issues were intertwined in a way than can cause difficulty for the researcher. As mentioned earlier in this chapter, the sampling mechanism used in grounded theory is theoretical sampling. Rennie and Phillips (1988) described this as a sampling approach that is influenced by the outcomes of the emerging analysis and that the collection of data ‘proceeds through successive stages which are determined by changes in criteria for selecting the data sources, according to what has been learned from previous data sources’ (p.142). Whilst conducting the grounded theory analysis it was found that there is a possibility of conflict between theoretical sampling and theory forcing. In particular it was found that going down a line of enquiry (based on initial emergent categories) and theoretically sampling on that basis, could possibly limit substantive theory generation, and also result in the imposition of emergent category-specific questions on subsequent participants. For example, if initial participants recounted that they were experiencing learning difficulties with programming, theoretically sampling on the basis of experiencing
difficulty alone was considered to be an unwise strategy that appeared to be inconsistent with inductive theory generation. In this regard, if initial participants had reported the ease with which they found programming and *ease* appeared to be an emergent category that seemed well on the way to approaching saturation, this might cause the researcher to only seek out those who were experiencing ease in subsequent samples. Sampling of this nature would seem to be determined by accidental chronological placement of participants. This situation is exacerbated by the fact grounded theory allows interview questions to be adjusted in subsequent data collection stages to facilitate theory saturation (Goulding, 1998; Hansen and Kautz, 2005). In this case changing interview questions to further saturate experiences of *ease* and posing them to a participant who is experiencing *difficulty* may have a number of unfortunate consequences that are not conducive to inductive theory generation. Firstly, given the experience gained in the pilot study, it may cause the participant to question their own lack of ability when presented with ease as a starting point. Secondly, it may result in the accusation of theory forcing where the researcher rather than looking for genuine concepts to emerge, actually tries to saturate their predetermined theories to the detriment of what data actually emerges in truly open-ended interviewing.

The danger of this theory stifling based on rigid adherence to theoretical sampling was also encountered by Charmaz (1990) who refers to ‘premature commitment to a set of analytic categories’ (p.1164). Here she refers to commitment to categories where the researcher ‘has not fully explored the issues, events, and meanings within the research problem, or setting and has not gained intimate familiarity with it’ (p.1164). This research has found that this situation presents a methodological dilemma for the researcher who wishes to sample in the correct fashion in conjunction with the constant comparative technique to successfully saturate context-specific theory. In dealing with this dilemma, it was decided in this research to adopt the recommendation of Rennie and Phillips (1998) where they advise that:

> Participants are selected who seem likely to represent the phenomenon and who are relatively similar. This is done in order to maximise the chances that aspects of the phenomenon will emerge clearly. (p.142)
Chapter 3 Methodology

It was found that adopting this approach of selecting participants based on whether or not they have experienced a first year programming course was successful as well as facilitating all types of experiences to emerge. This approach is consistent with the approach of open sampling as described in the Strauß and Corbin (1998) methodology to which this project is aligned. Using this approach, 'sampling is open to those persons, places, and situations that will provide the greatest opportunity for discovery' (p.206). Those emergent categories that reached saturation (in as far as saturation is possible) were reported and those that did not were not presented as part of the substantive theory presented in this thesis. It was found that whilst this approach to sampling could be argued as leaning towards the 'purposeful' side of theoretical sampling it did avoid the potential pitfalls of theory forcing and premature commitment as well as being consistent with the Strauß and Corbin (1998) variation of grounded theory (this issue will be addressed again at a later stage in this thesis). It was found that in situations like this the pragmatic researcher must take context specific factors into consideration and make a judgement call that will result in the most accurate presentation of the data possible. Furthermore, it is incumbent on the researcher to report in an honest and open fashion the methodological choices that were made during the data collection and analysis phases.

3.9 CONCLUSION

This chapter has provided a detailed description of the grounded theory method with particular reference to the Strauß and Corbin (1998) variant actually used in this research. It has described the successive levels of coding used in grounded theory and how abstraction plays an important role in the theory generation process. In addition, it has presented the reasons as to why grounded theory (and in particular the Strauß and Corbin variant) is considered to be a suitable research method to address the chosen research question. Furthermore, it has highlighted how the author has analysed in an in-depth fashion the underlying issues in grounded theory in advance of undertaking data collection and analysis. The net result of this pre-emptive analysis enabled the application of grounded theory to the research question in a methodologically sound and deliberate fashion.
Chapter 4

Data Collection

4.1 INTRODUCTION

This chapter describes the data collection methods used in this study. Whilst the interview is the most commonly utilised data collection technique in qualitative research, other methods implemented in the study will also be described in this chapter. In terms of applying each of these data collection methods, the focus in this study was to ensure the tenets of grounded theory were adhered to at all times. In particular, there was a deliberate policy of facilitating the participants to use their own voices, ideas, attitudes and experiences (despite how trivial they may have considered
these themselves) and a deliberate avoidance of any type of leading or theory-embedding questions that could lead to the criticism of theory forcing. After detailed consideration, analysis of literature, consultation with experienced grounded theory practitioners and pilot study implementation it was decided that a number of techniques would be used for data collection. In particular, the data collection methods utilised comprised interviews, focus groups and journals/diaries. The remainder of this chapter will discuss in detail each of these data collection methods and the practical issues that arose during the data collection phase of this study.

4.2 DATA COLLECTION PROCESS

The data collection process conducted in this research was not engaged upon until an in-depth understanding of grounded theory and its methodological considerations was completed. Creswell (1998, p.113) provides a table (see table 4.1) that serves as a very useful reference point in terms of summarising grounded theory data collection using the Strauss and Corbin (1998) methodology.

<table>
<thead>
<tr>
<th>Data Collection Activity</th>
<th>Grounded Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is traditionally studied? (site/individual(s))</td>
<td>Multiple individuals who have responded to action or participated in a process about a central phenomenon</td>
</tr>
<tr>
<td>What are the typical access and rapport issues? (access and rapport)</td>
<td>Locating a homogeneous sample</td>
</tr>
<tr>
<td>How does one select sites or individuals to study? (purposeful sampling strategies)</td>
<td>Find a homogeneous sample, a “theory based” sample, a “theoretical” sample</td>
</tr>
<tr>
<td>What type of information typically is collected? (forms of data)</td>
<td>Primarily interviews with 20-30 people to achieve detail in the theory</td>
</tr>
<tr>
<td>How is information recorded? (recording information)</td>
<td>Interview protocol, memoing</td>
</tr>
<tr>
<td>What are common data collection issues? (field issues)</td>
<td>Interviewing issues (e.g. logistics, openness)</td>
</tr>
<tr>
<td>How is information typically stored? (storing data)</td>
<td>Transcriptions, computer files</td>
</tr>
</tbody>
</table>

Table 4.1 Data Collection Overview
Whilst engaging in data collection, each of the issues highlighted in table 4.1 emerged. In this study data was collected from multiple participants across four separate sites. As grounded theory requires the sample to be homogeneous, all participants were engaged in a first year programming subject as part of a computing/computer science degree. Across the four sites 26 participants were interviewed, 10 engaged in focus group discussions and 3 submitted journals/diaries (as illustrated in table 4.2). Interviews and focus group sessions were recorded and transcribed at a later stage. Accessing participants at a time when they had adequate exposure to a programming course and when it caused minimal disruption to their course were examples of logistical issues that had to be dealt with.

4.2.1 Interviews

The purpose of in-depth interviewing is not to get answers to questions, nor to test hypotheses and not to evaluate. At the root of in-depth interviewing is an interest in understanding the experience of other people and the meaning they make of that experience. (Seidman, 1998, p. 3)

Interviews are the most common data collection technique used with the grounded theory method. In fact, Creswell (1998) suggests that other forms of data collection other than interviews only play a secondary role. This was the experience in this study where interviews and focus group sessions allowed the author to ask follow-up questions such as ‘why’, ‘how’, and ‘when’ a phenomenon was experienced. Whilst the diaries/journals did provide insights into task-related activities e.g. how they progressed on a problem solving task, they did not facilitate follow-up questions. As will be illustrated throughout this thesis, data collection and analysis techniques were chosen based on their suitability to addressing the research question. Seidman (1998) illustrates how applicable interviewing is in this research study where he states that ‘it is a powerful way to gain insight into educational issues through understanding the experience of individuals’ (p.7). Given the importance of proper interviewing in data collection, literature was analysed with the aim of ascertaining the crucial issues involved in qualitative interviewing. In this regard, Fontana and Frey (1994) draw the
researcher's attention to a number of important issues, a subset of which, were pertinent to this proposed study:

- Accessing the setting
- Understanding the language and culture of the respondents
- Deciding on how to present oneself
- Gaining trust
- Establishing rapport

In terms of access, student cohorts were chosen with whom the author has had no prior contact. This was done primarily for ethical reasons, in terms of avoidance of a conflict of interests and putting undue pressure on students to recount their genuine experiences with a person directly involved in that experience. Understanding the language and culture of the participants was not difficult as the author has been working as a programming lecturer with first year Irish programming students for over a decade and is acutely sensitive to the language and educational background of the participants. In particular, educational background refers to the nature of the primary and secondary school system in Ireland where, for example, the vast majority of students take lower level mathematics at Leaving Certificate level and do not take science subjects. As an example, statistics\(^1\) compiled by the Irish Department of Education on the Leaving Certificate examination highlight that over the past three years:

- 19% of candidates took higher level mathematics
- 14% of candidates took physics\(^2\) (this figure includes both higher and lower)
- 14% of candidates took chemistry (this figure includes both higher and lower)

In the pilot study that was conducted, presenting oneself, gaining trust and establishing rapport were found to be inextricably linked. Here, the participants were presented with a summary of the research project outlining its purpose, scope etc. Participants were given the option of not having the interview taped, however none of


\(^2\) Physics, chemistry and mathematics are listed as they are the main problem solving oriented subjects at Leaving Certificate level.
them chose to exercise this option. Great care was taken to gain the trust of the participant by explicitly stating that the interview was not an assessment of the extent of their knowledge, but rather an open discussion of their experiences with programming. As in the pilot study, the participants were made aware that there weren’t any right or wrong answers to the questions posed and that their personal experiences were being sought. This had the effect of making the participants feel at ease. This was aided by reinforcing the fact that their personal experiences were invaluable and that confidentiality was assured at all times. In the pilot study the author was very much aware that first year students would have little, if any, interview experience of this nature and every effort was made to reassure them that whatever they stated in the interview was pertinent and of interest to the researcher. Other commentators (Berry, 1999; Seidman, 1998; Deakin, 2004; Wimpenny and Gass, 2000) highlight the importance of interviewing technique. One of the most important issues addressed by these commentators is the fundamental nature of truly open-ended questions. According to Seidman (1998) ‘an open-ended question, establishes the territory to be explored while allowing the participant to take any direction he or she wants. It does not presume an answer.’ (p.69). In the context of a grounded theory research study, avoidance of questions that could be construed as theory forcing was deemed imperative. This issue can be explained with a simple example:

Question: ‘How satisfied were you with your teaching placement?’

Here, unwittingly the interviewer plants an idea in the mind of the respondent, i.e. the notions of satisfaction or dissatisfaction. Rather than asking them to recount their actual placement experience the interviewer asks them to speak about it within a certain frame within which they may feel uncomfortable. Wimpenny and Gass (2000) describes this as over-zealous, overly directive interviewing ‘whereby interesting directions and rich data are lost in the encounter’ (p.1488). Furthermore, they illustrate a similar type of leading question ‘tell me, how do you see empathy?’ (p.1488). Here, the question assumes that the participant has an opinion on empathy and results in forcing as opposed to enabling the emergence of theory. The approach to avoid leading of theory forcing questions is to ask truly open-ended questions e.g. ‘what was your student teaching placement like for you?’ (Seidman, 1998, p.69).

77
Charmaz (1990) also provides very good advice in terms of the nature of interview questions. In particular she indicates that the overly directive researcher can cut off interesting leads and potentially rich data. In addition, she highlights that the researcher should avoid loading assumptions into questions. For example rather than asking a patient 'how did you decide to have surgery' (which assumes the patient made the decision), the interviewer should ask 'how did you come to have the surgery?' As in the previous examples this leaves the question open to the participant to recount their own actual experience of a phenomenon. Charmaz (1990) goes on to state that these types of questions enable the optimal elicitation of narrative from the participant whilst requiring minimal framing by the researcher.

The combination of all of these considerations resulted in a comfortable interview setting for both the researcher and the participant in the pilot study interview sessions and subsequently in the full study.

**Interview Design**

In designing the study a decision had to be made in terms of the type of interview to be used. Patton (2002) describes three different approaches to interviewing:

*Informal Conversational Approach:* One that relies on the spontaneous generation of questions in the natural flow of interaction.

*General Interview Guide Approach:* One that outlines a set of issues to be explored with each participant before interviewing begins.

*Standardized Open-ended Approach:* One that consists of a set of questions carefully worded and arranged with the intention of taking the respondent through the same sequence of questions, asking each participant the same questions with the same words.

Given the nature of grounded theory and its reliance on the constant comparative technique, it seemed logical that rigid adherence to a fixed standardised set of questions would not be suitable. Goulding (2002) supports this stance on non-rigidity
of interviews and states that the interview ‘should be flexible enough to allow the discussion to lead into areas which may not have been considered prior to the interview but which may be potentially relevant to the study’ (p.59). Analysis of the data immediately after collection may result in new questions to be asked to subsequent participants in order to saturate a category, and this is deemed as a necessary task in grounded theory. This is consistent with the in-depth interviewing experiences of Berry (1999) where she states that they can not only be used to get a holistic understanding of the participant’s situation but also ‘it can be used to explore interesting ideas for further investigation’ (p.2). This further exploration is considered a necessary part of theoretical saturation. In this regard, it was deemed necessary to combine the standardised approach with the conversational approach whereby certain standard questions will be asked as well as having freedom to go down non-preconceived routes in terms unpredicted responses from participants. This type of freedom is consistent with the notion of theoretical sampling where sampling is done based on evolving categories. Furthermore, this is consistent with the notion of a ‘directed conversation’ as espoused by Pidgeon and Henwood (1996, p.89) and freedom to explore as discussed by Berry (1999).

In addition to the general questions posed, the use of probes was also implemented. Patton (2002) suggests three types of probes that may be suitable to this study:

**Detail oriented**

What was involved in this task?

**Elaboration oriented**

Could you tell me more about that if you can?

**Clarification oriented**

You said ‘success’ what do you mean by ‘success’?

Seidman (1998) highlights that the term ‘probe’ has negative connotations and prefers the term ‘exploration’. In particular he states:

> Literature often uses the word *probe*. I have never been comfortable with that word. The word conveys a sense of the powerful interviewer treating the participant as an object. I am
more comfortable with the notion of exploring with the participant
than with probing into what the participant says. (p.68)

In any case, whether it is labelled ‘probing’ or ‘exploration’ it is important to achieve
the optimal balance between necessary exploration and ill-timed interruption.
According to Seidman (1998) too little exploration can result in the researcher being
unsure of the actual meaning in the data. Furthermore he states that ‘it can also leave
the participant using abstractions and generalities that are not useful’ (p.69). It was
deemed imperative in the design of this study to take cognisance of this fact.
Grounded theory is a bottom-up method, where abstractions are developed during the
axial coding phase of data analysis. This research has found that raw data in the form
of non-decomposed abstractions and generalities are not suitable for two reasons.
Firstly, if the researcher is to use data in this form they may have to make
assumptions as to the meaning in the data. This deviates from one of the fundamental
tenets of grounded theory where theory must be derived from actual data not from
perceptions or assumptions. Secondly, if abstractions emanating from the participant
are not explored, they will be of little or no use in category saturation required in
grounded theory.

4.2.2 Focus Groups
Patton (2002) defines in a succinct and clear manner the exact nature of a focus group.
He highlights that it is a type of interview, not a problem solving session nor a
decision-making session. Furthermore, he indicates although interactions may happen
between participants it is not a set of informal discussions between participants but
rather it is an interview. According to ASA (1997) focus groups are ‘in-depth qualitative interviews with a small number of carefully selected people’ (p.1). It is
important to note here that this notion of ‘careful selection’ is consistent with
theoretical sampling as described earlier. Theoretical sampling entails seeking out
information rich participants that can provide valuable data relating to emerging
theory. Here, the researcher attempts to sample to achieve the widest variation
possible in relation to emergent theory. For example, if an emerging theory in
programming is that of ‘difficulty’ we may sample in a way that permits the
maximum number of variations of experiences of this concept to be uncovered and thus, saturate the category. Therefore the objective of a focus group is to acquire ‘high-quality data in a social context where people can consider their own views in the context of the views of others’ (Patton, 2002, p.386). Here the researcher assumes the role of moderator where they moderate or guide the discussion. A key skill required of the moderator is to nurture the conversation flow so as to gain as much theoretically rich data as possible. On the other hand, literature suggests (Patton, 2002; ASA, 1997; Cohen et al., 2000; Anderson, 1998) there are many potential pitfalls associated with focus groups and these will have to be avoided in the data collection process. A particular pitfall that is pertinent to this research is what Cohen et al refer to as extreme care with sampling. Here they suggest that ‘every participant is the bearer of the characteristic required or that the group has homogeneity of background in the required area’ (p.288). In terms of the proposed research a focus group that combines a mix of students with varying abilities may cause those experiencing problems in programming to feel inadequate as compared to their more able counterparts and therefore may hinder them from intimating the issues they experience for fear of inadequacy. This issue is frequently encountered in an educational setting where students experiencing difficulty prefer to ask questions in a one-to-one tutorial session or in a laboratory setting. In this regard, during the initial briefing session, potential participants were given a description of each of the three data collection techniques and were informed that they could volunteer for any one (or all) of their choice. Table 4.2 highlights that a number of participants volunteered for both interviews and focus groups. Anderson (1998) gives detailed advice to assist the task of focus group moderation in terms of dealing with the varying participant types as well as providing useful tips on data analysis. These were studied in detail in advance of the data collection and utilised accordingly.

4.2.3 Journals/Diaries

Process journals are described by Lewandowski (2003) as informal journals kept by students as they work on a programming project (the use of similar types of journals in a qualitative study is reported by Carbone and Mitchell, 2001). They can be used to acquire qualitative information about a student’s problem solving and programming
process. One of the advantages of using such journals is that students can highlight the real issues they experience, without having to feel conscious about discussing problems with interviewers or groups. In this way, they may be more open to discuss more issues in an open and frank manner. In fact, this phenomenon can be witnessed on electronic bulletin boards where students can anonymously post the issues they encounter in an open and unconstrained way. The intention with this study was to provide students with diaries either in hard copy format or electronically. The diaries asked them to document their experiences with programming over a typical week. The participants were asked to add anything they felt like documenting in relation to their programming work. Participants were also given instructions on filling in diaries. The diaries were provided to participants in electronic format. A copy of the diary can be viewed in appendix 1.

4.3 PREPARATION AND IMPLEMENTATION

As mentioned at the beginning of this chapter, careful preparation went into the data collection section of the chapter. Careful consideration was given to many practical issues such as access, logistical constraints and suitability of sites and participants in terms of the research question. These issues will be discussed in this section.

4.3.1 Data Collection Sites

One of the key issues in any research project pertains to gaining access to suitable participants at one or more data collection sites. When undertaking this research it was found that this is an issue not to be underestimated. In this regard, contact was made with lecturers and heads of computing in a number of third level institutions whose geographical location made it possible for the author to visit them at short notice. Subsequently, data collection consisting of interviews, focus groups and journals/diaries was undertaken at four separate sites. Each of these sites was an Irish third-level educational institution offering computing courses to first year students. It was decided that whilst having multiple research sites would present logistical problems it would have the possibility of generating theory that represented the
experiences of a broad range of geographically dispersed participants. In fact, Creswell (1998) makes particular reference to this issue when he states:

‘In a grounded theory study, the individuals may not be located at a single site, in fact if they are dispersed, then they can provide important contextual information useful in the axial coding phase of the research. They need to be individuals who have taken an action or participated in a process that is central to the grounded theory study’. (p.114)

At each individual site there was a lecturer who acted as a contact point with the author and thus, played a key role in terms of access. The individuals concerned were contacted by email initially after which discussions of the proposed study ensued. In advance of the author attending the data collection sites the lecturers gave the students an overview of the research and asked if any of them were interested in the author coming to their institute to describe the research project. In the four data collection sites the students were happy for the author to come along and give an overview of the research followed by question and answer session. This visiting of data collection sites will be discussed in more detail in the next section.

The choice of interview timing proved critical to the process as the facilitating lecturers in each of the institutes contacted the author as to when would be the least disruptive time to interview the participants. It so happened that the participants were interviewed at each of the sites within the space of three weeks of each other coinciding with close proximity to the end of the second term approaching the Easter break. This time window was viewed as optimal for two reasons. Firstly, participants would have covered at that stage the vast majority of their computer programming syllabi. Secondly, any time later in the year would have been too close to the exams and thus, would have been an unfair distraction to students preparing for examinations. The interviewing process at this time of the year proved to be very successful and caused the minimum disruption to the students who participated.
4.4 LOGISTICAL CONSTRAINTS

As is possible in any study, certain logistical constraints were experienced during this research. These constraints were exacerbated by the fact that data collection was conducted at multiple data collection sites. One constraint that immediately emerged related to sampling. In the last chapter the sampling strategy employed was described as 'open sampling'. In this research, the participants volunteered in advance of data collection. In situations like this Backman (1999) describes the sampling strategy as a type of selective sampling that is used by grounded theory researchers where 'the subjects are mainly chosen before the data collection and the data are collected within a certain time' (p.149). Backman (1999) goes on to describe that this approach is a result of restrictions imposed on the research work. In this research these restrictions took the form of interviewing students at multiple research sites at times that were least disruptive to participants' course work. Furthermore, in order to minimise disruption and to acquire the optimal recounting of the participants' experiences, all sites were visited in the space of a couple of weeks. In addition, due care was taken neither to interview at a time too close to examinations nor too early in the term where participants wouldn't have had exposure to the full first year programming experience.

In this study, logistical constraints also impinged on saturation. Theoretical saturation requires the researcher to continue data collection in the field until one gets to a stage where no new information about the categories is being uncovered. In the experience gained in this study, logistical and contextual constraints make this quite difficult. In the context of third-level students, it would be unfair to ask participants to attend subsequent interviews at a time close to their exams. In this situation all the researcher can do is present the categories that reached saturation point (if this is ever possible given the intrinsic evolutionary nature of theory generation) and perhaps identify others that are short of saturation but may be looked at in future research. In this regard, Strauss and Corbin (1998) advise the researcher to be aware of constraints that may frequently exist in a grounded theory study:
Chapter 4 Data Collection

In reality, if one looked long and hard enough, one always would find additional properties or dimensions. There is always the potential for the "new" to emerge. Saturation is a matter of reaching a point in the research where collecting additional data seems counter productive; the "new" uncovered does not add that much more to the explanation at this time. Or as is sometimes the situation, the researcher runs out of time, money, or both. (p.136)

This pragmatic advice from Strauss and Corbin (1998) enables the researcher to be aware that just because real constraints exist, the quality of the study can still be maintained if the researcher stays within the confines of non-forced rigorous analysis. Just as true randomness may be difficult to achieve in quantitative studies, issues like pure theoretical saturation and theoretical sampling in qualitative studies can be difficult to achieve given the existence of many constraints that emerge in the field.

4.5 ENLISTING PARTICIPANTS

Once the research sites had been decided upon, the next task entailed enlisting participants. This contact lecturer acted as a gatekeeper and facilitated a number of important tasks in terms of access. Firstly, they facilitated the formal granting of permission from the relevant heads of department. This was achieved by the author providing a formal letter documenting the ethical approval of the study from the University of Leicester. Secondly, they described the purpose of the study to their students in advance of the author attending for the initial contact briefing session and informed them voluntary nature of participation in the study. Thirdly, they allowed the author to visit each site to talk to the programming students for approx 15-20 minutes at the beginning of one of their programming lectures (i.e. the aforementioned first contact briefing session). Whilst choosing participants may be viewed as a crucial issue in all research projects, it is particularly the case in a grounded theory study where the participants must fulfil the requirement of theoretical sampling as described in chapter 3. Initially it was planned that a general e-mail would be sent to all first year programming students at each of the data
Chapter 4

Data Collection

collection sites describing the purpose and nature of the project as well as inviting participation. However in preparation for the interviewing process, the advice of Seidman (1998) in relation to the researcher making personal contact with the potential participants was taken and implemented:

Do it yourself. Try not to rely on third parties to make contact with your potential participants. No matter how expedient it seems to have someone else who knows potential participants explain your project to them, try to avoid doing so. Building the interviewing relationship begins the moment the potential participant hears of the study. Third parties may be very familiar with potential participants, but they can seldom do justice to the nature of someone else's project. They have not internalised it in the way the researcher has; they do not have the investment in it that the researcher does. Once having introduced the subject, they can seldom respond to questions that naturally might arise. Third parties may be necessary for gaining access to potential participants but should be used as little as possible to make actual contact with them. (pp. 39 – 40)

In this regard, the author, as described earlier engaged in a contact briefing session at each site, approximately one week in advance of data collection. This strategy recommended by Seidman (1998) had a number of benefits. Firstly, and probably most obviously, it gave the potential participants to meet the researcher in advance of the research. This resulted in those participating being comfortable with the researcher and the research project. Secondly, it allowed the author to answer any questions relating to the research project in terms of confidentiality and other issues of concern to the participants, such as being able to pull out of the study at any time or engaging in only some of data collection activities e.g. being able to fill in a diary and not have to do an interview. Finally, and very importantly, it allowed the author to reassure the potential participants that the data collection process was not about asking them how much programming material they actually knew or didn't know but rather was looking for an open discussion on their own personal experience of the programming course. On reflection, it is the contention of the author that these issues, that were
viewed as important to the participants, could not have been clarified in a satisfactory manner had personal contact had not been made. During this information briefing session an explanatory information forms and participation forms were handed out students. Those students who were interested in participating returned the participation form to the author and volunteered for the study. This information and participation forms are illustrated in appendix 1.

<table>
<thead>
<tr>
<th>Number</th>
<th>ID</th>
<th>Interview</th>
<th>Focus Group</th>
<th>Diary</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>P4</td>
<td></td>
<td></td>
<td>1</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>P7</td>
<td></td>
<td></td>
<td>1</td>
<td>M</td>
</tr>
<tr>
<td>6</td>
<td>P5</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>7</td>
<td>P8</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>8</td>
<td>P9</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>9</td>
<td>P10</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>10</td>
<td>P14</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>11</td>
<td>P16</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>12</td>
<td>P18</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>13</td>
<td>P19</td>
<td>1</td>
<td>1</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>P20</td>
<td>1</td>
<td>1</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>15</td>
<td>P21</td>
<td>1</td>
<td>1</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>16</td>
<td>P22</td>
<td>1</td>
<td>1</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>17</td>
<td>P23</td>
<td>1</td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>18</td>
<td>P24</td>
<td>1</td>
<td>1</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>19</td>
<td>P25</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>20</td>
<td>P28</td>
<td>1</td>
<td>1</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>21</td>
<td>P29</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>22</td>
<td>P33</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>23</td>
<td>P35</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>24</td>
<td>P36</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>25</td>
<td>P37</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>26</td>
<td>P39</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>27</td>
<td>P41</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>28</td>
<td>P42</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>29</td>
<td>P43</td>
<td>1</td>
<td>1</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>30</td>
<td>P44</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>31</td>
<td>P45</td>
<td>1</td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>26</td>
<td>10</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2 Data Collection Participation Summary
As mentioned earlier, three data collection methods were used, namely interviews, focus group sessions and diaries/journals. Table 4.2 gives a summary of the participants and the data collection methods they participated in. In particular, illustrates that there were 31 participants, some of which engaged in more than one data collection activity. In order to preserve the anonymity of the respondents, a breakdown of groups by data collection site was not undertaken. Furthermore, there were two focus group sessions with participating numbers of 3 and 7 at each session respectively. Given the voluntary nature of participation only one student volunteered for a focus group discussion at the other two sites and thus, this form of data collection was not feasible. This student had also volunteered for interview and therefore was not lost as a participant.

4.6 THE PROGRAMMING COURSE AT EACH SITE

All of the participants were first year students engaged in the study of computer programming. In three of the data collection sites the Java programming language was used, whilst the C/C++ language was used as the language of instruction at the fourth site. This was determined in advance not to be a problem as the syntax of the Java language is derived from C. Furthermore, in advance of the data collection a meeting was undertaken with each of the contact lecturers to discuss the nature of the material covered with the students. The results of these discussions found that the progression of the material at each site followed the standard approach to first year programming where material generally followed the structure illustrated in table 4.3 (based on the author's discussion with lecturers at each site, his own professional experience as well looking as syllabi taught at other Irish Institutes (Madden and Chambers, 2002; Hyland and Clynch, 2002) and outside of Ireland (Herrmann et al., 2003; Garner, 2005).

As highlighted in table 4.3 a first year programming syllabus typically begins with the fundamental material and builds up to more complex concepts like parameter passing and polymorphism. It should be noted that rather than being a discrete set of topics, computer programming syllabi operate on a building block approach where a specific
Chapter 4 Data Collection

topic requires accommodation of all of the previous topics, for example writing code to manipulate array data structures requires the student to understand all of the topics and concepts that precede it in table 4.3.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Brief Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Solving</td>
<td>Introduction to problem solving and algorithm development</td>
</tr>
<tr>
<td>Computer Language</td>
<td>Introduction to the relevant programming language, its structure and syntax. The concepts of variables and data types are also introduced.</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Input and output of data, simple arithmetic expressions and the notion of sequential processing</td>
</tr>
<tr>
<td>Selection Control</td>
<td>Selection control using primarily “if” and “switch” statements</td>
</tr>
<tr>
<td>Iteration Control</td>
<td>Iteration control structures of “for” and “while”</td>
</tr>
<tr>
<td>Characters and Strings</td>
<td>Introduction to non-numeric data types</td>
</tr>
<tr>
<td>Arrays</td>
<td>Introduction to 1-D and 2-D arrays of various data types e.g. integer, character, string etc</td>
</tr>
<tr>
<td>Object Orientation</td>
<td>Objects, classes, methods, attributes, parameter passing, polymorphism</td>
</tr>
</tbody>
</table>

Table 4.3 Common Programming Course Structure

It should also be noted that in terms of teaching methodology, two sites used the traditional lecture and laboratory approach where laboratories enable the student to work on problems based on material presented in the lectures. The other two sites utilised a hybrid Problem-Based Learning approach (PBL) (Greening, 1998). Whilst pure problem based learning does not require lectures but rather constant coverage of a progressive set of problems in a pure laboratory environment, the hybrid approach taken at the two sites was problem based laboratory work supplemented with lectures.

Finally, it should be noted in this thesis that the author lectures in computer programming to first year students, using a similar syllabus to that described in table 4.3. As it was unethical for him to use his own classes in data collection, classes at sites with which he has no link were used. Given the similarity of language and syllabus with that of his own professional work, theoretical sensitivity was greatly enhanced. This is consistent with the recommendation of Seidman (1998) where he
indicates that ‘the teacher-researcher should seek to interview students in some other setting with some other teacher with a similar method or curriculum’ (p.35).

4.7 CONTEXT

Practical planning and engagement in data collection required the author to take the context of the research question as an important issue. In fact Strauss and Corbin (1998) explicitly state that ‘by context we mean the conditional background or situation in which the event is embedded’ (p.106). In this project the context of the study was first year programming students in Irish third-level institutions. It has been found that the primary issue in this research relating to context pertained to data collection. In particular, the nature of the participants, their location and logistical constraints are contextual issues that cannot be ignored by the researcher and must be embraced and dealt with as effectively as possible.

One example of context in this research pertained to the nature of the participants, who were primarily school leavers for whom the open-ended interview they undertook may have been their first ever interview. In this regard, great care was taken to make the interviewing environment as non-intimidating as possible. For example, a deliberate decision was taken to visit each research site in person to present the nature of the project to the students. This is recommended by Seidman (1998) where he states that ‘building the interviewing relationship begins the moment the potential participant hears of the study’ (p.39). It was found that this approach had positive implications for the project. In specific terms, students who volunteered to take part in the study were those who felt comfortable with the researcher and the nature of data collection in the project as it was described to them in the first contact session. It was also found that this comfortableness extended to the interview and focus-group sessions where the purpose and the ‘setting at ease’ issues were reiterated to the participant. Deliberate action was taken to inform the participant that the interview questions were not being framed as knowledge-level acquisition questions but rather were borne out of a genuine need to explore the participants’ own personal experiences of computer programming. Furthermore, given the age and background of most of the participants (i.e. the vast majority were school leavers just
out of secondary school) deliberate action was taken to ensure that potential off-putting activities like exhibiting an expert level of computer programming, or reacting incredulously to certain answers, were avoided. In addition, on the odd occasion where participants were struggling to remember a particular term e.g. ‘index positions’ in an array data structure, the author occasionally prompted the term to reduce any discomfort they exhibited. This had the effect of making the remainder of the interview more comfortable for the participant and reduced their anxiety in terms using incorrect terminology. The author was overtly aware of this given the considerable amount of domain-specific programming terminology that a student is exposed to on a typical first year programming course.

4.8 THE INTERVIEWING PROCESS

The interviewing process was similar for both the one-on-one interviews and the focus group interviews. Hanson and Kautz (2005) recommend the first minutes of the interview be used to make the atmosphere more congenial. Every attempt was made to make the participant comfortable by assuring them that any answers or comments they gave were useful for the research and would be kept confidential. At the beginning of each interview the author spent a few minutes describing the process and allowed the student to read and sign the consent form (illustrated in appendix 1). Furthermore, in a further attempt to make the atmosphere relaxed and conducive to successful interviewing it was stressed to the student that the purpose of the interview was not to find out how much programming they actually learned but rather its aim was to ascertain how they personally experienced the programming subject. The interview durations varied in length amongst participants with the average interview time being approximately 19 minutes. Interview time duration was constrained by room and participant availability where interview time slots were only available when participants had a free period. Every effort was made to fit in with the availability of the students and the contact lecturer at each site.

In the pilot study it was found that on some occasions, participants for whatever reason do not give elaborate answer to an open ended question. In order to deal with
this situation a directed conversation approach was taken where certain questions
would be asked if the participant was not elaborating very much to the early question
pertaining to what programming was like for them. In cases where the student did
give elaborate answers, this was followed up by relevant probes pertaining to the
answers they gave. This flexibility was very useful during interviewing. In the cases
where it was necessary, the following list of questions was posed to the participant as
part of the directed conversation:

*What has the programming course been like for you?*

*How do you find the material you have covered in programming so far this year?*

*How do you find the lectures and the labs?*

*What types of projects/assignments have you had so far this year?*  
  - How did you get on with them?

*How do you go about learning / understanding programming?*

*When you come to a new topic in programming how do approach learning and
understanding it?*

*How does this approach compare to your learning and understanding of other
subjects in first year?*

*Are there any other experiences you have had in programming that you may like to
discuss which we haven’t touched upon so far?*

Whilst these questions were used as a general guide, the author was very much aware
of the advice given to grounded theory researchers by Wimpenny and Gass (2000):

‘Although unstructured interviews may frequently, in practice have
a general guide which seeks to cover the theme to be developed in
depth, the researcher seeks to follow the major concerns or point of
view of the respondent. This approach is considered as the best means of securing the personal and private concerns of respondents'. (p. 1487)

In this regard, in situations where the participant was giving detailed personal accounts of their programming experience, follow-up questions were used to acquire more detail on the various issues recounted. The following segment illustrates this flexible interviewing approach adopted in the interviews:

Q. What has programming course been like for you?

A. Personally I found it tough...........

Q. How did you deal with that?

A. Put it to the back of my mind.......and eventually approached the lecturer about extra tutorials

Q. How did you find those tutorials?

A. I found them good..........

Q. What changed, if anything?

A. Just knuckling down..........

As can be seen from the above segment of interviewing, follow-up questions on issues raised by the participants were implemented. This was found extremely useful for two reasons. Firstly, it showed the participants that the interviewer was able (with the help of theoretical sensitivity) to pick up on issues that were important to them. Secondly, the participants, in many cases, answered their own personal experience probing questions in a more emphatic manner. It must be noted here that the author stressed to participants that all personal experiences of the student were pertinent to the research and not just a subset of them. This enabled the non-forcing of questioning and
Chapter 4

Data Collection

resulted in what the author refers to as 'natural saturation' as opposed to 'forced saturation'. This issue will be discussed further under methodological memoing.

Furthermore, grounded theorists (Strauss and Corbin, 1998; Goulding, 2002) indicate to the researcher that the general list of open-ended questions may be changed in subsequent interviews. This is consistent with the experiences of Hanson and Kautz (2005) who indicate that 'if we came upon new information in an interview, we added new questions to the interview guides, which other respondents subsequently could elaborate on' (p.4). This technique was used in cases where participants were slightly reluctant to recount some of their experience, especially in situations where their progression or work ethic on the subject wasn’t as high as they would have liked it. In this case the following question was added to the list of general open-ended questions:

What advice would you give to a first year student starting off on a programming course?

This question was used because it allowed the participant to speak in the third person and focus on issues that they would have avoided earlier. In many cases they would highlight behaviour that was inadvisable. A segment of data pertaining to this question is as follows:

A. Go in with and open mind and don’t miss lectures. Some people go in and see the course content and they get absolutely terrified........

A. Pay attention right from the start, because it goes off really quickly. You think you are on top of it and all of a sudden you are saying what is going on ........

In terms of uncovering theory, it was deemed necessary in interviewing participants to take handwritten notes as well as taping the interview. It was necessary to note any important issues the participant raised to that they could be revisited subsequently in the interview for further exploration. This is consistent with the interview techniques espoused by Seidman (1998) where he recommends follow-up questions as opposed to interrupting a participant in the full flow of answering a question. Furthermore, a small unobtrusive electronic voice recorder was used to record the interviews and
participants were given the option not to have this utilised. Furthermore, participants were reassured that the interviews were confidential with the researcher being the only one who would have access to the recordings. Given this reassurance, all participants elected to have the interview and focus group sessions recorded.

The focus group sessions were approached in the same manner as the open-ended interviews. Two sessions were undertaken at two separate sites. The first session entailed seven participants and the session lasted just under an hour. The second session entailed three participants and lasted 45 minutes. Given the group nature of these sessions, it was found that only one question needed to be asked as the beginning of the session:

*What has the programming course been like?*

It was found that this general open ended question opened up many other issues and experiences that were followed up with other questions to enable elaboration. Care was taken to enable, in as far as possible, each participant to voice their experience on an issue. This was necessary as some participants allowed a subset of the group to bring up an issue and they would then come in and voice their experience if they felt comfortable.

As can be seen in table 4.2, only three participants filled in the journal/diaries. This reinforced the importance of personal contact as described earlier. Whilst e-mail contact may be effective to an extent it was found to be in no way as effective as personal contact with students. Despite all of this, any data collected from participants was relevant to the study and journals did enable participants to describe the technical difficulties they were having with programming concepts.
4.9 PARTICIPANT TERMINOLOGY

According to Maykut and Morehouse (1994) ‘the data of qualitative inquiry is most often people’s words and actions and thus required methods that allow the researcher to capture language and behaviour’ (p.46). During the study this issue became very apparent. In particular, when describing their programming-specific experiences it became apparent that terminology and approaches become engrained within a class. For example, in talking about the array data structure in computer programming, participants from one group referred to ‘cells’ whilst others referred to ‘index positions’ or ‘locations’. This reinforced the need for theoretical sensitivity on behalf of the researcher so that concept-specific follow-up questions or probes could be asked in an informed yet unbiased manner to achieve theoretical saturation.

In addition, as well as being theoretically sensitive to the subject matter i.e. computer programming, it was also essential that the author be sensitive to the semantics of the non-programming terminology used by the participants. A primary example of this pertains to the term ‘learning’. In Ireland, the term ‘learning’ is frequently referred to as ‘learning off’,\(^3\) which is an alternative term for rote learning. Here, learning refers to rote learning as opposed to understanding. For this reason, a question relating to this issue contained a reference to learning and understanding and this was clarified to the participant in instances where any doubts arose:

*How do you go about learning / understanding programming?*

Given the fact that grounded theory is a method whose data collection techniques facilitates participants recounting their experience in their own voices (Strauss and Corbin, 1998), it was important that the author not only paid attention to in-vivo terminology and theoretical saturation but also the semantics of the language of those participating across four separate data collection sites. This is consistent with the advice of Mischler (1979) who indicates that meaning should be viewed within the social context in which it occurs.

\(^3\) This is an abbreviation for “learning off by heart” an explanation of this can be found at http://idioms.thefreedictionary.com/learn%2Flearn+by+heart.
Chapter 4 Data Collection

4.10 METHODOLOGICAL MEMOS

Given the importance of unbiased, unforced data collection in a grounded theory study, the author was careful to constantly reflect on the various methods used and assess the implications these methods had on the quality of the data. In particular, two issues that were reflected upon and documented by the author were the grounded theory concept of *theoretical sensitivity* and a second issue which the author describes as *natural versus forced saturation*. Each of these will now be discussed in this section.

4.10.1 Theoretical Sensitivity

Chapter 3 describes how the grounded theory researcher must possess *theoretical sensitivity* i.e. they must be sensitive to the meanings embedded in the data within the problem/phenomenon domain. This was found to be a crucial issue in developing theory in an area that has a high level of domain-specific terminology and concepts. In this regard the following memo (based on an interview with a participant) was developed in relation to accurate extraction of meaning:

*A. Arrays were tough, but you come across everything like that.*

*Q. Was there anything in particular that made arrays difficult?*

*A. Just understanding what they did really, I understood the layout, but just understanding what they did, how they worked in code.*

*Q. How did you deal with that?*

*A. I had to ask about it. In the lecture I didn't understand it, so I just asked about it, its just held for storing memory and all that, it's an easier method, saves a lot of typing out.*
Chapter 4 Data Collection

Memo – This extract from interviewing is a good example of the theoretical sensitivity required of the researcher. In the last answer, the participant attempts to describe how the array data structure works and why it is used. Even though the description is slightly inaccurate, my theoretical sensitivity from working as a lecturer/tutor enables me to understand the message the student is trying to convey. For example, by saying ‘it saves a lot of typing out’ the student is referring to the reduction in code duplication that can be achieved by using an array where you can declare an array of 10 integers instead of declaring 10 separate integer variables.

In certain instances understanding responses like this enables the researcher to ask further questions that are relevant to the answer just given. In the data collection in this project, the participants, even though sometimes struggling with trying to get a coding concept across (because they are not 100% confident of their understanding of the concept), often appreciate the fact that you have understood what they are trying to say and therefore become more open to elaborating on further probing type questions. It is important to re-iterate here that in advance of the interviews, participants were explicitly informed that the interview was not an assessment of their theoretical knowledge of programming, but rather how they personally experienced the programming course.

The development of memos like this in terms of generating accurate theory that was grounded in the actual data was viewed to be crucial in this research project. Furthermore, methodological memos of this nature must be written down immediately by the researcher as soon as they occur, otherwise the rich methodological issues that arise when immersed in collection and analysis may lose their precision or incisiveness.

4.10.2 Forced Versus Natural Saturation

As highlighted in chapter 3 the major criticisms by Glaser (1992) of the Strauss and Corbin (1998) method pertain to contextual description and theory forcing. Given the generic requirement of inductive theory generation that transcends both Glaser and Strauss and Corbin methods it seemed clear that theory forcing would fly in the face
of inductive research. In fact, whilst Glaser (1992) is vigorous in his desire to avoid forcing, Glaser and Strauss (1992) also indicate that the central category 'should be logical and consistent, there is no forcing of data' (p.147). The following methodological memo relates to the author's experience on what he refers to as natural versus forced theory saturation.

Memo – Across my four data collection sites I am persisting with the open sampling approach as prescribed by Strauss and Corbin and as I gather data and analyse it I find it results in what I call 'natural' saturation of categories, i.e. those that emerged again and again across the various sites in an unbiased, unforced manner. It was decided that to proceed with other theoretical sampling strategies as described by Locke (2001) (on pages 81-82) would have two undesirable consequences. Firstly, it might have the effect of using categories uncovered in the first cluster of interviews as the key indicators of emergent theory for further saturation even. This may result in the danger of the initial phenomena being local to the initial site and may never reach saturation (given their context specificity to site one) and implementing a dangerous strategy of chronological ordering of research sites as drivers for a direction to go in. Secondly, seeking out participants in a purposeful way by attributing them with some sort of preconceived label such as 'struggling' or 'weak' in a population of participants for whom the majority are just out of secondary school may have undesirable consequences. In particular it may have a negative effect of compounding the difficulties some students are experiencing if they are being interviewed under the umbrella of some label or category. In addition this may have a negative effect on participants in terms of potentially reinforcing their personal worries about their limited programming ability as well as having an adverse effect on their other non-programming subjects.

Using a forced approach to theoretical saturation, the analyst might only look in specific places for what he/she considers to be theoretically rich sites. This type of predetermination is analogous to what Curtis et al. (1987) describe as looking under the lamppost:
Chapter 4

Data Collection

‘Everyone knows the old story about the man who lost his wallet across the street, but searches for it over here under the lamppost because the light is better’. (p.96)

Given the above memo and in terms of proper ethical research, it was found whilst conducting this study that contextual conditions must be taken into consideration with the educational welfare of voluntary participants being paramount, and vastly outweighing any other research aims. This decision was taken given the author’s eleven years of professional experience teaching programming to first year students and his overt familiarity of the ‘fragile’ (Perkins and Martin, 1986) nature of learning in the programming subject encountered in lecture, laboratory and tutorial classes as well as discussions with students. The author has found that if not dealt with sensitively it can have adverse effects on the student’s confidence.

Given these unavoidable contextual and ethical constraints, it was decided to persist with the open sampling strategy and this resulted in natural category saturation. Theoretical sampling requires the researcher to seek out participants who are information rich in terms of the research question. Given the open and non-preconceived nature of the research question, the use of voluntary participants was found to fit the criteria of theoretical sampling. The only requirement of sampling in this study was that the student was present on a first year programming course. Any experience emanating from this was pertinent to the research. Whilst some issues may have emerged that didn’t reach saturation point, only those that did to the author’s satisfaction were reported. Using this strategy, the order in which the sites were visited did not matter as all participants with experience of programming were deemed to be suitable for the research. In this regard, the theory that emerged emanated out of natural saturation (by asking open-ended questions to all participants) as opposed to forced saturation using initial issues that emerged as the drivers for future sampling (which may be a consequence of the random chronological ordering of the data collection sites). It should also be noted here that asking additional questions did not change this situation. In particular, additional questions asked in this study did not pertain to any specific emergent category, but rather were phrased in an open-ended way e.g. ‘what advice would you give to a first year programming student just starting off?’. Here, one can see that nature of the question is to elicit more data
on the participant’s experience rather than forcefully saturate a category. For example, asking the participant ‘do you find it difficult to find clear programming code examples?’ would be an example of an additional question that forces the saturation of the emerging categories ‘difficulty’, ‘code examples’ and ‘clarifying examples’ (these codes will be discussed in detail in chapter 6).

Furthermore, in terms of Glaser’s (1992) criticism of theory forcing and the paradigm model, whilst engaging in interviewing the author was conscious of avoiding the forcing of properties and dimensions, where the researcher might be tempted to deliberately seek them out. This situation is analogous to analysing a narrative in entity-relationship modelling where it is up to the analyst to identify the attributes of an entity (e.g. the credit limit attribute of the customer entity) where they are described either implicitly or explicitly in the data. This finding is consistent with the advice given in Deakin (2004) where it is indicated that ‘the interviewer must be careful not to inadvertently instil an attitude about something where none may have existed’ (p.7).

Finally, this strategy of open sampling and natural theory saturation proved to be wise in hindsight where the study found that students oscillate between different states of mind and attitudes during their first year programming course e.g. from struggling (negative) to progressing (positive) and vice-versa. Interviewing them under a preconceived label (e.g. only those experiencing difficulty exhibiting extreme weakness) might have had the impact of reinforcing a negative state of mind. In particular, given the fragility of programming learning as described in chapter 2, reinforcing any negative emotion (e.g. interviewing them as a student falling into the ‘weak’ category based on their past exam results) might have had the unfortunate consequence of the participant accepting difficulty or weakness as a category of learner type from which they would never emerge. In addition, this pre-determined labelling of students in advance of discussing their experiences with them violates inductive enquiry where the theory should emerge naturally and not be looked for in assumed or preconceived locations.
4.11 MULTIPLE METHODS AND TRIANGULATION

Various commentators recommend to researchers to avail of multiple data collection methods and sites where possible. As Raftopoulos (2005) highlights:

‘In order to assure the quality of our qualitative research we used triangulation (in-depth interviews, focus group and direct observation). It is a validity method that compares the results from either two or more different methods of data collection, or two or more sources. It may therefore be better to be seen as a way of ensuring comprehensiveness and encouraging a more reflexive analysis of the data as only one pure test of validity. This is a genuine test of validity as any weakness in one method will be compensated by strengths in another’. (p.2)

This view is echoed by Wimpenny and Gass (2000) where they indicate that ‘the need to elicit and illuminate the social situation often requires diversity in data collection methods, ensuring that theory is grounded in the data’ (p.1491).

One of the keys to success in a grounded theory study is the gathering of theoretically rich data. The primary purpose of multiple data collection methods is to provide participants with a choice of method that they are comfortable with. The author has considerable experience of the Irish educational system and the cultural backgrounds of first year students. The author’s practical professional experience suggests that many students (especially the students experiencing problems with programming) are often reluctant to discuss their issues openly, and therefore journals may be more appealing to them. The second purpose of multiple data collection facilitates the process of triangulation. Here, category existence in multiple data collection methods can serve to further validate the theory generated in the study (Anderson 1998).
Chapter 4  

4.12 CONCLUSION

This chapter has described in detail the data collection process used in this study. In particular, it has shown how interviews, focus groups and process journals/diaries were used. It has shown that proper preparation in terms of access and meeting prospective participants on a face-to-face basis is an integral part of recruiting participants on a voluntary basis. In addition, it has highlighted the logistical constraints that existed in this project and the steps undertaken to minimise their effect. It has provided the reader with a description of the nature of the programming courses across the 4 data collection sites.

In terms of the grounded theory process it has shown how methodological memos proved to be an integral part of the data collection and analysis process. In particular, it has shown how and how theoretical sensitivity is vital in understanding clearly participant terminology within the context-specific domain of computer programming. Furthermore, it has described the notion of natural versus forced theory saturation whereby the open sampling technique was used in a way that allowed theory to emerge naturally from the data from information-rich participants across the various data collection sites. Finally it has indicated the advantages of using multiple data collection methods in a study of this nature.
Chapter 5

Analysis Process

- Introduction
- Open Coding
- Axial Coding
- Selective Coding
- Summary

5.1 INTRODUCTION

This chapter describes in detail the analysis process conducted in this study. In particular, it will discuss in detail the various levels of coding and ancillary activities that were conducted during data analysis. Firstly, it will describe the open coding process where data is analysed for the existence of codes (as described in chapter 3). Secondly, it will describe the process of axial coding where properties and dimensions of the various categories were developed, as well the development of higher order categories known as *phenomena*. It will show how this process was aided by a technique developed by the author which is referred to as *conglomeration*. Thirdly, it will highlight the approach taken to selective coding that was aided by the development of another technique by the author known as *association diagramming*. Finally, it will describe how all three stages of data analysis and coding were aided by developing a technique known as *methodological memoing*. 
5.2 OPEN CODING

The first step undertaken in open coding was microanalysis. As described in chapter 3, this entails a line by line analysis of the data to generate initial categories (called codes), their properties and dimensions and the relationship amongst these categories. In advance of engaging in this activity a detailed re-examination of the Strauss and Corbin (1998) methodology was undertaken. In terms of microanalysis they draw the researcher’s attention to three significant aspects of analysis. Firstly, the data, which in this research was audio taped data of interviews and focus groups as well as programming journals/diaries. Secondly, the researcher’s interpretation of that data, which will inevitably have an element of subjectivity but hopefully, will be insightful and assisted by theoretical sensitivity. Thirdly, the interplay between the researcher and the data where he/she is constantly reacting to, and working with the data as new categories and variations of categories emerge using the grounded theory concept of constant comparison where new instances of codes are compared to those already existent in the data for similarities and variation. In terms of the latter two issues, whilst total objectivity is difficult to achieve, rigorous adherence to inductive theory generation combined with research and professional experience can be used to enhance the creative aspects of the analysis. In particular, this was found to be the case where analysis of the data was aided by the author’s eleven years practical experience in teaching computer programming to first year students. However care was taken to frequently stand back from the data in order to ask the simple question ‘what do we have here?’ and this significantly aided the avoidance of subjectivity and theory forcing. According to Strauss and Corbin (1998) microanalysis conducted in a proper way, enables a number of important functions in data analysis (p. 65):

1. It is a focussed process where the researcher is required to consider the full range of plausibility and not to take one stance or another towards the data. In this regard, they do not prematurely commit to one plausible version of events e.g. that computer programming is too difficult for first year students with no prior experience of it.
2. It obliges the researcher to examine the specifics of the data where they are not only looking for new categories but variations in terms of properties and dimensions of existing ones. For example, terms like 'sometimes', 'always' 'ages' used with respect to categories facilitate developing variations on properties e.g. 'extent' or 'duration' as illustrated in the conglomeration document of appendix 2.

3. It compels the researcher to pay close attention to what the participants are saying and how they are saying it. In this way they point out the analyst can attempt to understand how the participants are interpreting certain events. They also state that if the analysis is fortunate he/she will be provided with in-vivo concepts by the participants that are useful in stimulating analysis.

4. It requires the analyst to constantly ask himself/herself questions about the data. This constant analysis of the data may help the generation of better interview questions in subsequent interviews e.g. specific questions pertaining to properties and dimensions of a category e.g. 'you say it takes a long time for new concepts to sink in, is this the case for all concepts or just some, can you elaborate?'. Here the analyst can determine whether a long sink in time relates to a full or partial quantity of the material on the syllabus for that particular participant. Here, the category is ‘sink-in’, the property is ‘quantity’ and the dimensions range from ‘partial’ to full’.

5. It facilitates conceptualising and classifying events. Doing line-by-line analysis with a deliberate focus on identifying emerging categories and their properties and dimensions as well as relationships between categories takes the analyst beyond description into a ‘conceptual mode of analysis’ (p.66).

6. It requires the imaginative use of theoretical comparisons for raising questions about the data e.g. ‘what do we have here?’ and discovering properties of the emergent categories.
Chapter 5

Analysis Process

7. It enables the researcher to examine what assumptions about the data they are taking for granted. Comparing one’s assumptions against the data in a directed way ‘brings those assumptions to the surface’ (p.68). In addition, ‘false assumptions will not stand up when rigorously compared against the data incident by incident’ (p.68). By constantly comparing new manifestations of a concept against those already uncovered in the data, the analyst gets an accurate picture of what the data are actually saying. This may or may not concur with the analyst’s assumptions.

Furthermore, Strauss and Corbin (1998) indicate that open coding is aided by the uncovering of in-vivo concepts (i.e. where concepts are labelled using verbatim terms from the data). During the analysis in this study many of the concepts uncovered were in-vivo such as ‘sink-in’, ‘skip-over’, ‘words’, etc. and these will be elaborated upon and presented in the next chapter on data analysis.

The open coding process in this study was a time-consuming and labour intensive task. It entailed going through the data on a word by word basis with no pre­formulated theories of ideas but with an open mind. As each line of data indicated a concept it was marked using the comments function in Microsoft Word. When this was complete, a section was created for each participant and the categories that emerged in their data were put into a separate word document as illustrated in fig 5.1. As can be seen in fig 5.1 the open coding document contains comments in the margin that highlight issues of interest in the data as well as potential categories and dimensions. Furthermore, fig 5.1 illustrates a sample of the pervasive use of theoretical memos in this study. As indicated in chapter 3, these memos are crucial in the development of theory as they indicate the researcher’s thoughts at the time when they are immersed in data.
‘Interesting, it started off not that hard. I kind of got my head around it pretty fast. I didn’t have any problems, whereas now as we’re getting on to harder stuff [PD] [D] it is getting challenging (extent). I enjoy project work, especially now we are working on stuff that’s not small and stupid, which you’ll never use.’ (P36)

Memo – Here we see another dimension to the challenging code uncovered in earlier interviews. This student is finding the programming very manageable but at the same time challenging. For the better performing students it seems this challenging element is one of the things that keep the students interested in the programming subject.

As may be evident from fig 5.1 open coding is a process that is very time consuming and requires multiple iterations through the data where categories become refined as new data pertaining to them emerge. When open coding was complete, the net result was a considerable number of categories that had emerged from the data and these
were then analysed in detail again to see if any could be merged into one category. This happened on a number of occasions where 'guess' and 'trial and error' were merged into one category called 'guess' [GUESS]. In cases like this, merging was used where it was found that participants were just using slightly different terminology to articulate the same phenomenon.

Some parts of open coding were more difficult and time consuming. For example, over the course of data collection three particular concepts that emerged related to:

- The amount of time it takes programming concepts to 'sink in'.
- The notion of concepts 'clicking in'.
- The whole idea of 'getting your head around' programming and its concepts.

The dilemma here was that whilst all of these topics were interrelated, the issue was whether to group them as one single concept or present them as explicit subcategories of a higher order abstracted category (where this higher order category would be generated at the axial coding stage). In this case the key question pertained to whether merging them would dilute useful theory that could describe the programming experience at a lower macro-level. Furthermore, the theoretical question that was asked here pertained to whether or not these three separate concepts related to different types of experiences that could stand on their own with their associated properties and dimensions. The net result of this theoretical questioning was that these concepts did validly stand as three separate categories as they described different types of accommodation experience. Given this fact, they were presented as sub-category of a higher order category 'accommodation issues' at the axial coding stage. From the perspective of abstraction, identifying logical groupings and generating abstracted phenomena with suitable labels is a non-trivial process that if not done properly may result in a misrepresentation of the actual data as one moves towards the higher order categories (phenomena).
5.3 AXIAL CODING

As described in chapter 3, axial coding is a process of relating categories to their subcategories and is given the term 'axial' because coding takes place around the axis of a particular category. The overlap between open and axial coding is crucial in theory building as the analyst begins to ask theoretical questions of the data in terms of who, what, how, when and why (Strauss and Corbin, 1988). The purpose here is to conceptualise what is going on in the data by classifying events and outcomes. This is achieved by using abstraction, a technique by which the analyst who has a deep understanding of the data groups categories into higher order abstractions known as phenomena. This abstraction, refinement and data reduction is what Strauss and Corbin (1988) describe as making 'the difference between theoretical and descriptive coding' (p.66). According to Strauss and Corbin (1998) phenomena are important analytical ideas that emerge from the data and answer the simple inductive question 'what is going on here?' (p.114). The remainder of this section will describe the techniques employed during axial coding in this study.

5.3.1 Conglomeration

Another form of memoing that was developed by the author in this study is referred to as conglomeration. Using this technique, during open coding, each code as it emerged was inserted into the data in square brackets as illustrated in fig 5.1 e.g. [DISENG] was inserted anywhere disengagement was referred to in the data. Conglomeration then entailed amalgamating all references to a category in one location with each individual piece of data being followed by the participant number in brackets e.g. (P.47) from whom the data was extracted. This process entailed trawling the open coding document for all instances of the code using the find utility in Microsoft Word. This was a slow and labour-intensive task, but nevertheless proved to be very useful in coding process in terms of getting a richer picture highlighting variations in the category as required by constant comparison and category saturation. The full conglomeration document is contained in appendix 2. In the conglomeration document each category is described with an overview table followed by all instances of that category as illustrated in table 5.1.
## Chapter 5 Analysis Process

### Code: [DISENG] Long Name: Disengagement

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported by participants who consciously disengage from part or all of a programming concept. By disengage here is meant that they do not try to understand the concept/example/issue currently being covered in the programming course. As illustrated in the data this may take many forms from being a strategic decision in terms of focusing energy on other parts of the computer science course to that of a reaction to difficulty or confusion. Terms used by participants are ‘switching off’, ‘drifting’ and ‘fading’.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>Frequency</td>
<td>Rare – Often</td>
</tr>
<tr>
<td>Duration</td>
<td>Short - Long</td>
</tr>
</tbody>
</table>

**P & D Overview**

- **Level** relates to the amount of the programming material the student actually disengages from. In the data we see participants deciding to ‘pick it up again’ and putting some material to the back of their mind.
- **Frequency** relates to how often a student disengages from material. In the data we see ranges here from ‘sometimes’ to more permanent/absolute terms like ‘switching off’ for those students who find programming concepts too difficult to come to terms with.
- **Duration** pertains to how long they disengage from a particular topic, where participants mention they disengage ‘for a while’ or indefinitely by indicating getting it ‘eventually’ or ‘I’ll sort it out later’

### Associated Categories

[D] [CONF] [GH] [EXSCH]

<table>
<thead>
<tr>
<th>Table 5.1 Category Overview Table</th>
</tr>
</thead>
</table>

Table 5.1 illustrates a number of sections in a category overview table. In the first section we see the short code [DISENG] followed by the long name for that code. Long names which are taken verbatim from the text will be followed by the term ‘in-vivo’, signifying that they emanate directly from terms used by one or more participants. An example of this type of in-vivo category is ‘skip over’. The description section of the table presents a summary overview of the category based on issues that emerge in the data. This is followed by a list of properties and dimensions and their descriptive overview. Finally, the associated categories section highlights all of the other categories that this particular category was associated with in the data. For example, a student who gets overly confused [CONF] by the current
programming topic may disengage [DISENG] from it for a certain period of time. Generating this list of associated categories was slow and time-consuming and involved considerable cross-referencing with all other categories in the conglomeration document, but proved a useful tool in developing the paradigm frame for each phenomenon.

5.3.2 Properties and Dimensions
The Strauss and Corbin (1998) method also requires the researcher to develop where possible the properties and dimensions of categories, where properties represent the various characteristics of a category and dimensions representing the location of a property along a continuum or range. It should be noted that at the beginning of this research, in advance of axial coding the author was somewhat apprehensive about properties and dimensions in terms of being able to actually find them in the data. However, implementation of axial coding found that properties and their dimensions did emerge naturally and enabled the unearthing of how participants experience various phenomena differently. Furthermore this proved to be vital aid in learner classification. Whilst on the surface it may be difficult to read two chunks of data from different participants and discern between them, identification of key properties and their dimensions allows these differences to become explicit in the data and facilitate constant comparison and category saturation. Table 5.1 illustrates how the properties level, frequency and duration facilitate the researcher in differentiating between the various types of disengagement employed by participants. It should also be noted that the properties and their dimensions are highlighted in brackets within the data in the conglomeration document in appendix 2.

5.3.3 Association Diagrams and Axial Coding
As described earlier, axial coding entails generating higher order concepts by abstracting a number of related lower level categories. In this abstraction hierarchy the higher level categories are known as phenomena. Furthermore, the categories themselves may have subcategories. In grounded theory, this abstraction is shown in
the form of diagrams with related categories appearing around the axis of the abstracted category (hence the term 'axial' coding).

Whilst engaging in literature study of a vast number of grounded theory texts and journal papers, the author did not come across any form of diagramming that was informative in terms of describing the phenomena. Given this situation, and the fact that abstraction diagrams are pervasive in the computer science discipline within the realms of data, process and object modelling, the author decided to develop his own technique which is described as *category association diagramming*, which is given this name because of the way it illustrates associations between categories. Furthermore, not only are the abstraction levels and associations made clear but also they are labelled in a very precise manner as illustrated in fig 5.3 which illustrates the *progression strategies* phenomenon (which will be described in detail in the next chapter).

![Progression Strategies Diagram](image)

**Fig 5.3 Category Abstraction Using Association Diagrams**

Fig 5.3 illustrates an abstraction diagram that uses very precise terminology in terms of the labelling of the relationship between the categories and their higher order phenomenon as well as between the categories and their subcategories. For example, if we look at the relationship between the category *persistence* and the phenomenon

---

1 In fact, despite the fact that the use of diagrams is described by Strauss and Corbin (1998) as an integral part of theory generation very few diagrams illustrating open and axial coding were encountered in the literature.
progression strategies we see the association labelled as 'may involve'. This type of relationship using the term 'may' is borrowed from the computer science discipline of entity-relationship modelling where relationships between entities are labelled as optional/obligatory using the term 'must', and non-obligatory relationships are labelled as 'may'. Fig 5.3 illustrates that persistence may be used as a progression strategy. Furthermore, subcategories are labelled using the term 'is a kind of' also adopted from computer science entity and object modelling. In fact, the conceptual origins of this diagram lie in the notion of an 'association diagram' commonly used in entity-relationship modelling and class diagrams in object modelling in systems development (Brown, 2002; Hoffer et al., 2002) as illustrated in fig 5.4.

Another reason for developing this technique for grounded theory presentation is that the labelling of the diagram facilitates the creation of a storyline (a requirement of selective coding and described later in this chapter). For example, a mini storyline describing the progression strategies phenomenon might read as follows:

'Progression strategies may involve the employment of a number of strategies such as persistence, assistance (peer or lecturer) or guessing the answer. Furthermore, known bits of code may also be used in progressing at a task. This engagement in persistence
strategies of this nature may be driven by the desire for accomplishment'.

The above simple narrative illustrates that the label is changed depending on the nature of the association between the category and the phenomenon. In the above example we see the two separate labels 'may involve' and 'may be driven by'. This detailed labelling, in the author's opinion, serves to bring greater description and precision to the category association diagrams.

5.3.4 Coding Paradigm

Another integral part of the Strauss and Corbin (1998) methodology is the coding paradigm as illustrated in the paradigm frame (table 3.1 in chapter 3). This model is used to describe phenomena in terms of their context, causal conditions, intervening conditions, action strategies and the outcomes of these strategies. This enables the researcher to see how, when and why certain variations of phenomena exist and how individuals deal with these variations and the outcomes of the strategies they employ. In this project, as well as a category association diagram, a paradigm frame for each phenomenon was developed. This paradigm frame was considerably aided by both the category association diagram and the conglomerate document in appendix 2.

5.4 SELECTIVE CODING

In the detailed description of grounded theory in chapter 3 selective coding was presented as process that entails identifying a central phenomenon and relating central categories to it using statements of relationships. From the very beginning of data collection the author was conscious of unearthing a central or core category as the categories began to emerge from the data. In this regard, analysis of data entailed the documenting potential candidates for the central category. However care was taken not to prematurely commit to a central category too early in the analysis process as this might have had the effect of focussing data collection and interview questioning on that category. For example, one initial candidate for a central category was that of
'the difficulty of programming'. However subsequent data collection and analysis highlighted that whilst this was an important category experienced by a good number of participants it wasn’t the core category. Making this assumption too early in the process might have resulted in specifically seeking out only those participants experiencing difficulty and asking them specific questions relating to difficulty. This premature decision making could have resulted in theory forcing and a move away from inductive theory generation as prescribed by the grounded theory methodology.

5.5 SUMMARY

This chapter has presented an overview of the analysis process engaged upon in this study. It has shown how the Strauss and Corbin (1998) approach was implemented using their prescribed steps of open, axial and selective coding. It has described a number of techniques developed by the author to make the coding process more exhaustive and explanatory. In terms of exhaustiveness it has described the technique of **conglomeration** where all instances of codes are merged into one location with an associated category overview table that presents properties, dimensions and cross-category associations. In terms of explanation, it has shown how the **association diagramming** technique was developed to clearly present categories and subcategories as well as to specifically label the relationship between phenomena and their subcategories. Furthermore, it has indicated practical examples of each of these coding steps and techniques in order that the reader can acquire an insight into how the entire process was implemented in advance of reading the next chapter on analysis.
Chapter 6

Data Analysis

• Introduction
• Learning Theory
• Open Coding
• Axial Coding
• Selective Coding
• Learner Type Categorisation
• Validity and Reliability
• Conclusions

6.1 INTRODUCTION

This chapter provides a detailed account of the analysis conducted in this study. Firstly, it presents the reader with a synopsis of key learning theory concepts that were useful in generating abstracted phenomena. It then describes in detail the three stages of analysis engaged upon, namely open, axial and selective coding. It will illustrate the open coding concepts that emerged from exhaustive analysis and re-analysis and how they were transformed into a reduced set of categories. It will then describe in detail the abstract phenomena that emerged from axial coding as well as a synthesis of these with existing theory and literature. The final part of coding will focus on developing a central or core category and how it interacts with the major phenomena. In addition, the theory will be presented in a way that enables the
generation of learner type categorisation with each category being illustrated in terms of the major phenomena. Finally, a detailed description of how the study conforms to the evaluative requirements of validity and reliability will be presented.

6.2 LEARNING THEORY

Given the fact that the primary purpose of this thesis was to gain an insight into the learning experience of novice programming students, it was decided that in order to analyse the data correctly a clear and unambiguous understanding of key learning concepts would be required. Table 6.1 provides a synopsis of these key learning concepts as described by Woolfolk (2001).

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisation</td>
<td>The ongoing process of arranging information and experience into mental systems or categories.</td>
</tr>
<tr>
<td>Scheme</td>
<td>Mental systems or categories of perception and experience. Organised systems of actions or thought that allow one to mentally represent or think about the objects and events in one’s world.</td>
</tr>
<tr>
<td>Adaptation</td>
<td>Adjusting to ones environment using assimilation and adaptation</td>
</tr>
<tr>
<td>Assimilation</td>
<td>Fitting new information into existing schemes.</td>
</tr>
<tr>
<td>Accommodation</td>
<td>Altering existing schemes or creating new ones in response to new information.</td>
</tr>
<tr>
<td>Equilibration</td>
<td>Search for mental balance between cognitive schemes and information from the environment.</td>
</tr>
<tr>
<td>Dis-equilibration</td>
<td>An “out of balance” state that occurs when a person realises that his or her current ways of thinking are not working to solve a problem or understand a situation.</td>
</tr>
</tbody>
</table>

Table 6.1 Learning Concepts

The analysis process of this study was considerably aided by identification of these learning concepts within the data. Whilst being mindful of the fact that avoidance of theory forcing was necessary (where one doesn’t look for the existence of concepts but rather lets them emerge), given that this is a study of learning, it was felt that a deep understanding of learning theory concepts was important when they emerged in
the data. Whilst the author was theoretically sensitive to programming concepts, it was decided that in order to complete a study on learning, this must be complemented by an equal level of sensitivity in terms of learning concepts. These concepts as they emerged will be discussed during the axial coding section of this chapter.

6.3 OPEN CODING

As described in chapter 5, open coding was a slow, labour-intensive process that entailed multiple passes through the data. The first step entailed microanalysis of each interview transcript on a line-by-line basis. The implementation of this technique involved keeping an open mind with any issues that came to light being inserted in as comments in the margin of the open coding document (as was highlighted in fig 5.1 in chapter 5). Using microanalysis, when a new or existing code was encountered it was inserted into the text in square brackets. The net result of this process was a large document that listed each of the codes that were encountered for each participant as well as accompanying memos highlighting theoretical issues that occurred during immersion in the data. Fig 6.1 illustrates a portion of this open coding process for an individual participant.
Pressure Build Up [PB] + [ALC]

‘As I said programming was completely alien to me. I’ve a better understanding of it now. I’ve been keeping on top of things and not letting them build up, that’s the biggest thing really’ (P8)

Knuckle Down [KD] In-Vivo

‘If I don’t understand, I knuckle down and don’t move on until that’s clarified, not to skip over [SO] stuff that I only half understand because it will come back to haunt you’ (P8)

Accomplishment [ACC]

‘It takes a good bit of work for some programming [SI], you get stumped at them, but you just keep at them [P] then when you get something displaying to the screen that’s supposed to be there and you are putting maybe 20 minutes or more and then you get your goal, you get a good feeling’ (P8)

Memo – In relation to my earlier memo, here we see the participant alternating between states of letting things consciously go over their head and knuckling down and not consciously skipping over material that they are not getting first time. It will be interesting to find if future participants drift between different strategies/approaches or mental states.

Memo – When asked about any other experiences they have had this participant gives an answer in the form of advice, i.e. things not to do when you are a novice programming student, it will be interesting to see if this emerges in future interviews.

Fig 6.1 Open Coding Example

Whilst engaging in the open coding process, no codes were ruled in or out. When engaging in conglomeration it was found that some data reduction took place in the form of combining two or more codes into categories (e.g. where categories like ‘trial and error’ and ‘guess’ merged) as well as the relegation of some categories to action strategies or outcomes as represented in the coding paradigm. An example of this would be ‘knuckle down’ which was looking, on the first pass through the data, as a potential code but given its isolation was deemed to be an action strategy in a more saturated code (in this case it was an action strategy in the ‘skip over’ code). This process of numerous passes through the data, using the conglomeration technique (as
Chapter 6 Data Analysis

described in chapter 5) resulted in the final list of categories and is illustrated in table 6.2.

<table>
<thead>
<tr>
<th>Code</th>
<th>Full Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ACC]</td>
<td>Accomplishment</td>
</tr>
<tr>
<td>[ALC]</td>
<td>Alien Concepts</td>
</tr>
<tr>
<td>[AOO]</td>
<td>Awareness of Others</td>
</tr>
<tr>
<td>[BW]</td>
<td>Bewilderment</td>
</tr>
<tr>
<td>[CHALL]</td>
<td>Challenging</td>
</tr>
<tr>
<td>[CLAREX]</td>
<td>Clarifying Examples</td>
</tr>
<tr>
<td>[CLK]</td>
<td>Click-In</td>
</tr>
<tr>
<td>[CODEX]</td>
<td>Code Examples</td>
</tr>
<tr>
<td>[CONF]</td>
<td>Confusing</td>
</tr>
<tr>
<td>[D]</td>
<td>Difficult</td>
</tr>
<tr>
<td>[DISENG]</td>
<td>Disengaged</td>
</tr>
<tr>
<td>[ENJOY]</td>
<td>Enjoyable</td>
</tr>
<tr>
<td>[EXSCH]</td>
<td>Existing Scheme</td>
</tr>
<tr>
<td>[GH]</td>
<td>Get Head Around It</td>
</tr>
<tr>
<td>[GUESS]</td>
<td>Guess</td>
</tr>
<tr>
<td>[IE]</td>
<td>Ineffective Example</td>
</tr>
<tr>
<td>[INTIM]</td>
<td>Intimidating</td>
</tr>
<tr>
<td>[ISO]</td>
<td>Isolation</td>
</tr>
<tr>
<td>[KB]</td>
<td>Known Bits</td>
</tr>
<tr>
<td>[LA]</td>
<td>Lecturer Assistance</td>
</tr>
<tr>
<td>[LEC]</td>
<td>Lecture Notes</td>
</tr>
<tr>
<td>[MD]</td>
<td>Material Delivery</td>
</tr>
<tr>
<td>[P]</td>
<td>Persistence</td>
</tr>
<tr>
<td>[PA]</td>
<td>Peer Assistance</td>
</tr>
<tr>
<td>[PB]</td>
<td>Pressure Build-Up</td>
</tr>
<tr>
<td>[PK]</td>
<td>Prior Knowledge</td>
</tr>
<tr>
<td>[SE]</td>
<td>Self Example</td>
</tr>
<tr>
<td>[SI]</td>
<td>Sink-In</td>
</tr>
<tr>
<td>[SLFDBT]</td>
<td>Self Doubt</td>
</tr>
<tr>
<td>[SO]</td>
<td>Skip Over</td>
</tr>
<tr>
<td>[W]</td>
<td>Words</td>
</tr>
</tbody>
</table>

Table 6.2 Open Coding Categories

6.4 AXIAL CODING

As described in previous chapters, axial coding was implemented to develop the properties and dimensions of the various categories uncovered in open coding. Furthermore, this coding process conceptualises by facilitating the abstraction of
related categories into logical groupings known as phenomena. The techniques involved in this process entailed utilising the paradigm model and conglomeration. The remainder of this section provides a detailed analysis of the various phenomena uncovered in the data after many hours of open coding and conglomeration.

6.4.1 Progression Strategies

This abstracted category (phenomenon) encapsulates a number of positive undertakings by the student who is trying to progress in a given programming task and is illustrated in fig 6.2. A detailed analysis of this phenomenon in the data indicates that a student engaging in this positive strategy utilises one or more activities to understand a given programming task or concept.

As indicated in appendix 2 a given student may engage in persistence [P] to various extents, whereas some may stick to lecture notes examples others may exhaust all means such as internet examples, books, peer and lecturer assistance etc. Depending on the type or category of student learner type we are dealing with the duration of this persistence may very considerably e.g. in the data we see participants recounting ‘staring at it for an hour’ and trying out things ‘over and over’. Furthermore, the nature of this persistence can be random in the form of guessing or trial and error or planned in the form of having a pre-defined sequence of tasks that are undertaken. For example, the student may begin by implementing all of the code sections (e.g. keyboard input/output of memory variables) of a task that are known to them (known bits [KB]). These simpler tasks may be completed in order to attain some level of accomplishment [ACC] that may give the student the confidence and self belief to persist further with the more demanding tasks. Therefore, we see a complex set of interactions between the associated sub-categories of this phenomenon. The interaction between these sub-categories may differ for each learner type and each task undertaken. Whilst fig 6.2 suggests a hierarchical structure (a technique used in grounded theory to represent categories and sub categories) it should be noted that the data suggests progression strategies may entail a complex network of interacting subcategories.
Chapter 6 Data Analysis

This intricate network of interacting categories was evident in the data where we see an example of interaction between three categories, notably code examples [CODEX], persistence [P] and peer assistance [PA]:

'We have examples of applets [CODEX] using arrays and not applications, I most likely implement the code from the applet into the application and try and modify it till it works [P]. If that didn't work I'd turn to people in my class [PA] (extent)' (P3)

In this piece of data from interview transcripts one can see a number of issues at play. Firstly, we see an interaction between two subcategories of the progression strategies phenomenon namely persistence [P] and peer assistance [PA]. Secondly, we see an interaction with a category outside of this abstraction, namely code examples [CODEX]. This suggests an even more complex relationship where interaction between them transcends phenomena as illustrated in figure (fig 6.3).
The paradigm frame for this phenomenon is illustrated in table 6.3. In terms of *causal conditions* the student may find themselves presented with a task which may require them to implement one or more strategies to complete it. This task could be for example, extracting a surname from a string field. The student may make a conscious decision to implement one or more progression strategies. In terms of *context*, this task, depending on its current and future importance may determine whether or not the student progresses satisfactorily in project and laboratory work. For instance, progressing at methods/functions (by being able to pass parameters and return values as one participant put it) may be seen as of significantly high importance for progression and is illustrated in the data:

‘If statements and for loops, I got to grips with fairly easily but the stuff we are doing now with the public and private methods. I find that a bit hard [D] It’s easier to understand the “for” and “ifs” because you can put in what they are doing. But the methods, you have to think about it, they are more complex. I’d like a few more examples’ (P21)

‘When I start new concepts like methods, constructors, classes, inheritance etc, all of like in 2 weeks or something, so much to take in and try to put them together’ (P19)

‘Functions took me a long time to get the hang of and pointers did too. They were the two biggest ones. Everything else I got fairly
OK but with those two in particular, if you miss something at the beginning, it is very hard to catch up’ (P43)

Given this context, there may also exist a range of intervening conditions where students may have been given code examples [CODEX] as part of their lecture notes [LEC]. As highlighted in their properties and dimensions these code examples may vary in their size, ease of understanding and explanatory power.

Depending on these dimension levels the student may decide on which particular action strategies to undertake. This may entail a number of actions which may be self-based, peer/lecturer based, or a combination of both. For example:

'Just knuckling down and going back over the notes [LEC] as much as I could [P] and asking how other people did things [PA]' (P8)

'It means you are reading back through notes [LEC] more to find answers to your questions, stuff that you could have missed you’re now picking up on’ (P8)

Self-based strategies may entail planned approaches where they start off the program by putting in all the known bits [KB] of code they are actually sure of. The data also indicates that this may have varying levels of effectiveness:

'I’d start off writing the bits I did know, I’d troubleshoot, I’d try and troubleshoot it to see which part was wrong’ (P1)

'You do the parts you can do first and then you look at the rest’ (P9)

'By taking a bit at a time, when we have projects I get that bit working then the next bit working. I end up with five or six different functions and I have to put them all together which doesn’t work very well (level), but I get there’ (P42)
Eckerdal and Berglund (2005) refer to this practice as writing 'short pieces of programs, “program chunks”, as characterising what learning means’ (p.137). Another self-based action strategy is that of persistence [P] which as suggested in appendix 2 varies on its extent, duration and type. These properties indicate that students may go to different lengths, persist for varying durations, and do so in either a random or planned basis. Bruce et al. (2004) also uncover this strategy and describe it as ‘perseverance and many hours spent in front of the computer feature in the behaviour of a student who goes about learning to program’ (p.150). The varying degrees of persistence may depend on many factors such as enjoying [ENJOY] programming (despite its level of difficulty [D]) and the sense of accomplishment [ACC] when persistence pays off:

‘At the start it was OK but then it got harder, but the fact that I liked it [ENJOY] and kept at, it’s OK again’ (P22)

‘When you’re actually successful at doing a hard program, you’re quite relieved and you just love [ENJOY] it because it took so much effort (effort) to get it working. If you are not interested (interest) you won’t like it’ (P1)

Another self-based action strategy relates to guessing [GUESS] pieces of code in order to complete a task. This strategy was also reported in the grounded theory study of Fitzgerald et al. (2005):

Perhaps the most amusing strategy is guessing. Subjects employ guessing when they have no idea what the correct answer is (pure guessing) and sometimes after reducing the number of viable choices through elimination (educated guessing). (p.75)

This approach is also described by Perkins et al. (1989) as ‘tinkering’:

Some students often program by means of an approach we call tinkering – they try to solve a programming problem by writing some code and then making small changes in the hopes of getting
it to work. In some cases this strategy can be effective, while at other times it interferes with students’ progress in solving programming problems. (p.272)

The data in this study indicates that nature of guessing may be random or planned (which appears to correlate with pure and educated guessing as mentioned above). In the data this guessing was also referred to as ‘trial and error’ and is consistent with the findings of Bruce et al. (2004) who indicates ‘they [students] seek examples in texts on the internet and from other sources, trial and error approaches to inputting code may be adopted’ (p.149). In terms of the different categories of learner types, whilst the data indicates that both weaker and better performing students may guess at various stages in a task, it was found that the better performing students may retrospectively analyse the code with the aim of understanding why the guessed code works. Examples of this in the data make for interesting reading:

‘In some of my programs I guessed around it and got it to work and then I had to sit back and print it off, read it and go back and finally figure out how I made it work’ (P42)

‘If I get it to work I usually try to test it to make sure it’s not just a fluke (retrospective assessment). Sometimes I might not get a full grasp of how exactly everything works but I’ll understand how to use if and after a couple of different programs using the same thing differently I’ll understand it in time eventually[SI] (planned)’ (P36)

Bruce et al. (2004) also noticed this diversity in guessing (trial and error) and described the higher performing students as those with a ‘focus on understanding’:

Some students with a focus on understanding as a fundamental part of learning may feel that coding or adopting a trial and error approach is their only means of achieving that desired understanding. (p.151)
This finding is similar to that of Linn & Dalbey (1989) who indicate that weaker novices spend little time testing code, and tend to attempt small fixes rather than significantly reformulating programs. Robins et al. (2003) when discussing the work of Perkins et al. (1989) highlight that "extreme movers, "tinkers", who are not able to trace/track their program, can be making changes more or less at random, and like stoppers have little effective chance of progressing" (p.155). Furthermore, depending on how well they are progressing the student may seek assistance from either their peers [PA] or from their lecturer [LA] which entails some form of interaction. Bruce et al. (2004) also found this where they indicate that 'input from experts is sought and intensive direction from teachers is expected' (p.149). As appendix 2 indicates lecturer assistance [LA] may range from full to partial, where they may give suggestions in a scaffolding manner to assist the student in completing the task on a bit-by-bit basis. Here, scaffolding is defined as 'giving information, prompts, reminders, and encouragement at the right time and in the right amounts' (Woolfolk, 2001, p.49) and as such, can gradually empower the student to do more and more on their own. Peer assistance [PA] may either fully or partially explain or solve a problem as well as being either reciprocal or one-way. The data indicates that reciprocal assistance may entail a group of students working together in a mutually supportive fashion:

'We sit down and work on it, that helps. You're helping them and they're helping you and once you get it done it clicks in my head [CLK]' (P19)

'If you work with other people, you get kind of get their ideas as well as especially if you think to yourself "I'm lost". It's good to get people working together, by listening to others and what they are saying, it kind of sinks in with yourself [SI], you kind of learn it from each other (reciprocal)' (P9)

'We get a group of friends together and bounce ideas off each other as to how the theory is best applied' (P37)
One-way assistance entails a student asking a classmate a specific question on a particular topic, task or concept with no reciprocation. Furthermore, analysis of the data suggests that there seems to be a pretty even dispersal of both types of peer assistance, and this is consistent with the author’s professional experience of supervising computer programming laboratory sessions. Robins et al. (2003) draw the researcher’s attention to the merits of peer collaboration and indicate that ‘in the context of practical tasks, paired or collaborative work and “peer learning”’ has also been shown to be beneficial’. (p.158). Webb and Lewis (1989) refer to this as ‘peer interaction’ and indicate that it is useful strategy for a student by providing a means to ‘evaluate their own and others’ answers, ideas and opinions, confront their own misunderstandings and a lack of knowledge and as a consequence, restructure their own thinking’ (p.181).

In terms of outcomes/consequences, these can depend on the various combination of strategies employed. When a certain strategy is employed e.g. persistence [P] the net outcome may result in the task being achieved or the concept being understood. This is evident in the data:

‘Just sitting it out. Working it over and over until it clicks [CLK].
There is a moment when it just clicks and you understand it’ (P29)

‘Try the example until you get it working [P]. If you get the example you’ve got it nailed down’ (P37)

‘I use hard graft, try to find different ways of doing it. Usually you get a way of doing it that does work’ (P39)

In the data above we see a number of positive outcomes where the material eventually ‘clicks’ [CLK] or is ‘nailed down’. Another interesting outcome that was consistent with the author’s professional teaching experience in the programming laboratories was as follows:

‘The more you work on them the more you find how similar they are, it’s only a few lines of code that are different’ (P3)
In this interview the participant recounts a common experience where they see that the solution to a given task often only involves slight changes to code they have already. The following theoretical memo in fig 6.4 extracted from appendix 2 describes how this was viewed during data analysis.

**Memo** – This is a situation where students who initially find something difficult, realize when they persist and get there that it actually wasn’t a major departure from what they would have done already. This is consistent with my experience of dealing with students in laboratories where in some cases when they get something working e.g. extracting a surname from a string, they almost kick themselves for not seeing the solution earlier (which involves changing a parameter or two in a method call in an example where they have already been shown how to extract the first name). This is also matched with a sense of happiness and achievement when it does work especially when the ‘eureka’ moment is encountered.

A common outcome/consequence that seems to transcend the learner categories from weak to highly progressing is that of *accomplishment* [ACC]. In the data the participants indicate that the intensity of this accomplishment is dependent on a number of factors (as indicated in the list of properties) such as amount of interest the student has in completing the task, the amount of effort expended, and the amount of time spent on completing the task. In this regard, the effort put into the other strategies like persistence [P] and implementing known chunks of code [KB] may be driven by this desire for accomplishment [ACC]. This is represented by the relationship between the phenomenon ‘progression strategies’ and ‘accomplishment’ where the relationship is labelled as ‘may be driven by’ in fig 6.2. Analysis of the data in appendix 2 will uncover jubilant outcomes such as ‘good feeling’, ‘amazing’, ‘satisfying’, and ‘proud’.

Finally, it should be also noted that the net outcomes of implementing these progression strategies are not always positive. One participant utilising the guessing strategy recounted the following:

---

130
Chapter 6

Data Analysis

'I went to the laboratory and typed up the stuff and hoped for the best, but with that you don't learn very much. For pointers I didn't get it straight away, I just type in anything that would resemble what it should and just hope for the best. I kept getting errors.' (P43)

Other participants indicated that the net outcome of this failure to get a progression strategy working successfully may be a feeling of frustration and an awareness of not being able to advance satisfactorily with the material given the building block nature of programming:

'I just do my best, just keep trying until I can't understand it, go over it, get help, do everything I can because you can't really advance if you can't understand it' (P44)

'You're doing something and when you succeed there is a reward at the end, if you get something to work it is amazing (level). If you can't get it done it's frustrating, but when you get it to work, it's amazing' (P19)

This failure to progress satisfactorily after persisting for a certain length of time or seeking assistance may result in them employing a negative avoidance strategy that will be described in the next section.

In many of the data examples highlighted in this section we see many associations between various categories with the values of the attributes or properties of one determining the value of another. For example in terms of a sample association between persistence [P] and accomplishment [ACC] one participant indicates:

'It takes a good bit of work for some programming, you get stumped at them, but you just keep at them [P] then when you get something displaying to the screen that's supposed to be there and you are putting maybe 20 minutes (time) or more and then you get your goal, you get a good feeling [ACC]' (P8)
As indicated earlier in this chapter, appendix 2 lists all of the associations between categories as indicated in the data. In order to conceptualise and understand the nature of the association between categories like this Strauss and Corbin(1998) recommend the use of ‘mini-frameworks’ that graphically depict the relationships between concepts. A mini-framework portraying the relationship between persistence [P] and accomplishment [ACC] is illustrated in fig 6.5.

![Fig 6.5 Mini-Framework](image)

The utilisation of diagrams of this nature is recommended by Strauss and Corbin (1988) when conducting grounded theory analysis when they indicate that ‘diagrams are very important devices. Their use should begin early in the analysis because they help the analyst think through possible relationships’ (p.141). They also quote Dey (1993) who asserts that ‘diagrammatic displays are not just a way of decorating our conclusions, they also provide a way of reaching them’ (p.192).

To summarise, this section has described in detail using data samples the various progression strategies employed by students. The time duration and extent to which students employ these strategies can vary considerably. Success at one or more of these strategies can instil a positive outcome such as a deep sense of accomplishment,
whereas failure can result in a sense of frustration and perhaps employment of an avoidance strategy.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description (Memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenomenon</td>
<td>Progression Strategies</td>
</tr>
<tr>
<td>Causal Conditions</td>
<td>Student encounters a concept or a task in a computer programming laboratory or classroom and decides to engage in one or more activities that may assist in their completion of this task and facilitation of understanding or accommodating the programming concept. This is seen as a positive strategy where the student is determined to progress in the relevant programming task.</td>
</tr>
<tr>
<td>Context</td>
<td>Understanding a programming concept or task like the current one under consideration is a prerequisite for progression in either the current laboratory task, project, or assignment. An example of this might be as one participant (P22) recounted to write a method that takes two array arguments and returns a Boolean as its result. The ability to write a value returning method of this nature may determine progression in current and future projects. The student wishes to persist at this task in order to complete it.</td>
</tr>
<tr>
<td>Intervening Conditions</td>
<td>Lecture materials, i.e. notes, text books, code examples will been given to the student illustrating how the concept works, accompanied by a relevant explanation by the lecturer.</td>
</tr>
<tr>
<td>Action/Interaction Strategies</td>
<td>The student engages in one, more or all of a number of strategies from persisting at the task until they succeed to seeking assistance from their peers or lecturers to guessing. These strategies may be self-based or interaction based (by seeking help of peer or lecturer).</td>
</tr>
<tr>
<td>Outcome/Consequences</td>
<td>Student either succeeds in the task using one or more of these strategies or alternatively seeks to reapply one or more of these strategies using alternative programming concept examples. The eventual completion of this task may instil a sense of accomplishment.</td>
</tr>
</tbody>
</table>

Table 6.3 Progression Strategies

6.4.2 Avoidance Strategies

In addition to dealing strategies phenomenon described in the last section, analysis of the data also uncovered a phenomenon that was labelled avoidance strategies, where in this case the students employ various methods or strategies to avoid dealing with a
task, concept or project-based issue e.g. transforming their code into value-returning methods. This data exhibits considerable overlap with the findings of the significant literature review of Robins et al. (2003) who indicate that the strategies employed by programming students are a central consideration. They go on to indicate that certain broad attitudes and conduct characterise unsuccessful novices. In particular, Perkins at al. (1989) describe 'behaviours such as stopping, neglect of close tracking, casual tinkering, and neglect of or systematic errors in breaking problems down' (p.277). Fig 6.6 illustrates that the two avoidance strategies that emerged in the data were disengagement [DISENG] and skipping over [SO].

![Avoidance Strategies Diagram]

Fig 6.6 Avoidance Strategies

The paradigm frame for this phenomenon is illustrated in table 6.4. In terms of *causal conditions* the student may encounter a concept being covered during a lecture, or a task in a laboratory as a stumbling block in a project and takes a conscious decision to skip over or disengage from working any further on it. An example of this in the data is where one participant who didn’t understand the concept of functions and told themselves that they would ‘get it all sorted out later’. Other causal conditions reported in the data relate to the large volume of material, the progressive difficulty of the programming course as well as the pace of the delivery of the material. Another causal condition that emerged in the data which hadn’t been previously been encountered in professional experience related to encountering words [W] that didn’t make sense and thus they were skipped over [SO], e.g.
'I go to the lecture notes [LEC] and everything is just big words, just jumping out at me, I don't understand it and say "OK", so I kind of skip over [SO] the big words where I know the big words actually mean something and have relation to the things I can't get' (P2)

In terms of context this phenomenon seems to manifest itself in any situation be it lecture, laboratory session or self-study at home. Therefore the engagement in this phenomenon is not the preserve of any one situation. In the data we see terms like 'drifting' and 'fading' that may be caused by lack of interaction in the classroom or a gradual loss of enthusiasm as the year progresses:

'When I came in first, you'd be kind of going "I want to do this!" like starting a new job, you want to work a lot harder and then as time goes on you get more lazy or something and say to yourself "I'll catch onto it later". So with a lot of the concepts I probably think it's a lot harder [D] than it actually is, because you are not giving it as much effort as you did at the start. When you start something new like "if" statements, they were like really easy because it was at the start and everyone is taking all the notes and then you have loops or something and everyone is not getting it because they are not up for it as much as they were' (P19)

'I feel if I can't interact in the class I drift, I start thinking about something else' (P42)

The theoretical memos provided in appendix 2 indicate that with some students there appears to be an ever-present temptation to employ avoidance strategies:

'Don't let it go by the wayside saying "I'll do it later", because you are never going to do it later because you are always progressing' (P42)
'The temptation to still overlook memory allocation is still annoying me. I want to get it nailed down. It's the only thing on the course that I couldn't get nailed down. The first few times I saw memory allocation I said I haven't a clue what this is and I'll come back to it at some stage and saying it will be alright, I'll never see it again (importance level)' (P37)

The theoretical memo relating to the above piece of data indicates that there may in some cases a be a level of self-justification at play where the student decides it's not such a big deal because they will never have to employ the concept again or alternatively as one participant put it, 'I'll catch onto it later'. This in turn brings us to the issue of the frequency and importance level of the avoided material, where the justification may lie in the fact that skipping over a less pervasive topic like memory allocation will have less disastrous consequences than assignment statements which exist in practically every program that the student will write. An example of an assignment statement in the Java language is as follows:

```java
average = (number1 + number2) / 2;
```

In this simple snippet of code above the assignment operator is the “=” symbol and the “/” symbol is used for division. In an assignment statement, the variable on the left hand side of the assignment operator (in this case the variable `average`) is given the result of the expression on the right hand side, which in this case is the average value of the variables `number1` and `number2`. The assignment statement is the most common statement in every computer program as it is the means of assigning values to variables after they are computed. The data suggests that skipping over [SO] is something that the participants appear to keep to themselves as there is no evidence of them discussing it with fellow students.

Once again, an important contextual issue is the progressive nature of computer programming where subsequent topics build on the current ones and those that have gone before it. Therefore, the level pervasiveness or importance of the skipped concept may determine the net outcome or consequences of this strategy.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description (Memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causal Conditions</td>
<td>Student encounters a problem or a stumbling block relating to an issue or a concept in computer programming and rather than tackle it further decides to avoid the issue either on a temporary or permanent basis. For the most part this is seen as a negative strategy where the student consciously avoids material, except in situations where it is part of a deliberate strategy to return to it and some point in the future when assistance of some sort becomes available or when interest and energy for tackling the topic is reinvigorated.</td>
</tr>
<tr>
<td>Context</td>
<td>The student is either in a lecture or laboratory situation and comes across a topic that doesn’t make sense to them or that they can’t engage with. The concept or issue that is skipped over or disengaged from may have varying importance given the building block nature of programming. The student may or may not be overtly aware of this. For example avoiding “if” statements may likely be more detrimental than “switch” statements in Java given the pervasiveness of the former.</td>
</tr>
<tr>
<td>Intervening Conditions</td>
<td>Lecture materials, i.e. notes, text books, code examples will been given to the student illustrating how the concept works, accompanied by a relevant explanation by the lecturer in the lecture room. The material or explanations provided in the lecture may be presented in a way that causes disengagement or disinterest. Alternatively the concept or task may appear alien to the student in terms of their existing schemes.</td>
</tr>
<tr>
<td>Action/Interaction</td>
<td>Given the conditions of disinterest and/or exposure to alien concepts the student may decide to disengage from that topic in a lecture. This is described in the data in different ways such as ‘switching off’, ‘drifting’ and ‘fading’. A similar action to switching is consciously skipping over a topic in programming that doesn’t make sense at a given point in time. In this case, the student may return to the topic/issue when explanations or solutions become available.</td>
</tr>
<tr>
<td>Consequences</td>
<td>The outcome of the chosen avoidance strategy may vary in terms of the level and frequency of disengagement. One outcome described in the data is that of ‘falling behind’. The outcome of skipping over material depends on the importance of the topic, whether or not it is ever revisited. Reported outcomes in the data vary form skipped material coming back to ‘haunt’ students to impeding progression in subsequent material.</td>
</tr>
</tbody>
</table>

Table 6.4 Avoidance Strategies
In terms of *intervening conditions* the student will have been provided with explanatory material in the form of notes, sample programs, text book examples etc both in lecture and laboratory environments. Furthermore, the *pace* and *volume* of the material is such that the student may put a particular concept (e.g. the assignment statement illustrated above) to the back of their mind for either a temporary or permanent basis. In the data this varied from ‘a day or two’ to ‘weeks’ or to indefinitely depending on the perceived importance of the topic for future work. Here, the data indicated if it was a vital topic such as ‘functions’ that are required for projects then it would be viewed as highly important. Furthermore, the data collected indicates that this skipping over of material may be temporary if either solutions to those tasks are put onto the student network or the skipping over was a deliberate strategy that entailed coming back to the material when help was available, or as one participant put it, ‘when inspiration strikes’.

The data suggests that the *action/interaction strategies* in terms of avoidance strategies may be wide and varied. Disengagement may happen on a *full* or *partial* basis. The earlier in the term certain material is disengaged from, the higher the likelihood of disengaging from future material given that programming material is delivered with the easier topics first e.g. input/output, arithmetic expressions, assignment statements etc. One participant recounts material ‘fading out’ but ‘picking it up again’ in the second semester. Skipping over material may be done on a *permanent* or *temporary* basis, on *small* or *large* sections of programming material with this material being of varying importance in terms of future consequences. This notion of future consequences was also found by Carbone and Mitchell (2001) when they indicated that ‘concepts that were not to be assessed were ignored by students’ (p.93). Furthermore, Robins et al. (2003) describes how Perkins et al. (1989) distinguish between two main kinds, “stoppers” and “movers”. When confronted with a problem or a lack of a clear direction to proceed, stoppers (as the name implies) simply stop. According to Perkins et al. ‘they appear to abandon all hope of solving the problem on their own’ (p.265). This abandonment has significant overlap with disengagement uncovered in this study where it may last either on a permanent basis or until assistance of some form (be it peer or lecturer) becomes available. In the data, participants describe how they disengage ‘for a while’ or indefinitely by
indicating ‘getting it eventually’ or ‘I’ll sort it out later’. According to Robins et al. (2003):

Students’ attitudes to mistakes/errors are important. Those who are frustrated by or have a negative emotional reaction to errors are likely to become stoppers. Movers are students who keep trying, experimenting, modifying their code. Movers can use feedback about errors effectively, and have the potential to solve the current problem and progress. (p.155)

It appears that there is a strong overlap between the ‘stopping’ and engagement of ‘avoidance strategies’. On the other hand, there also seems to be similarity between ‘movers’ and those who persist [P] and do not engage in ‘avoidance strategies’.

The outcome or consequences will depend on the avoidance strategy taken. Students reporting use of the disengagement strategy indicate that it has resulted in them falling behind in the subject, resulting in more work to be done to get back on track as well as attending tutorials in an attempt to ‘catch up’. Participants who skipped over certain material recount this decision as ‘coming back to haunt’ them at a later stage as well as preventing them from completing other parts of the course such as project work. In this regard, the building-block nature of learning programming becomes explicit whereby knowledge is built on a systematic basis.

When analysing these two subcategories of the avoidance strategies phenomenon, one may ask the question, what is the difference between disengagement [DISENG] and skipping over [SO]? The reason for two separate categories can be found in detailed analysis of the data, where, whilst both are conscious strategies to avoid current material there is a slight difference. The data tells us that disengagement has to do more with lack of application of the student in the lecture or laboratory where they may become lazy by losing initial enthusiasm or allowing themselves to ‘drift’, become ‘disinterested’ or ‘switch off’ from the topic/lecture/laboratory question at hand. On the other hand, with ‘skipping over’, rather than allowing themselves to drift, participants are mentally engaged in a state of working through the programming material and skip over the parts that stump them. Rather than ‘drifting’
and disengaging completely, they may often work on something else in the mean time that they do understand and revisit the difficult topic when either help becomes available or solutions are posted on the network. This appears to be consistent with the grounded theory study by Fitzgerald et al. (2005) where they indicate ‘a related strategy was to leave the question and return to it later’ (p.75). Alternatively they may choose to skip over the material indefinitely. This phenomenon was also uncovered by Bruce et al. (2004):

They may struggle to understand one concept before moving onto 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. (p.150)

6.4.3 Perception of Programming

The perception of programming phenomenon, as its name suggests, encapsulates the various different ways the participants in the study perceived the programming subject and is illustrated in fig 6.7. In this regard, the perceptions varied between positive and negative attitudes towards programming with some alternating between positive and negative feelings towards the subject depending on what task or stage the participants were at in the syllabus. This phenomenon has considerable congruence with the concept of the ‘perceptual boundary’ uncovered by Bruce et al. (2004):

The perceptual boundary is not limited to the language currently being learned, nor to a single program, but to the idea of programs and programming in a broader sense. (p.153)
Table 6.5 illustrates the paradigm frame for the phenomenon. In terms of causal conditions varying perceptions may be caused by project work, individual programming concepts, difficult new topics being covered in lectures etc. For some participants, encountering alien concepts [ALC] such as iteration and parameter passing can trigger a sense of trepidation:

'It was a steep learning curve. I had never done any programming before and I was thrown in at the deep end. It's been tough. I won't lie, it's been tough. Everything else I was expecting, the maths etc but the programming just threw me completely' (P42)

'Programming is so new like, you've never done anything like it before. Like maths and stuff you can learn theory but in programming there is only so much theory. Coding, that's the big difference. Getting used to sitting down and actually coding is hard to get used to' (P43)
Chapter 6  

Data Analysis

<table>
<thead>
<tr>
<th>Element</th>
<th>Description (Memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenomenon</td>
<td>Perception of Programming</td>
</tr>
<tr>
<td>Causal Conditions</td>
<td>Over the course of the academic year the student may have one of a number of perceptions or attitudes to the programming subject. The data indicates that this may be caused by different parts of the syllabus, laboratory work, project work, group work etc. A negative feeling attitude may be caused by excessive difficulty with programming, and this may change to a positive enjoyable experience when that difficulty is overcome.</td>
</tr>
<tr>
<td>Context</td>
<td>For the majority of students (those who have completed the leaving certificate, equivalent to UK A-level) this is the first time they have been exposed to computer programming as computer science is not part of the Irish second-level programme. The minority of students who have some programming experience will have done programming as a hobby or as part of a post leaving cert (PLC) course.</td>
</tr>
<tr>
<td>Intervening Conditions</td>
<td>When going through programming syllabus material students recount varying experiences. On the negative side students see themselves as being exposed to ‘big chunks’ or ‘big blocks’ of unfamiliar or alien [ALC] material and can be equated to being exposed to a ‘whole new language’ consisting of new unfamiliar terms or ‘words’ [W]. Furthermore, some students consider themselves as being ‘thrown in at the deep end’ and facing an ‘unexpected level of toughness’ or difficulty [D] with a ‘steep learning curve’. On the positive side, certain tasks may to be worked on result in programming being perceived as interesting and enjoyable [ENJOY].</td>
</tr>
<tr>
<td>Action/Interaction Strategies</td>
<td>In terms of dealing with alien and difficult topics students work at them and try to ‘keep on top of things’ to avoid ‘letting them build up’. Because programming is alien some students find themselves having to work harder at it as compared to other subjects. In dealing with difficulty some find doing more examples makes the material easier to understand, and this may be helped by starting off with the easier examples and building up. The data suggests that an emerging strategy is to ‘Google’ the concept to see what examples emerge. Some students skip over’ [SO] material that doesn’t make sense to them. In order to get an enjoyable experience students may ‘go in with an open mind’ and focus on the ‘interesting’ nature of the subject.</td>
</tr>
<tr>
<td>Outcome/Consequences</td>
<td>The outcomes are dependent on the on whether the student sees these issues from a positive perspective like ‘difficult but enjoyable’, or ‘interesting’ or ‘difficult and intimidating’ from a negative perspective. Positive outcomes consist of putting alien concepts into perspective and working at them. Negative outcomes entail a feeling of programming as being overly difficult [D], intimidating [INTIM] or falling behind or ‘being gone’ for the rest of the year.</td>
</tr>
</tbody>
</table>

Table 6.5 Perception of Programming
This data where the participants contrast the alien nature of programming to other subjects like mathematics is consistent with the findings of Rogalski and Samurcay (1990) where they indicate:

> Programming as a knowledge domain differs from other neighbour domains such as mathematics in two ways’ First, there are no everyday intellectual activities that can form the basis for spontaneous construction of mental models of programming concepts such as recursion or variables, in contrast to such notions as number or velocity. Secondly, programming activity operates on a physical machine which may not be transparent in its functioning for learners. (p.162)

Eckerdal and Berglund (2005) also describe this issue as emerging in their study where they indicate where a student ‘experiences a lack of method and theory when learning to program compared to when learning mathematics. Mathematics is thus experience as an easier subject to study’ (p.138). On the other hand, experiencing programming as an interesting and enjoyable challenge can trigger a positive attitude. In addition, given the dynamic nature of programming where the student can see the fruits of their work immediately when they compile their programs and execute them, it can prompt positive senses of achievement and accomplishment [ACC] that reduces boredom.

The context of this phenomenon strongly relates to the background of the participants where the vast majority have been direct entrants from the Irish Leaving Certificate examinations (equivalent to the UK A-Level). Whilst computer applications type material such as word processing and desktop publishing is covered in the Irish secondary school system, computer science is not. For the vast majority of students, this will be the first time they have seen programming, algorithms and problem solving. In this regard, some students may see a combination of material that is perceived to be difficult [D] in a format that is alien [ALC] to them:
'I found it very tough [D], it was completely (extent) alien to me and when I started looking at programs, sitting in lectures scratching my head and I wasn't really sure what was going on [BW]' (P8)

'It gets tough [D], at the start you are like trying to learn a whole new language which I'm really bad at [SLFDBT]' (P19)

'With most subjects in college, they kind of continue from secondary school, but there was nothing like this in secondary school. Everything is so different' (P29)

This intersection of the alien concept [ALC] category and difficulty [D] is a good example of a 'contextual condition' as described by Strauss and Corbin (1998) where one identifies in the data:

Specific sets of conditions that intersect dimensionally at this time and place to create the set of circumstances or problems through which persons respond through actions/interactions. (p.132)

In this context, a high level of difficulty over a long duration encountered with concepts that are considered to be very alien may cause the student to struggle and foster a negative perception of programming. A minority of participants in this study had prior programming experience [PPK] either gained through other Post-Leaving Certificate courses (PLC) or through having programming as a hobby. This finding in the data is consistent with that of other researchers. For example, Cantwell-Wilson and Shrock (2001) outline how they uncovered in their study 'previous programming experience - a dichotomous variable determined by whether the subject had engaged in any programming prior to the course' (p.185). They go on to indicate that this can be broken into two sub-categories:

(a) formal programming course taken – a dichotomous variable determined by whether the subject had a previous programming course or not and (b) self-initiated programming experience – a
Chapter 6

Data Analysis

dichotomous variable determined by whether the subject learned to write programs separate and apart from a formal class. (p.185)

These findings closely resemble those uncovered in this study. In another study, Hadjerrouit (1998) relates this to this as an issue pertaining to students’ preconceptions based on prior experience and the impact that this may have:

Identify students’ preconceptions. We know that preconceptions, e.g. procedural programming, interfere with the learning of Java concepts. This prior programming experience is, to some extent, a significant barrier that makes it difficult to understand the new concepts. Our experience suggests that the best way to overcome these preconceptions is for students to confront them. Thus, learning activities should involve problem situations that lead students to recognise that their approach is manifestly inadequate. Then, we introduce Java concepts, e.g. object oriented concepts, as a way to modify students’ prior knowledge. (p.106)

As is clear above, this study not only identifies the issue of prior experience but also discusses the implications this has on teaching practice. In this regard, there seems to be a responsibility on the educator to point out the inadequacy of the student’s approach in a way that avoids de-motivation, perhaps by comparing and contrasting their proposed solution with a more suitable one. Here, the student might view the correct parts of their proposed solution as a positive factor and may motivate them to work harder on completing the remaining portion. In this way, a potential negative issue of failure to complete an entire task can be transformed into a positive one, where potential strengths of the student are focussed on. The implications of similar issues uncovered in this study will also be discussed later in this thesis.

Participants also describe issues of alien concepts [ALC] and difficulty [D] in terms of what they already know, e.g. the following participants had programmed in Visual Basic as a hobby before coming onto the course:
'Classes were difficult to understand, not classes, but the concept of classes. As I hadn't seen it in Visual Basic before, it was something new, apart from that I didn't have any problems. Classes were a major difference to what I had seen beforehand' (P20)

'Well classes were definitely new. That was the first major thing that was different for me. It was a totally new way of thinking about it. The whole thing about encapsulation and keeping the different parts of the program separate from the other parts. The whole thing about object-oriented programming really because I hadn't come across it' (P28)

In terms of the above data one witnesses significant similarity with the 'inconsistency problem' described by Spohrer and Soloway (1989) who indicate that 'because novices understand how a construct works in one situation, they may assume that the construct will work in the same manner in another slightly different situation' (p.411). Furthermore, from these examples one witnesses the varying contextual conditions within which a student perceives the alien nature of computer programming which is relative to what they have or haven't seen before. If one examines the data in the alien concept [ALC] category in appendix 2 one can see that a number of participants contrast this to the mathematics subject which is seen as a logical continuation from secondary school and something to which they can ascribe some sort of meaning to.

Analysis of the data uncovers a number of potential intervening conditions that may be at play in terms of this phenomenon. One participant recounts experiencing a 'steep learning curve' based on lack of prior exposure to programming and this 'threw them completely'. In terms of the perceived difficulty [D] of programming participants describe being presented with 'big chunks' of information and code in terms of 'methods' and recount the transition from theory to practice as non-trivial. This situation may be exacerbated by the fact that as one participant put it, 'there are hundreds of ways to do things' and this may have the effect of complicating matters. Another interesting situation a number of participants experienced was that of encountering 'words' [W] or terms for the first time that didn't make sense:
'I go to the lecture notes [LEC] and everything is just big words, just jumping out at me, I don't understand it and say “OK”, so I kind of skip over [SO] the big words where I know the big words actually mean something and have relation to the things I can't get' (P2)

Programming is all new words that you have to learn. In programming it's write a loop and put in variables and all this stuff' (P45)

'Well using stuff like arrays. That stuff really doesn't do what it says, the name doesn't help a lot' (P39)

'If it's over 60 or something you get a certain grade, you can relate to “ifs”. Ifs were used with ticket prices and things like that. When you were doing arrays you have a single 'i' and "customer ++"' (P24)

The last three pieces of data are interesting to the extent that the participants suggests one reason for this unintelligible words [W] experience is where perhaps an ‘if’ statement relates to the English language usage if the word ‘if’ and may be easier to understand or relate to, e.g. 'if the grade the student attained is greater than or equal to 40 then they will pass the subject' as illustrated in the following piece of Java code:

```java
if (grade >= 40)
{
    System.out.println("you passed");
}
else
{
    System.out.println("you failed");
}
```

This notion of difficult words in an unfamiliar or 'alien' environment recounted by participants in this study has particular resonance with the findings of Clarke et al.
(1998) where they not only relate to the existence of these unintelligible words in the object-orientation topic of programming but also the net outcome of the non-accommodation of these words:

Compounding this problem is that the (English) language used in talking about object-oriented programming is fairly sophisticated language. Words such as “instant”, “encapsulation” and “hierarchy” are quite difficult, and particularly so for the substantial number of students for whom English is not their first language. By the time we have introduced “method”, “inheritance”, “casting and “polymorphism” the combination of unfamiliar and difficult words to describe unfamiliar and difficult concepts is very daunting. Naturally, students with difficulties comprehending objects and classes are doomed to struggle throughout the course, as all the other concepts are built upon this. When asked to write Java code, these students would literally not know where to start - some would write methods that did not belong to any class, while others would simply attempt to write lines of Java instructions. (p.175)

The detailed discussion of ‘words’ [W] above highlights an association with another related category existing schemes [EXSCH] that will be discussed in this chapter. On the positive dimension, the engagement in work of an interesting and challenging nature can result in programming being a non-boring and enjoyable [ENJOY] experience:

‘I enjoy project work, especially now we are working on stuff that’s not small and stupid (extent), which you’ll never use’ (P36)

‘It’s the most interesting. You can see it being used, you compile something and it works’ (P35)

‘I enjoy it, it’s interesting. I know a lot more than I did at the start’ (P28)
‘The programming language looked to me first like a boring task, then when I got used to it I found it interesting and today I spend hours in front of my screen writing java and applet programs’ (P4)

These findings relating to enjoyment are consistent with those of Perkins et al. (1989) who indicate that ‘with appropriate instruction and encouragement, students can learn not to give up so easily. The result may be enhanced learning and enjoyment for many students’ (p.267). To summarise, in terms of the various intervening conditions that can arise for the student one can see that they can range from negative to positive with a range in between depending on how the student progresses with the material or task at hand.

The range of action/interaction strategies may vary depending on contextual and intervening conditions being experienced. In terms of dealing with alien and unfamiliar concepts [ALC] students employ various strategies such as trying ‘to keep on top of things’ so that they don’t ‘build up’ as well as working harder on programming than other subjects because of its alien nature:

‘It’s difficult [D] because it takes a lot more brain power. The rest of the subjects came naturally to me but the programming hasn’t. Programming has been something I’ve had to work at. The other subjects have sat OK with me’ (P42)

In terms of dealing with difficulty students may employ many strategies such as beginning with smaller examples and working their way up to more complex ones, ‘Googling’ the concept on the internet if there isn’t adequate assistance available to overcome the difficulty as well as looking at the program or task to try and get a perspective on it (referred to by a number of participants as ‘getting their head around it’ [GH] described later in this chapter). Other positive strategies employed vary from ‘keeping an open mind’ to working with programs for longer durations as part of projects or other laboratory tasks. Negative action strategies entailed skipping over [SO] unintelligible words [W], alien [ALC] or difficult [D] material. This has also been reported by Carbone and Mitchell (2001):
Programming tasks that do not sufficiently emphasise the key points or contain too many unfamiliar concepts or involve designing long coding solutions can lead students to attend to parts of the exercise and overlook other sections. (p.94)

An interesting issue that emerged in the interviews and a focus group discussion was a view from participants that this negative skipping over strategy could be avoided if certain explanatory mechanisms could be put in place:

'My lecturer just went through it bit by bit instead of a long paragraph of stuff in the notes' (P43)

'A language dictionary explaining the terms would be useful. A simple description in lay man's terms and then a more detailed description would be ideal' (P24)

'I think it would be better to get a full description of everything at the start and why you use it' (P23)

To summarise, the data indicates that varying strategies may be employed by students which are dependent on what actual perception of programming they are experiencing at any given time. Once again, it is noteworthy that the data suggests that this may be in a constant state of flux where enjoyment at implementing 'if' statements and 'loops' may be replaced with difficulty or a sense of intimidation with the alien nature of object-orientation. This finding is consistent with that of Clarke et al. (1998):

At some early stage in the course, it becomes necessary to explain exactly what objects and classes are. These two concepts are inextricably linked, but it is not very useful to the students to be told that an object is an instance of a class, and that a class is a collection of objects. Trying to give a formal definition of an object ("an object is the encapsulation of data together with methods to manipulate that data") just introduces even more terms with which the students are unfamiliar. (p.175)
Finally an interesting issue that emerged related to participants suggesting mechanisms that could reduce the impact of alien terms and material e.g. a type of dictionary that would explain difficult words.

The *outcome* or *consequences* of this phenomenon are dependent on whether the perception of programming is positive or negative. Negative outcomes varied significantly in the data. One participant described a feeling of self doubt [SLFDBT] when faced with unfamiliar concepts such as integers and booleans (data types) and loops (iteration control structures), whilst another faced with the same type of unfamiliarity finds themselves ‘slogging’ (working really hard) in the programming subject. Viewing difficulty [D] from a negative perspective and not getting a basic understanding (or getting their head around it [GH]) may cause the student to be ‘gone for the rest of it’ or finding the programming course overly intimidating [INTIM] or daunting. Failure to understand the new programming terms or words [W] and subsequently skipping over them [SO] can result in impeding progress in future related topics as described in the ‘avoidance strategies’ phenomenon. Positive outcomes can take the form of the student persisting [P] with the unfamiliar material until they find themselves eventually getting ‘used to it’. Students who find programming interesting and enjoyable [ENJOY] from the beginning persist [P] with tasks by getting their programs to compile and eventually working.

To summarise, within this perception of programming phenomenon one witnesses a wide range of issues that not only range between negative and positive experiences but we also may see alternation between positive and negative perceptions depending on the values of the properties and dimensions of mutually occurring categories at a given point in time. For example, one may see a transformation of difficulty [D] from a positive and interesting experience to a negative and intimidating one. This issue will be further discussed in the learner categorisation section.
6.4.4 Programming Organisation

The term *organisation* is defined by Woolfolk (2001) as 'the ongoing process of arranging information and experience into mental systems or categories' (p.28). Given the nature of certain categories uncovered in the data it was decided that *programming organisation* was a suitable label for the phenomenon that encapsulated issues such as material delivery (MD), prior programming knowledge (PPK) etc. (illustrated in fig 6.8) which may influence the student’s process of organising delivered programming concepts into mental schemes. Table 6.6 presents an overview of this phenomenon.

![Fig 6.8 Programming Organisation](image-url)
<table>
<thead>
<tr>
<th>Element</th>
<th>Description (Memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenomenon</td>
<td>The student will be presented with a sequence of programming concepts as they progress through the syllabus. The material may be presented in a way that can either fit their existing mental schemes or overlap with prior programming experience they may have. On the other hand, the way the material is delivered and/or the programming examples used may not fit with existing schemes, there may exist long durations of time for material to be digested or ‘sink in’ [SI]. The data suggests that some material e.g. “if” statements may sit well with both those with and without prior experience as they have conceptual overlap with everyday English language usage of the word “if”.</td>
</tr>
<tr>
<td>Context</td>
<td>For the majority of students this is the first time they have been exposed to computer programming concepts and their prior programming concepts [PPK] will be non existent. Therefore, they may not have built up any prior mental schemes [EXSCH] that can facilitate accommodation of programming concepts and thus the delivery mechanisms using suitable programming examples may be important in terms of progression. The minority of students who have some programming experience will have done programming as a hobby or as part of a post leaving cert (PLC) course and may have built up some schemes to facilitate learning programming. The data indicates that this prior knowledge is mostly partial and no instance was uncovered of prior knowledge of object orientation.</td>
</tr>
<tr>
<td>Intervening Conditions</td>
<td>Given the variety of subcategories a wide variety of conditions make occur at any one time. A student may not understand a concept until it is explained in a way that fits one of their pre-existent schemes. The pace and depth of material delivery [MD] may determine how well the material is accommodated by the student. As the material progresses various code examples [CODEX] will be presented to students and these may be viewed as clarifying the concept to a certain degree [CLAREX] or alternatively may be viewed as ineffective [IE] because they either don’t explain the concept fully or they do not use an example that appeals to the student, e.g. a chess board or soduku.</td>
</tr>
<tr>
<td>Action/Interaction Strategies</td>
<td>The action strategies undertaken vary with the intervening conditions. When encountering a new topic a student may attempt to assimilate by fitting newly encountered concepts into existing schemes or coming up with their own example (self example [SE]) of how it will be applied. When faced with new material being covered at a relatively fast pace a student may attempt to ‘keep on top of it’. Self examples [SE] may be written (varying from simple to elaborate) to facilitate understanding of lecture notes, make programming more applicable to what one is interested or to avoid boredom. The student may also search various sources e.g. lecture notes, books and internet for code examples [CODEX] that implement the concept under consideration.</td>
</tr>
</tbody>
</table>
For the majority of participants the desirable outcome is successful accommodation of the material. Failure to achieve this may result in the student being ‘gone for a while’. During lectures a number of participants recount a significant period of time for the lecture material to ‘sink in’ [SI] which can be dependent on the perceived pace of delivery. Others recount that it takes a certain period of time to ‘get your head around’ [GH] the whole new concept of programming. For others, certain material and concepts may suddenly ‘click in’ [CLK] in a ‘eureka’ type moment. These outcomes may vary in the amount of time it takes to accommodate programming concepts and whether the student has achieved partial or full accommodation.

<table>
<thead>
<tr>
<th>Outcome/Consequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the majority of participants the desirable outcome is successful accommodation of the material. Failure to achieve this may result in the student being ‘gone for a while’. During lectures a number of participants recount a significant period of time for the lecture material to ‘sink in’ [SI] which can be dependent on the perceived pace of delivery. Others recount that it takes a certain period of time to ‘get your head around’ [GH] the whole new concept of programming. For others, certain material and concepts may suddenly ‘click in’ [CLK] in a ‘eureka’ type moment. These outcomes may vary in the amount of time it takes to accommodate programming concepts and whether the student has achieved partial or full accommodation.</td>
</tr>
</tbody>
</table>

Table 6.6 Programming Organisation

In terms of causal conditions the student may be exposed to a sequence of programming concepts (one participant describes this as ‘a whole lot of stuff being thrown at you’). For example, a typical programming syllabus may begin with simple input and output and eventually move on to more elaborate concepts like abstract methods, polymorphism etc. as part of object-orientation. In the case of a student with prior programming knowledge [PPK] coverage of material that they haven’t encountered before may cause some difficulty:

‘Classes were difficult to understand, not classes, but the concept of classes. As I hadn’t seen it in Visual Basic before, it was something new, apart from that I didn’t have any problems. Classes were a major difference to what I had seen beforehand’ (P20)

‘I did VB there, but after 6 weeks here I was into more detail than the whole year at the other college (extent). This course is more in-depth’ (P24)

‘It’s the Java language, you have to know everything (quantity) to go and do it. In VB you have to know the code as well but it’s more straightforward’ (P18)

154
One participant recounts that moving from 'if' statements (something that made sense to them given the conceptual overlap between its programming meaning and its English language meaning) to an unfamiliar more abstract concept like arrays caused difficulty as it felt like it was moving too fast in terms of pace. Other participants describe similar experiences:

'I found the start quite easy when we were doing the "ifs" and I really got the hang of them, but after that we just flew through arrays. It was like all the hard stuff [D] we just flew through it' (P23)

'It's a bit difficult [D], there's a lot to learn like all those arrays and stuff. I don't know I'm not catching it in lectures' (P2)

From the perspective of context, as highlighted throughout this thesis, the majority of participants have had no prior knowledge of programming. In this regard, they may not have built up mental schemes that correlate with abstract concepts like arrays, iteration control structures, and parameter passing in functions or methods. In this regard, Sheil (1981) contends that 'as novices do not have the specialised knowledge and skills of the expert, one might expect their performance to be largely function of how well they can bring their skills from other areas to bear' (p.119). This issue was referred to in chapter 2 where the student does not possess programming plans which were described as a catalogue of stereotypic action sequences that expert programmers have and are able select and use by adapting them to the need of the current situation and thus, demonstrating knowledge transfer. Furthermore, for some students with limited prior knowledge the pace of material delivery [MD] may appear very fast:

'It is very fast paced (pace), you take in a lot (quantity) in each class and if you don't understand it you move on to the next class without asking questions [SO], you fall behind' (P14)
'When I start new concepts like methods, constructors, classes, inheritance etc, all of like in 2 weeks or something, so much to take in and try to put them together' (P19)

'You don't have a lot of time to do stuff so it's tough going [D] (P19)

'It goes off really quickly. You think you are on top of it and all of a sudden you are saying what's going on?' (P42)

'You have to cover the stuff so fast, you're constantly moving. Unless you do revision yourself there is no time to go back over something you miss, even the slightest detail, you are pretty much gone for a while (consequence)' (P43)

The conglomeration document in appendix 2 introduces two noticeable properties of material delivery [MD] i.e. pace of coverage and perceived depth of material. This is consistent with the findings of Samurcay (1989) where he indicates that 'the notion of complexity is not only defined in terms of the quantity of information processed simultaneously, but also in terms of conceptual difficulty' (p.163). The data also uncovered the notion of prior knowledge [PPK]. The extent and usefulness of this prior knowledge varied amongst participants, with one emerging issue being that Visual Basic is not an object-oriented language and therefore does not give the student experience of objects, classes, methods and other object oriented concepts, which are an integral part of programming in Java and C++, i.e. the two languages used at the data collection sites. On the other hand Bruce at al. (2004) refer to prior experience as being 'considered valuable by students who experience learning to program as learning to solve problems' (p.152). This notion was found in this study where some participants with prior programming knowledge [PPK] found programming enjoyable [ENJOY] as it built upon the knowledge they already gained and thus increased their skill set.

In terms of intervening conditions a number of situations may exist at any given time. When encountering an abstract concept like polymorphism for the first time the
student may not have a pre-existing-scheme to accommodate it. This may also be exacerbated by the perceived depth and pace of delivery where polymorphism is but one of a number of concepts covered in object orientation. In attempt to explain these concepts the lecturer usually presents a number of different code examples [CODEX] to students e.g. a ‘for’ loop that prints out the numbers one to ten on the screen or an example like Sodoku or Chess to explain two dimensional arrays. An illustration of the former example of a ‘for’ loop is as follows (where it prints out the numbers 1 to 10 on the screen each on a separate line):

```java
for (index = 1; index <= 10; index ++)
{
    System.out.println (index);
}
```

A ‘loop’ is a computer programming term that relates to repetition, where one can imagine getting to the last line of the code and ‘looping’ back up to the top again to execute the same sequence of code again. This looping reoccurs until a satisfied condition has been met i.e. in the above case the control variable ‘index’ reaches the value 10. According to Robins et al. (2003) ‘example programs are sources of programming knowledge, but the strategies that go into creating a program are not usually visible in the final product’. (p.166). In the cases where these strategies are visible in the code example a given student may view these types of examples as clarifying [CLAREX] the concept to a certain degree:

'I have a couple of examples that stick with me' (P37)

'I'm always looking for an example somewhere, once I have an example it's usually set in stone' (P37)

'My lecturer sat down with me and had a nice little diagram that was like a tree basically where every branch came back to the main trunk of this tree. My lecturer then showed me what I had which looked like a car crash basically. That one little diagram stuck with me and that's what showed me the light' (P37)
The above extracts from the data crystallise the concept of a clarifying example where the participants indicate a 'setting in stone' moment that sticks with them from that point onwards. This finding has significant resonance with the notion of 'focal design' as illustrated by Rist (1995):

Focal design occurs when a problem is decomposed into the simplest and most basic action and object that defines the focus of the solution, and then the rest of the solution is built around the focus. Essentially, the focus is where you break out of theory into action, out of the abstract into the concrete level of design. (p.537)

The data suggests that these clarifying examples can be simple or elaborate and tend to appeal to the student when they are presented in a way that either uses examples that are familiar to them or sparks their interest and have the effect of the student not 'drifting' away from the topic. In their grounded theory study, Fitzgerald et al. (2005) relate to this issue with a code described as 'semantic':

Subjects sometimes identified an idea with which they were not familiar. In those cases, many subjects constructed an understanding of this new idea by relating it to some previously understood information or by using an example to make understanding clearer. (p.74)

The above finding highlights an example of the overlapping of the codes 'existing schemes' [EXSCH] and 'clarifying examples' [CLAREX]. Conversely, other code examples presented to students may be considered ineffective in getting the concept across:

'The example in the notes was to do with chess. I didn't realize what the lecturer was talking about because I don't like chess at all' (P21)
For example, arrays were explained in the form of a grid as in an excel spreadsheet but that didn't integrate very well with me which probably isn't a good thing. You need to link with logic that's already in your head [EXSCH]' (P41)

These types of example are referred to on the conglomeration document in appendix 2 as 'ineffective examples' [IE]. When these were uncovered the author documented the following theoretical memos:

**Memo** – here we see an overlap with the 'conscious disengagement' code. Using an ineffective example in this case caused the student to disengage from trying to assimilate the concept of two-dimensional arrays

**Memo** – I am reminded of a previous participant who referred to an inadequate example of a chess board to explain two-dimensional arrays. It is interesting to find so many overlaps across multiple sites

Clarke et al. (1998) also uncovered this notion of an ineffective example in terms for teaching-object oriented programming with the Java programming language:

Ultimately it seems that the only way forward is to present to the students a range of examples of objects and classes examples such as "John is an instance of the class Person", "Fido is an instance of the class Dog". This then relies on the students' ability to abstract general principles from specific examples - an ability that is not very highly developed, particularly in the weaker students. (p.175)

As highlighted in the theoretical memos above, it may be the case that there is a small window of opportunity within a fast moving subject like programming for a student to accommodate a concept like two-dimensional arrays or the notions of objects and classes as described by Clarke et al. (1998) above. If this opportunity is lost because of inadequate examples in terms of appealing to the existent schemes or learning
styles of students then this may cause the student to engage in an avoidance strategy such as disengagement. This experience was also found by Bruce et al. (2004) where they indicate ‘if the material is presented in a way that doesn’t match their expectations or perceived needs, they experience frustration’ (p.148).

As table 6.6 indicates the action strategies undertaken vary with the intervening conditions experienced by the student. In attempting to accommodate a new topic in programming the student may try to link or associate it with an existing mental scheme [EXSCH] that they possess:

'It would depend on how the lecturer had gone about explaining it in the first place, because I normally try and link that in with something that's already there and just expand on something already inside my mind (matching level) rather than creating an entirely new subject' (P41)

'Go through it yourself and figure it out in a way that makes sense in your head' (P42)

As described in appendix 2 the matching level property of this code relates to how well the schemes the participant possesses actually matches the programming concept under consideration. If this is very partial the student may have to generate a new scheme to facilitate the concept. As illustrated in table 6.6 this action strategy is consistent with a transition from assimilation to accommodation. This notion of an existing scheme has also been uncovered by other researchers. According to Robins et al (2003) 'programs are usually written for a purpose – with respect to some task, problem, or specification. Clearly an understanding/mental model of this problem domain must precede any attempt to write an appropriate program' (p.149). In terms of prior knowledge they go on to state ‘given that knowledge is (assumed to be) uniformly low, it is their pre-existing strategies that initially distinguish effective and ineffective novices’ (p.166). Bruce et al. (2004) refer to a phenomenon in their study of ‘understanding and integrating’ where students ‘view learning to program as building on prior experience’ (p.150). In a related issue, Mayer (1989) describes the concept of ‘meaningful learning’ where the learner makes a connection between new
material and knowledge that already exists in memory. In particular he indicates that ‘the existing knowledge in memory has been called ‘schema’ and the process of connecting new information to it has been called assimilation’ (p.129). This description presented by Mayer (1989) in the context of learning computer programming is consistent with the learning theory concepts provided in table 6.1. In terms of existing schemes [EXSCH], Samurcay (1989) in his study, also found that ‘the students are more able to process variables for which they have already a representation and processing system constructed in domains other than programming’ (p.177).

A related action strategy in this context is engaging in the development of self examples [SE]. As illustrated in appendix 2 the data indicates there may be varying rationales for this type of strategy. Firstly, its purpose may be explanatory where the student creates the simplest possible example to understand a concept. Secondly, as we have just discussed, the student may develop an example that more neatly fits into one of their current mental schemes. Thirdly, self examples may be written to avoid boredom or to make programming more interesting by applying it to areas that they have a personal interest in e.g. golf, text based games, calculators and mathematics are a few examples that appear in the data. This notion of ‘self examples’ has also been documented in literature. Hadjerrouit (1998) indicates:

> It is the practical problem that should motivate students to learn the tools and to construct the Java concepts, as opposed to presenting the concepts and the tools with de-contextualised examples and definitions independently of problem solving. Thus, a problem solving approach motivated by constructivism should begin with intellectually stimulating, realistic, and intrinsically motivating problems. (p107)

In this regard, Hadjerrouit concurs with the findings in this study whereby contextualisation of illustrative examples may be a key factor in the student accommodating a concept. If this isn’t achieved we may find a de-contextualised example may fall into the category of an ineffective example [IE] that may result in avoidance strategies with respect to the respective programming concept. In relation
to this, Winslow (1996) brings the educator's attention to the possibility of the ineffectiveness of a self example [SE] where he indicates that models are crucial to building understanding. Models of control, data structures and data representation, program design and problem domain are all important. Furthermore, 'if the instructor omits them, the students will make up their own models of dubious quality' (Winslow, 1996, p.21). In this regard, one sees an overlap between 'ineffective examples' [IE] and 'self examples' [SE] where self examples created by the student may be (unknown to them) ineffective or of dubious quality in actually accommodating the concept under consideration. Robins et al. (2003) indicate the potential effectiveness of self examples:

Mayer (1989) also showed, as we would predict from the general educational literature, that students who are encouraged to actively engage and explore programming related information (by paraphrasing/restating it in their own words) performed better at problem solving and creative transfer. (p.158)

As indicated in table 6.5, the desirable outcome for most students is successful accommodation of the programming material. Failure to reach a satisfactory level of accommodation may result in the student 'falling behind' or 'being gone for a while'. The latter outcome displays overlap with the disengagement [DISENG] code described earlier in this chapter. A number of participants describe a period of time for material delivered in the lecture to actually 'sink in' [SI]. The data suggests that for a number of participants this sink in time takes longer that may have been envisaged by the lecturer:

"I don't learn a lot when I'm sitting in a lecture room because it doesn't go into my brain, so I'm just sitting there trying to learn it' (P19)

'I look at my notes, talk to someone in the class that I do programs with or something and understand as much of it as I possibly can, because I don't think one lecture on the subject is enough to get it’ (P19)
"I'm sitting looking at notes and they are not making any sense to me. I'm there following the flow and still not grasping it, then coming back to it again and still not grasping it' (P24)

"All of this is coming at you fast [MD] and you are being asked do you have any questions. But you don't know because this is your very first time seeing something, so you are not going to take it all in until you go back and read the notes. A lot of the time in a lecture you cover a lot of material. You might cover a topic in the space of an hour maybe and I find I had to take it all in. You'd understand it while the lecturer is explaining what is going on but when you sit down to do it yourself you say to yourself "what was the lecturer saying there?". You read back over the notes and it looked easy when they were doing it. When you are doing it yourself step by step on the computer it probably sinks in a bit better rather than the lecturer just calling it out' (P10)

If this sink in time [SI] is longer than the lecturer envisaged, the students may find themselves working on assessment and exams on material for which they haven't had sufficient time to grasp or digest. This issue may have implications for teaching practice and overlaps with the findings of Spohrer & Soloway (1989) who indicate that:

From our experience we conclude that students are not given sufficient instruction in how to "put the pieces together." Focusing explicitly on specific strategies for carrying out the coordination and integration of the goals and plans that underlie program code may help to reverse this trend. (pp. 412–413)

These findings of 'sink in' time [SI] in the data are also consistent with the work of Sloane and Linn (1988) who describe the notion of 'active learning time' whereby effective teachers maximise the proportion of the time students are engaged in academic tasks in order to optimise the students' opportunities to learn and practice material. In terms of programming Sloane and Linn (1998) become more specific:
In programming classes active learning time has two components: (a) class time devoted to learning and practising problem-solving strategies, and (b) time spent working on line. Pre-college programming classes often do not provide sufficient time in amount of instruction or practice on the computer for students to develop problem solving skills. (p.210)

From this assertion we can see that Sloane and Linn (1988) are of the opinion that this is a situation that can be addressed in some way by increasing active learning time. On a related issue, other participants in the study describe a certain length of time to get their head around the whole concept of programming in general:

'At the start I had to get my head around it, then it was OK' (P18)

'Just getting your head around the whole idea of it at first. It can be daunting' (P39)

'Sometimes I might not get a full grasp of how exactly everything works' (P36)

This concept of difficulty in getting one’s head around the whole notion of programming has also been found by Winslow (1996) who refers to an important point that relates to:

The large number of studies concluding that novice programmers know the syntax and semantics of individual statements, but they do not know how to combine these features into valid programs. Even when they know how to solve the problems by hand, they have trouble translating the hand solution into an equivalent computer program. (p.17)

This finding of Winslow (1996) has significant resonance with the extent property of the [GH] code (see appendix 2) where coming to grips with syntax and semantics of a programming language like Java is only part of the requirement. In addition to this,
the student must be able to come to terms with merging these syntactic and semantic notions with that of problem solving and algorithm development. Interestingly, the study carried out by Eckerdal and Berglund (2005) also uncovers this issue where they indicate that “many of these students seem to have problems identifying what “programming thinking” involves. Some express themselves as if it is something magic, difficult to catch” (p.137).

On the other hand, the longer the duration of a concept ‘sinking in’ or ‘getting your head’ around programming in general can be contrasted to something suddenly ‘clicking in’ [CLK] in a sort of ‘eureka’ moment. One participant describes waiting anxiously for this moment to occur, whilst another regrets it not happening earlier. Finally, the implementation of self examples [SE] appears to have positive outcomes such as a sense of ‘satisfaction’ and feeling more ‘comfortable’ with certain concepts. Robins et al. (2003) concur with this assertion by indicating that ‘working on easily accessible tasks, especially programs with graphical and animated output, can be stimulating and motivating for students’ (p.158). Conversely, Carbone and Mitchell (2001) indicate that in some cases the self example approach may have negative results where they indicate that ‘the learner may focus on an interesting example and ignore a major point’ (p.94). Here the interesting part that appeals to them may not address the core conceptual issue that needs to be accommodated. For example, when using a golf score card example to implement the ‘array’ data structure, the student may focus on the user interface rather than the actual code to add and extract data to and from an array.

To summarise, programming organisation may be assisted by prior knowledge whether in the form of existing schemes or prior programming knowledge. In addition, the perceived pace and depth of the material as it is being delivered may determine how well the student progresses. Furthermore, the student may take varying amounts of time in accommodating certain programming concepts. In order to assist understanding certain parts of the syllabus the student may engage in utilising varying types of programming examples in the hope that they will be effective in clarifying the task or concept.
### Chapter 6 Data Analysis

#### Element Description (Memo)

<table>
<thead>
<tr>
<th>Phenomenon</th>
<th>Description (Memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Causal Conditions</strong></td>
<td>Encountering varying issues at different times in the year may cause a number of states of mind. The requirement to gel together a large amount of intricate material into a working program can seem confusing [CONF] and ‘crazy’. Sitting in a lecture or a laboratory where each successive concept builds on the previous ones can cause a sense of being ‘lost’ and bewilderment [BW]. For others, this confusion [CONF] may be viewed as a challenge to work harder.</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>The data indicates that many participants find themselves covering material in lectures, moving to laboratories and projects and then having to put it all together given the intricate and exact nature of programming where leaving out a semi colon will cause the project not to compile and execute. Furthermore, this confusion [CONF] can be exacerbated by the perceived unintelligible format of programming languages. Some students find themselves sitting in lectures wondering what the lecturer is talking about may feel sense of bewilderment, self doubt, and isolation. A number of participants also describe being aware of the progress of others [AOO] that can be perceived as being better or worse than their own. Negative feelings may be exacerbated by the pressurised nature of programming where the quantity and intensity of the work is perceived to be high.</td>
</tr>
<tr>
<td><strong>Intervening Conditions</strong></td>
<td>Persistence and working through material may reduce the level of confusion. On the other hand moving on to more difficult material can result in a transition from satisfaction to bewilderment [BW]. For some students awareness of the more satisfactory progress of others [AOO] may entrench bewilderment and self doubt, whilst for others it is an incentive to work on and get through the material.</td>
</tr>
<tr>
<td><strong>Action/Interaction Strategies</strong></td>
<td>Many students persist their way through material and work in a consistent manner to overcome confusion [CONF]. Others who have let negative feelings overcome them in the past may make a deliberate decision, ‘to get back into it’. Others may engage with those who are having similar levels of confusion and difficulty and listen in to the advice the lecturers are giving those students in the laboratories. In order to deal with pressure [PB] a student may make a concerted effort to ‘keep on top of’ the material.</td>
</tr>
<tr>
<td><strong>Outcome/Consequences</strong></td>
<td>As in other phenomena The outcomes are dependent on the on whether the student sees these issues from a positive or negative perspective. Some students can deal with confusion and persist until the material ‘becomes a little bit clearer’ ‘slowly but surely’. Others may find themselves being overcome by the intensity and pace of the material and can become ‘disillusioned’ and begin to doubt their participation on the course [SLFDBT].</td>
</tr>
</tbody>
</table>

Table 6.7 State of Mind
6.4.5 State of Mind
Fig 6.9 illustrates the phenomenon given the label, 'state of mind'. This abstracted category represents how one witnesses in the data the student going through many different states of mind, some positive and some negative. Interestingly, categories like confusing [CONF] and awareness of others [AOO] may represent both positive and negative states of mind depending on the type of student in question. For some, the confusing nature of programming may be seen as a challenge that can be overcome with hard work and application (positive) and awareness of the difficulties of others may have little or no impact on their concerted efforts to succeed. On the other hand, confusion for long periods over a large amount of material may cause negative feelings of bewilderment [BW] and isolation. Furthermore, awareness of the superior progression of others may serve to entrench these negative feelings. Table 6.7 illustrates the paradigm frame for this phenomenon. This phenomenon exhibits considerable congruence with the findings of Robins et al. (2003) where reference is made to underlying issues such as confidence and emotional responses:

What underlying properties make a novice effective? How can we best turn ineffective novices into effective ones? A deeper understanding of both kinds of novices is required. The range of potentially relevant factors includes motivation, confidence or emotional responses, and aspects of general or specific knowledge, strategies, or mental models. (p.165)

Fig 6.9 State of Mind
In terms of causal conditions, these can vary throughout the progression through lecture, laboratory work and projects. The emergence of significant volumes of programming material that appears confusing [CONF] may cause a negative state of mind:

‘You can’t do this, you just can’t figure out why it is there, what’s the point of having a class separate to there, inheritance, just put it all into one class and do it like there, having separate classes, all doing inheritance, interfaces and all, its just crazy’ (P19)

‘Lectures can be quite daunting, classes are what we are on now and the lectures they just seem crazy to me. There’s like set and get and there is so much happening (quantity), but you sit down in the laboratory, it’s hard to understand [D] just looking at notes’ (P29)

As can be seen above, in describing their attempts to deal with object-oriented programming the students see themselves faced with many concepts that must all gel successfully into a complete program. Others find themselves being confused by what they see as the non-intuitive nature of programming syntax:

‘If you miss a day (context) and then you come in and see “i” and “j” and stuff and you don’t not what the hell they are on about’ (P22)

‘The best way to learn it first time, call it equal to the variable + 1. It makes it easier for people to understand. You are just adding one to the same value you had before. If you say the “variable++” you are just confusing people’ (P23)

Furthermore, given the integrated nature of programming, subsequent weeks build on the current week’s topics e.g. using single letter variable names like ‘i’ and ‘j’ for incrementing a ‘loop’ and this may be seen to exacerbate the feeling of being further ‘lost’ in material that wasn’t understood in the first place.
The data suggests a common context across the four data collection sites with lecture, laboratory, and project work being undertaken that requires a certain level of concept accommodation to complete project and continuous assessment tasks. Issues reported by participants relate to the exact syntactic demands of programming:

'It has "if this, blah, blah" then switch does one of them and you have to figure out which one. It's the Java language, you have to know everything (extent) to go and do it. In VB you have to know the code as well but it's more straightforward' (P18)

'It becomes a little bit clearer but you are still saying, how does this work? You have a left side and a right side. You are supposed to put the constructors into this compiler. You can't just throw in any stuff. It has to be set out the way the compiler understands' (P25)

'The dot stuff confuses people. The first time I saw it I said that's just weird. I thought why isn't it just print and then the line but then I got that, it's like a method of a class of a package. It was off-putting at the start. I always want to know why something is that way. Anyone who's like that found it off-putting wondering why it is that way. I suppose I'll have to accept that there will be stuff that you won't understand. You can kind of ignore the structure and just look at the statements inside it. It was easier to accept the structure as that's the way it was' (P29)

The data above illustrates the way a state of mind can be positive or negative depending on the type of student we are dealing with. Whilst some students may get dismayed and bewildered at the need for exactness, others may either gradually come to terms with it or see it as a series of integral nuances of the language that have to be accepted.

Another contextual issue that emerged related to an awareness of the progress or lack of progress by others. A student who may feel bewildered by the material as they sit
in lectures may have this sense exacerbated by focussing on the superior progression of others [AOO]. For others this may be a slightly more positive experience where they focus on the fact that they are not the only one struggling:

'I seem to get errors that everyone else doesn't, you need static here, a colon here. So it's just going wrong, it's just going wrong for me, disaster!' (negative) (P2)

'You realize that you're not the only person who is struggling with it. Everyone's struggling, trying their best' (P19)

'Some people just hear it once and can do it straight away, I have to sit down and study it for an hour, 10 pages of notes, studying them for an hour to get it embedded into my brain, so I could use it properly' (P19)

'If other people are having similar problems you can listen into their same problem' (P45)

'You want to get over it and you have to get over it because you know that other people go through the same problems' (P22)

Examination of the data in appendix 2 will also uncover a category of student who whilst aware of the difficulty of others and may empathise with it to an extent will just forge on ahead working at their own satisfactory pace, in which case awareness of others [AOO] is neither a positive nor a negative state of mind:

'I'm conscious that people need to learn and haven't had the same amount of experience as me' (P20)

Finally, another contextual factor that may influence a positive or negative state of mind relates to the feeling of pressure or pressure build-up [PB] where the intensity of this pressure can affect the student:
‘It’s not like other subjects where you can learn a bit and it will be OK, with this it just seems to be all at once, but before that its completely over your head’ (P29)

‘At the start it’s OK but then it gets more complicated and you start adding in pointers and stuff and it gets worse (intensity)’ (P45)

‘It’s time, there’s not a lot of time to practice, you’re in college so much. Just so much to do as well with the laboratories, you always have problems. If there was one where you could work on stuff you are not sure on that would be good. It’s all about time, there’s too much to cover (P23)

‘We had 1 hour lecture and 4 hours laboratories per week. Tasks were given to be done during the tutorials time and this work was accomplished for Java Portfolio. The problem was that I could hardly finished before the next tutorials which means that I found myself overloaded by tasks and studies of the weeks’ (P4)

In the data some participants describe how the pressure may start to manifest itself at a certain stage e.g. after ‘if’ statements or when pointers or object-oriented programming emerge on the syllabus.

As regards intervening conditions the data suggest that certain moments in the course of the programming syllabus can have a positive or negative effect on the student’s state of mind. In terms of negative effects, encountering a certain topic may cause the student to be confused [CONF] or bewildered [BW]:

With functions I just didn’t get them at all and I said I’ll get it all sorted out later but just didn’t bother and it took a whole project to figure it out. The passing of functions and the parameters completely confused me’ (P42)
'I think it was hello world, I was like, this is going to be grand, you know, and then we started getting into methods and arrays and I was lost, in a world of my own' (P9)

The above data suggests that for some students there comes a time where a certain topic is covered that has an impact on their state of mind in programming. This may come earlier for some than others e.g. an early stage might be coverage of ‘loops’ whilst for others it may be later when pointers or advanced object-oriented concepts are encountered.

Discussions in interviews and focus groups (as well as professional teaching experience) indicate that a sense of other people experiencing the same difficulty can have a positive effect on students. In particular, this seems to instil a sense of solace and an incentive to persist [P] with hard work:

‘You realize that you’re not the only person who is struggling with it. Everyone’s struggling, trying their best’ (P19)

‘A lot of people in the space of a few weeks got lost and myself, I’m just starting to come back into it (positive)’ (P10)

‘You want to get over it and you have to get over it because you know that other people go through the same problems’ (P22)

With this phenomenon we can see varying action strategies. In dealing with confusion [CONF], participants strategies range from skipping over [SO] the confusing stuff to persisting [P] by ‘racking their brains’ until it becomes clearer. Some participants experiencing high levels of self doubt [SLFDBT] describe working their way through material as best they can but with little reward, which in turn may serve to exacerbate negative feelings:
Chapter 6

Data Analysis

'Algorithms and stuff are OK, I can sit down and work my way through them. But when the lecturer is going through loops and stuff like that I try to follow but sometimes I just can't. I say to myself what's going on?' (P42)

'When you try and put it all together and you just can't get it working, you just feel like should I be doing this? Why am I doing this? In an exam when it's actually given to you it's grand but when you have to write it yourself it's hard [D]' (P23)

This notion of self doubt is also described by Perkins et al. (1989) who indicates that 'such learners are most likely to feel unsure of what they are doing, harbour fears about handling the machine and hold doubt in their ability' (p.266). As mentioned earlier awareness of others [AOO] can range from positive or negative. The data indicates a number of positive action strategies in this regard from making a concerted effort to not allow oneself to fall behind like others have done to 'listening in' during laboratory sessions to explanations given to others who are experiencing the same difficulties. The former has been experienced by the author on many occasions during professional experience in the programming laboratory and is welcomed as a desire on the student's behalf to progress using whatever means possible. In some cases, the author has found that some students prefer to do this as they are too shy or embarrassed to put their hand up in the classroom and seek the lecturer's attention. In dealing with pressure [PB] participants describe attempting to 'keep up' or 'keep on top' of the material given the building-block nature of programming.

The outcomes uncovered in the data appear to be dependent on whether participants see these various states of mind as a positive or negative. For example, confusion [CONF] may be viewed as challenging [CHALL] or as being 'daunting' or 'intimidating'. Participants who view these feelings as a challenge and who persist [P] with the material recount positive outcomes like 'getting there slowly but surely', material 'becoming a little bit clearer' or 'it all comes together'. On the negative side, participants describe 'losing track', being 'lost', 'stuck' and being 'overloaded' by the programming material.
To summarise, the data suggests that participants experience many states of mind that can depend on a variety of conditions. States of mind like being confused, pressurised or aware of others may be seen as challenges to be overcome and result in positive strategies like persistence and seeking assistance. On the other hand, confusion and negative feelings about the perceived superior progression of others may cause the student to feel bewildered, disillusioned and isolated. Finally, depending on the nature of the material, the student may oscillate between positive and negative states of mind.

6.5 SELECTIVE CODING

As highlighted in chapter 3 and the previous chapter, selective coding entails identifying a central phenomenon and relating central categories to it using statements of relationships. Strauss and Corbin (1998, p.147) highlight the criteria for choosing a central category:

1. It must be central, i.e. all other major categories can be related to it.
2. It must appear frequently in the data. This means that within all or almost all cases, there are indicators pointing to that concept.
3. The explanation that evolves by relating the categories is logical and consistent. There is no forcing of data.
4. The name or phrase used to describe the central category should be sufficiently abstract so that it can be used to do research in other substantive areas, leading to the development of a more general theory.
5. As the concept is refined analytically through integration with other concepts, the theory grows in depth and explanatory power.
6. The concept is able to explain variation as well as the main point made by the data, that is when conditions vary, the explanation still holds, although the way in which a phenomenon is expressed might look somewhat different. One should also be able to explain contradictory or alternative cases in terms of that central idea.
Furthermore, in selective coding, a ‘storyline’ can be generated that narrates the categories and their relationships (Strauss and Corbin, 1998). In this study, the central category began to emerge early on in the data collection and analysis. This is consistent with the experiences of Strauss and Corbin (1998):

By the time the researcher starts to think about integration, he or she has been immersed in the data for some time and usually has a “gut” sense of what the research is all about. (p.148)

![Diagram showing the central category and its relationships with other categories.]

Fig 6.10 Central Category

Whilst participants described many experiences and phenomena, the data kept pointing to one recurring central issue in the mind of the author and that was the *challenging nature of programming*. For this reason, the challenging nature of the programming subject was considered to be the central category as illustrated in fig 6.10.
The central category or phenomenon as illustrated in fig 6.10 was chosen because it fulfilled the criteria as described by Strauss and Corbin (1998) in the last chapter where it essentially subsumes or ties all of the other phenomena together. In fact, the code challenging [CHALL] actually emerged in the data as an in-vivo code:

'The pace [MD] is challenging' (P14)

'It's challenging, a lot of new concepts and ideas. I find myself making silly little errors, leaving out curly brackets and stuff' (P18)

"In programming you have to learn how to make your own stuff up, your own programs. Other subjects are just notes that you have to memorise and write down' (P18)

'It's something new and challenging' (P19)

'I knew the programming was going to be a massive (intensity) challenge, it's kind of something that has to be worked at really hard [P]' (P25)

'For the most part it's interesting and challenging. The methods caused me quite a bit of grief at the beginning. But overall it is fairly good (nature)' (P29)

'It is interesting. The way it is taught is interesting. It's challenging enough as well to keep us interested all the way (avoid boredom)' (P28)

Memo – This is a very interesting slant on the 'challenging' code. Here challenging is seen as very much a positive experience. In this experience it is challenging enough to keep the students interest.
'Programming is definitely the most intense part of the course. It is the one subject where you definitely have to keep your head above water. If you start falling behind you've lost [BW]' (P37)

'Interesting, it started off not that hard. I kind of got my head around it pretty fast. I didn't have any problems, whereas now as we're getting on to harder stuff [PD] [D] it is getting challenging (extent). I enjoy project work, especially now we are working on stuff that's not small and stupid, which you'll never use' (P36)

Memo – Here we see another dimension to the challenging code uncovered in earlier interviews. This student is finding the programming very manageable but at the same time challenging. For the better performing students it seems this challenging element is one of the things that keep the students interested in the programming subject

Analysis of the data indicates that the challenging nature of programming may be positive or negative. The conglomeration document in appendix 2 also indicates that the challenge may vary in intensity where a perception of low intensity may not be a positive experience for the high-achieving student who requires constant levels of highly intensive challenge to sustain interest and to avoid boredom. For the student not performing as well, a high level of intensity over a large extent of the material may be viewed in a pessimistic light and may result in avoidance strategies, negative states of mind, unconstructive perceptions of programming and poor programming organization. It should be noted that fig 6.10 highlights a very important issue where the double arrow line between 'positive' and 'negative' depicts the situation where students can alternate or oscillate between negative and positive feelings towards the challenging nature of programming. It is the assertion of this research that the central category fig 6.10 with its particular labelling and relationship depiction fulfils the criteria set out by Strauss and Corbin (1998):
Relational statements, like concepts, are abstracted from the data. However, because they are interpreted abstractions and not the descriptive details of each case (raw data) they (like concepts) are “constructed” out of the data by the analyst. By “constructed” we mean the analyst reduces data from many cases into concepts and sets of relational statements that can be used to explain, in a general sense, what is going on. (p. 145)

The selection of this category as the central phenomenon was aided by the advice provided by Strauss and Corbin (1998):

What seems to be going on here? It may make two, three or even more starts to articulate one’s thoughts concisely. Eventually a story emerges. Often returning to the raw data and rereading several interviews or observations helps to stimulate thinking. This tends to work if one reads them not for detail but rather for the general sense, standing back and asking the following questions. What is the main issue or problem with which these people seem to be grappling? What keeps striking me over and over? What comes through, although it might not be said directly? (p. 148)

In terms of this advice, two issues emerged as the driving force behind the choice of central category. Firstly, one shouldn’t be looking for detail but rather the general sense of what is happening. Secondly, the notion of what keeps coming through in the data even though it may not be stated explicitly. These two drivers can be illustrated in the choice of central category. For example, if one examines the difficult [D] category in the conglomeration document in appendix 2 one will see that there are many references to it in the data, in fact far more than explicit references to the challenging nature of programming. One will also see that the ‘difficulty’ code has referenced associations with many other codes both inside and outside its own phenomenon of perception of programming. However the overriding factor here is that not all participants find programming difficult, whereas all participants interviewed found programming challenging [CHALL]. Whilst this may not be said
directly in as frequent a manner as difficulty [D] it represents the overriding sense that one gets from the interviewing the participants. In fact, the challenging nature of programming in first year encapsulates all of the other phenomena including that of ‘difficulty’. In some cases difficulty could be seen as a negative experience imposed by the overly challenging nature of programming, to other cases where its is a positive experience in which exposure to challenging material on a consistent basis keeps high-achieving students from being bored and in turn can increase motivation and interest in the subject in a feedback loop type scenario. Rogalski & Samurcay (1990) also describe this challenging nature of novice programming and the demands it imposes on students:

Acquiring and developing knowledge about programming is a highly complex process. It involves a variety of cognitive activities, and mental representations related to program design, program understanding, modifying, debugging (and documenting). Even at the level of computer literacy, it requires construction of conceptual knowledge, and the structuring of basic operations (such as loops, conditional statements, etc.) into schemas and plans. It requires developing strategies flexible enough to derive benefits from programming aids (programming environment, programming methods). (p.170)

The challenging nature of programming is also reported by Carbone and Mitchell (2001) who refer to the ‘high task demands of coding’ (p.93). Another issue depicted in fig 6.10 is that of oscillation between challenging being both a positive and negative issue for the student. This is referred to in theoretical memos in the conglomeration document as the ‘double-edged sword’ experience in programming:

'The temptation to still overlook memory allocation is still annoying me. I want to get it nailed down. It's the only thing on the course that I couldn't get nailed down. The first few times I saw memory allocation I said I haven't a clue what this is and I'll come back to it at some stage and saying it will be alright, I'll never see it again (importance level)' (P37)
Memo – This is very interesting, just after stating that they deliberately avoided the skipping over strategy, the student subsequently states that the temptation to employ this strategy is ever present. This presents to me another example of the higher level concept of the double-edged sword nature of programming where some students oscillate between various states of mind and strategy employment throughout the programming course. The self-justification employed where they say that they’ll never see it again is interesting. This brings up another issue in relation to concept frequency. In particular, skipping over a less frequent topic like memory allocation may have less disastrous consequences than skipping over assignment statements where assignment statements are used in practically every program the student will ever write. Given the building block nature of programming, the earlier the skip-over strategy is employed the more dangerous it is for the student’s progress (consequence).

‘I like programming when it works but if it doesn’t you are in big trouble, you have no way of catching up’ (P24)

Memo – Once again we see an example of this double-edged sword that programming seems to present, affability for the subject mixed with feelings of doubt and bewilderment

‘It has definitely been difficult, I wouldn’t say I was struggling with it, but it’s difficult to get into it. Overall it has been enjoyable [ENJOY]’ (P39)

Memo – Once again we see this double edged sword where the student experiences difficulty yet enjoyment

‘At the start it was OK but then it got harder, but the fact that I liked it [ENJOY] and kept at, it’s OK again’ (P22)
**Memo** — here the student’s persistence is driven by his/her liking for the programming subject. We also see that they are alternating between states of enjoyment and not being comfortable with programming, i.e. an instance of the double-edged sword again

‘A lot of people in the space of a few weeks got lost and myself, I’m just starting to come back into it (positive)’ (P10)

**Memo** — this student seems to be able to come out of temporary glitches successfully

‘I wish I started the way I feel now. I’m sorry this hasn’t clicked earlier. Maybe it takes somebody a bit longer (speed)’ (P25)

These instances of data all relate to the double-edged sword concept of struggling in their mind with the challenging nature of computer programming where they find themselves oscillating between positive and negative states of mind, strategies etc. Here we see instances of participants getting lost temporarily and coming ‘back into it’, and the perception of programming transforming from something that’s difficult into something that is ‘OK again’. The last piece of data encapsulates a feeling that emanated from many participants where they regret not having the same enthusiasm for programming at the start as they do now. Here the participant regrets not working harder and given the building block nature of the programming subject, lack of application to earlier material has negative consequences on later material.

### 6.5.1 Storyline Memo

According to Strauss and Corbin (1998), developing and presenting the central category entails making use of diagrams, writing the storyline and reviewing memos. As highlighted in the previous chapter, the category association diagramming technique was developed by the author to both explicate the theory in an intuitive fashion and facilitate the development of the storyline. In this regard, fig 6.10 depicts the category association diagram for the central category. At this stage in the research
Chapter 6  

Data Analysis

Strauss and Corbin (1998) tell us that we have a grasp on the essence of the research and are moving from description to conceptualization. As stated in chapter 5, the category association diagram is developed in such a way that it tells the story in a similar way to which an entity-relationship diagram tells us in a clear and succinct manner a story about the core entities in a system and their relationships to each other. The following narrative presents the storyline memo (using the format illustrated by Strauss and Corbin, 1998, p.150).

**Storyline Memo.** The challenging nature of programming may determine one of two types of dealing strategies employed by the student. If challenging is seen in a positive light the student may employ one or more types of *progression strategies*. The most frequently occurring progression strategy is ‘persistence’ [P] where the student goes to varying *extents* to progress with a task or concept. This persistence may vary in duration depending on how well it works for the student. The *type* of persistence employed by the student may be random or planned. In the former the student will persist with any means available to them whilst the latter may entail a prescribed set of steps, lecture notes first, then implementing ‘code examples’ [CODEX] from lecture notes, then textbook examples etc. The student may seek assistance from either their peers [PA] or their lecturers [LA]. Peer assistance may either be required to an *extent* where the student either wants a full explanation of the task/problem or a partial explanation in a scaffolding manner which enables them to progress themselves. Furthermore, the *nature* of peer assistance may be *one-way* (where the student is getting help from a more able peer) or *reciprocal* (where the students help each other by working through problems together by bouncing ideas off each other). In a similar manner, the student may seek ‘lecturer assistance’ [LA] which may either give a partial or full explanation of the task/problem. When working through problems the student may begin by employing the known bits [KB] of code they are already familiar with and build on these gradually in an attempt to get the task or problem solved. This
approach may have varying levels of effectiveness. When faced with a problem or part of a problem, rather than give up the student may employ the strategy of ‘guessing’ [GUESS]. The nature of this guessing may be random or planned. This is also referred to as the ‘trial and error’ approach. Depending on the student, the outcome of a successful guess may entail the student backtracking through the solution to see why the guess worked, i.e. what was the crucial pint in the code that made it work. Other students may be happy that the guess worked and just move on to the next task. Finally, the extent to which these progression strategies are employed may be driven by the desire for ‘accomplishment’ [ACC]. This feeling of satisfaction at getting a task complete may vary depending on the level of accomplishment achieved as well as the amount of effort employed and interest exhibited by the student and the amount of time they actually spent on the task or problem.

If the challenging nature of programming is seen in a negative light the student may avoidance strategies. In some cases, given the perceived overly-challenging nature of programming the student may decide to consciously ‘disengage’ [DISENG] from varying amounts of the material/concepts on the programming syllabus. This may entail disengaging from part or all of a concept e.g. they may understand functions or methods to some degree but disengage from the portions of the concept they cannot get e.g. parameter passing. The more challenging the student finds the material the more frequently they tend to disengage from concepts or material on the syllabus. In terms of disengagement, the students use terms like ‘drifting’ and ‘switching off’ to explain this phenomenon. An alternative avoidance strategy relates to students ‘skipping over’ [SO] material as they progress through a topic or concept. Here, rather than disengaging, the student skips a topic that doesn’t make sense and may come back to it at a later stage. The duration of this strategy determines how long the concept is skipped over for and this is reported as varying between ‘days’ and
‘weeks’. Material or concepts that are deemed of low *importance* in terms of future use (where the student perceives them to be ‘minor’ or ‘never seen again’) may be skipped over indefinitely. Depending on which type of avoidance strategy is chosen, avoidance of core concepts may cause the student to ‘fall behind’ or avoided material coming back to ‘haunt’ them. For example avoiding parameter passing may come back to haunt the student when it is deemed an integral part of project work.

The challenging nature of programming may influence *programming organization*. In this regard, students may possess *prior knowledge* in the form of ‘existing schemes’ [EXSCH] or ‘prior programming knowledge’ [PPK]. From the positive perspective, the student may be able to extend their existing knowledge and schemes to meet the challenging demands of programming. From a negative perspective, they may find that their existing schemes in no way *match* those required for programming or the existing programming knowledge they have is limited in its *usefulness* for the current material they are studying. Students who view programming as overly-challenging may struggle with the delivery of the material [MD] and consider the *pace* as being too fast and the *depth* of the material as being too deep. Those who are positively challenged may find the depth and pace at a level that stimulates and retains their interest in the subject. In terms of *accommodation* of the programming material concepts, we see both positive and negative aspects. From a negative perspective, students may experience a considerable amount of time between being exposed to a concept e.g. ‘arrays’ and it actually ‘sinking in’ [SI]. In some cases, the student may not be consciously aware of this until they start to struggle with a task. Students may take a considerably longer time to accommodate a concept than envisaged by the lecturer. Furthermore, students may find the key concepts may not ‘click in’ [CLK] quick enough in terms of their assessments and tasks whilst others experience not
being able to ‘get their head around’ [GH] the whole concept of programming. From a positive perspective, some students will find that key programming concepts like ‘loops’ and ‘arrays’ click in pretty quickly in a ‘eureka’ type moment after they persist [P] with the material for a certain period of time. The organization of programming material may be assisted by *programming examples*. From a positive perspective, some students may see the ‘code examples’ [CODEX] in the lecture notes as being clear and illustrative and possessing a high level of *explanatory power*. On the other hand, students may find the code examples [CODEX] supplied in lecture notes and text books as ‘ineffective’ [IE] and look for alternative sources e.g. internet for what they consider clarifying examples [CLAREX]. In some cases, these examples are small and succinct and illustrate one concept clearly in a way that is familiar to them. Finally, for students who are both struggling with programming material and those who are progressing at a very high level we may see utilization of ‘self examples’ [SE]. In the case of the struggling student who is overly challenged they may develop an example that relates to a hobby or an interest in the hope that it will explain how core concepts work or rejuvenate their interest for programming. The student who is having no trouble with any of the material may assign themselves their own examples or tasks to keep the challenge up to a level that sustains their motivation and avoids boredom.

As the student progresses through they syllabus, their *perception of programming* may be influenced by whether the challenge is viewed from a negative or positive light. Many students consider programming to be a ‘difficult’ [D] subject where they refer to it as ‘hard’, ‘tough’ and ‘tricky’. Difficulty may be viewed as a positive or negative experience, where some find it ‘difficult but enjoyable’ where others find some topics ‘extremely difficult’ and ‘too hard’. For some the *extent* of the material they find ‘too hard’ may determine their *state of mind* as they progress through the syllabus.
From a negative standpoint, the overly challenging perception may result in programming being perceived as an ‘intimidating [INTIM] subject as compared to the other first year subjects. For a good number of students, programming is perceived as alien where it is composed of ‘alien concepts’ [ALC] and unintelligible ‘words’ [W]. For some students this alien perception is seen in a positive light where new unfamiliar/alien object-oriented concepts like ‘methods’ and ‘classes’ (which they haven’t seen in other programming languages e.g. VB) are challenges that can be overcome with ‘persistence’ [P] and hard work. For others, failure to come to terms with alien concepts and unintelligible words in the lecture notes that do not make sense may result in negative states of mind (such as self doubt [SLFDBT] and bewilderment [BW]) and negative dealing strategies (such as skipping over [SO]).

The perspective on the challenging nature of programming may determine one of a number of states of mind. Whilst these may appear as predominantly negative, this may not necessarily be the case. For instance, a student who is comfortable with the programming subject may find certain parts of the subject ‘confusing’ such as System.out.println in Java where the ‘dot stuff” can appear initially confusing. However this confused state of mind may not last for long as the student may find the material ‘clicking in’ [CLK]. Alternatively, they can see it as integral part of the nuances of the language that just have to be accepted. From another positive standpoint, awareness of the negative plight of others [AOO] whilst may cause the higher achieving students to reflect and comment on this situation, it doesn’t seem to affect their own progression determination. Furthermore, students who are working hard to keep their head above water may use the notion that ‘we are all in the same boat’ to push themselves forward. On certain occasions some students may doubt their own ability and presence on a programming course [SLFDBT]. For
some, this may be cause by mild sense of awe and wonderment at the alien nature of concepts but it is overcome with diligent work and application. On the other hand, students who find the challenging nature of programming as negative may find themselves ‘confused’ [CONF] for considerable durations of time. Furthermore, the perceived rapid pace of lectures and labs may leave the student in a state of ‘bewilderment’ [BW] at the apparent onslaught of overly challenging material. In some cases, a number of students may experience a sense of collective bewilderment. Furthermore an over awareness of the superior progression of others in their class [AOO] may cause a feeling of ‘self doubt’ [SLFDBT]. High levels of doubt over a prolonged duration may have negative implications for the progression of the student. Finally, the pressurised [PB] nature of programming may be viewed as too challenging where the student is constantly exposed to a build up of complex material and concepts, many of which go over their head and eventually they become overloaded. Alternatively, ‘pressure’ may be viewed as an integral part of the challenging nature of programming and is an impetus for considerable time investment and hard work that pays off with an eventual understanding of the material.

Finally, the challenging nature of programming may alternate as both a negative and positive experience. Some students may temporarily oscillate between negative and positive states of all of the phenomena depending on how they experience the material at any given time. In this regard students tell us how they keep on top of things, ‘come back into’ the work ethic and experience a resurgence of interest after periods of difficulty and disillusionment. For some, the timing of these states is crucial.

The above storyline memo gives an overall theory of how the central category impinges on the phenomena uncovered in axial coding. What may or may not be obvious from this storyline is the infinite amount of complex interactions between
categories that transcend the phenomena boundaries. This is consistent with the experiences of Strauss and Corbin (1998):

Although not necessarily written as "these conditions are associated with this phenomenon or process" or "this action leads to this outcome", relationships such as these are implied. Also, notice that the relationships are not written in a cause and effect fashion. The paths of associations are more convoluted than direct, with all sorts of variables entering into the analytic picture to influence the path of action. (p.150)

For example, a negative perception of programming as being overly difficult [D] may cause a negative state of mind such as self doubt [SLFDBT] which in turn may result in an avoidance strategy of disengagement [DISENG]. This suggests that underneath the diagrammatic depiction of the central category in fig 6.10 there lies a complex network of interacting categories that cross phenomenon boundaries. Strauss and Corbin (1998) relate to this issue by indicating that 'sorting categories becomes more and more difficult as cross-relationships among categories may evolve' (p.154). As highlighted in chapter 6, care is taken to use the term 'may' in labelling the relationships between the central category and phenomena. This indicates that the phenomena may or may not be at play in a given situation and thus is consistent with the Strauss and Corbin (1998) assertion of not explicitly stating cause and effect.

6.6 LEARNER TYPE CATEGORISATION

One of the objectives of this study was to analyse the theory developed with the aim of generating a categorisation or taxonomy of novice computer programming learner types. The literature study in chapter 2 described various studies that resulted in categorisation of novice programmers such as 'stoppers' and 'movers' by Perkins et al. (1989) or 'strugglers' and 'averages' by Jenkins and Davy (2000). These classifications are presented to the reader in the form of a continuum, as is the transition of 'novice' to 'expert' described by Winslow (1996). Furthermore, analysis
of each of these studies entails isolating certain points on the continuum and assigning them a label. The experience gained in this project suggests that a set of researchers working in isolation on the same data might each come up with a different set of labels representing various points on the learning continuum. This assertion is based on the fact that abstraction to higher order categories and phenomena is an individual undertaking. For example, the 'perception of programming' phenomena in this study might be referred to as 'attitudes towards programming' by another researcher. In this regard, it seems that it is important to focus on the message conveyed in terms of the learner type and not its label. In addition, given the inductive nature of this study, this message must reflect the data. In this study, the existence of properties and varying dimensions was used to facilitate categorisation. It should also be noted that whilst isolating points on a continuum is a convenient method to present variation, these points are only a single snapshot of a complex set of interacting categories. The non-static nature of a continuum of this type is also referred to by Perkins et al. (1989) where they indicate that 'they image of a continuum in a way is misleading' as it suggests 'a distribution with most students in the middle' (p.266) whilst in fact this may not be the case. Naps et al. (2003) concur with this assertion when developing learner type categorisation when they indicate that whilst developing a learning taxonomy on a continuum where each point reflects different levels of learner engagement, it is wrong to 'consider this as an ordinal scale' (p.142). They have found (as has been found in the data in this study) that many categories overlap and they represent this using a Venn diagram. The overlapping of data amongst categories in this study is clearly evident in the conglomeration diagram of appendix 2.

Given this discussion above, it seems logical that any classification of learner types should reflect the challenging nature of programming as highlighted in the central category. Furthermore this classification should reflect the various codes and categories that emerged from the data that range from positive experiences such as enjoyment and accomplishment to negatives ones such as disengagement and bewilderment.
Fig 6.11 illustrates the learning continuum that emerged from the various levels of coding, namely open, axial and selective respectively. It shows that all points on the continuum are influenced by the challenging nature of programming (i.e. the central category). Furthermore, a cursory glance at fig 6.11 indicates that learners existing to the left of the continuum find programming a negatively challenging experience whilst those to the right experience this in a positive light. The ‘oscillating’ label relates to how the data indicates that some learners alternate between positive and negative attitudes towards the challenge presented by programming. The remainder of this section will analyse the selected learner types on the continuum and discuss them in terms of the phenomena uncovered in axial, open and selective coding.

6.6.1 Disengager
The disengager\(^1\) (the extreme leftmost level of the continuum) sees the challenging nature of computer programming in a negative light. The states of mind they are likely to encounter are likely to be overwhelmingly negative. They may encounter confusion on a large part (if not all) of the programming material and this confusion lasts for practically the full duration of their participation on the course. They find it difficult to deal with having to know ‘so much’\(^2\). In addition, this may be matched with a state of bewilderment where they experience the feeling of being totally ‘lost’ and in worst extremes this may cause a sense of isolation, i.e. as one participant put it ‘I was lost, in a world of my own’ whilst another indicates ‘I say to myself what’s going on?’.

\(^1\) The origin of this categorisation emanates from the ‘disengage’ category uncovered in data analysis.

\(^2\) Direct quotes in this section are taken directly from the data as presented in the conglomeration document of appendix 2.
Furthermore, it is likely that constant feelings of *self doubt* will exist over the full duration of their participation where the learner will indicate things like ‘not having a clue’ about what is going on and the constant feeling of being ‘stuck’. This self doubt may cause the learner to ask themselves the questions like ‘should I be doing programming?’ or ‘am I on the wrong course?’ on many occasions. This self questioning or doubt may also be exacerbated by a negative *awareness of others* where the learner believes they are the only one experiencing extreme difficulty at such a level e.g. ‘I seem to get errors that everyone else doesn’t, or ‘it’s just going wrong for me’. These states of mind are consistent with the notion of ‘causal attribution’ as described by Zimmerman (2002) where attribution to limitations in ability may have a damaging effect on student morale. Bandura (1994) relates to this as the notion of ‘self-efficacy’. In particular non-efficacious students ‘dwell on their personal deficiencies, on the obstacles they encounter, and all kinds of adverse outcomes rather than concentrate on how to perform successfully’ (Bandura 1994, p.72). Frequently, this causes them to lessen their effort and give up quickly. This situation may be exacerbated in computer programming, a discipline that is based on a building-block approach where early concepts need to be mastered before subsequent tasks can be attempted. Failure to master the earlier tasks can sometimes cause the student to become quickly disenchanted and deem the acquisition of programming skills as an insurmountable challenge. The final issue which may affect the state of mind is the intensity of pressure they encounter. In this regard, they may find the *pressure* build up as being too much which may serve to exacerbate feeling of being totally ‘lost’.

The data suggests that the disengager will experience a negative *perception of programming*. In particular, they see the nature of the *difficulty* of programming in a negative light as something that is very ‘tough’ and ‘hard’ where they are simply not grasping the material or ‘catching it’ in lectures. This experience has also been described as *bewildering* where a learner may find himself/herself ‘sitting in lectures scratching my head and I wasn’t sure what was going on’. This experience is likely to exist for the full duration (e.g. ‘since the beginning’) of their participation on the course over a significant amount of the material. In addition, much of the material encountered will be viewed as *alien* containing a lot of *words* which may ‘mean
nothing’ to the learner and failure to come to terms with it may cause them to feel 
intimidated by the overly challenging nature of the subject.

In terms of prior knowledge it is unlikely that the disengager will have prior 
programming knowledge nor will they possess existing schemes that will facilitate 
understanding of programming concepts. It is likely that they will find that pace of 
material delivery too fast and the depth that is gone into with respect to each topic is 
too high. In terms of accommodation of material, the disengager may begin by 
looking at the notes but nothing seems to ‘make sense’ to them with the net result of 
little or no material actually sinking in nor even small concepts like assignment 
statements clicking in. This may be matched with a general lack of ability to get their 
head around the whole concept of programming in general even with rudimentary 
material like input/output, variables and data types as well as simple sequencing of 
statements. Finally, it is unlikely that any code examples provided to the disengager 
will have a level of explanatory power that facilitates understanding. In this regard, 
they will see the vast majority of code examples as ineffective at getting the core 
concepts across to them.

Finally, with respect to dealing strategies the learner may engage in some forms of 
progression strategies such as some level of persistence on a random basis. However 
given the lack of material being accommodated this persistence may be only done on 
a partial extent (perhaps just looking at lecture notes and not using other resources) 
for a short duration. This may be interspersed with some form of assistance. If this 
type of assistance is peer-based it will be one-way in nature where the learner is likely 
to look for the full solution (full extent) rather than small bits in a scaffolding manner. 
Given the lack of success with limited implementation of progression strategies the 
learner may turn to avoidance strategies. This may begin with skipping over material 
that doesn’t make sense to them. If this results in skipping over a large amount of 
material over a long duration the learner may decide to disengage from working at the 
programming subject for good where they become ‘disinterested’ and eventually 
decide to ‘switch off’. This behaviour may often result in the student withdrawing 
from the course is reflected in the high withdrawal rates in first year computer 
programming reported in recent studies. In particular, Woszczynski et al. (2005) 
describe withdrawal rates from 40.5% to 52% in various studies they examined.
This category of learner type appears to have significant overlap with the 'extreme stopper' as described by Perkins et al. (1989) and illustrated in chapter 2. In particular these students 'consistently adopt the simplest expedient and just stop' (p.265). In addition they also indicate that 'some stoppers become so disengaged that they learn very little' (p.267).

6.6.2 Slogger
The slogger as illustrated in Fig 6.11 is depicted on the negative side of the continuum. This category represents a learner type who finds the challenging nature of programming as a predominantly negative experience but persists in trudging through it with some minor levels of success (for example, one participant indicated they had the general gist of 'loops' but wouldn't consider themselves a great programmer). In fact, in grounded theory parlance this represents an in-vivo category where it was taken from the data in the case where one participant indicated that the programming subject 'requires a lot more sitting down and "slogging" through the work' In some cases, this decision to 'slog' through the syllabus may be driven by a greater goal e.g. to get through first year and concentrate on non-programming streams in subsequent years.

In terms of state of mind the learner may find substantial amounts of material confusing and this may be exacerbated by the need 'to know everything'. Here the all or nothing demands of programming in terms of syntax and semantics can heighten confusion and bewilderment. This bewilderment may be initiated when the transition from simple programs like 'hello world' to more complex material like arrays and loops are encountered. In some cases, this may entail sitting in a lecture 'wondering what the lecturer is talking about' and a constant feeling that 'it doesn't get any easier'. In addition, the learner may experience a significant level of self doubt and consider themselves 'not comfortable enough' with the material. In some instances

---

3 Defined in http://www.thefreedictionary.com/slogger as someone who makes (one's way) with a slow heavy pace against resistance.
4 This is consistent with the notion that programming is essentially an 'all or nothing' subject given the precise demands of syntax where a semi-colon or a bracket in the wrong place will cause the program not to execute.
5 This is the classic opening programming in text books where students are shown the code necessary to output the message 'hello world' to the screen.
the slogger may hold out the hope 'that somewhere along the line I'll get an understanding of this' and at the same time ask himself/herself 'when is it going to happen?'. In some instances, they may console themselves in the fact that they are not 'the only person struggling with it' and in some cases there may be a large 'consensus in the class' that the programming subject is presenting difficulty. In terms of a negative awareness of others, this learner type contrasts the amount work they have to do as compared to others in their class e.g. 'some people just hear it once and can do it straight away .... I have to sit down and study it for an hour to get it embedded into my brain'. Here, the learner may experience pressure with a large amount of programming concepts going 'over their head'. This is exacerbated by the building-block nature of the programming subject of which the learner is aware e.g. not succeeding in a task 'makes it very difficult to do other tasks because a lot of programming comes hand in hand'.

For the majority of the time the slogger may experience a negative perception of programming. The learner finds the programming subject 'hard' and 'very tough' and 'extremely difficult to get a grasp of'. The learner, despite their hard work may find themselves 'struggling since the beginning' of the course. Rather than seeing programming as an enjoyable experience they see it as intimidating where there is a large block of conceptual material that has to be understood and 'interlinked'. In addition they may find a large amount of the material as 'completely alien' to them and as a result they find themselves 'struggling' to grasp new concepts as they emerge. In some cases, programming material (i.e. lecture notes, lab sheets and handouts) may appear as largely consisting of unintelligible words where the learner may find themselves 'not really knowing what they are reading about'.

As regards material organisation the slogger may have some prior exposure to other programming languages but may get to a stage in the new language (e.g. Java, C++) quite quickly where this prior programming knowledge no longer aids the accommodation of new concepts e.g. object-orientation. In terms of prior knowledge, the learner may have some rudimentary schemes pertaining to conceptual overlaps with the English usage e.g. simple 'if' statements and 'while' loops. In addition, the learner finds the pace of delivery quite challenging and frequently moving onto a subsequent topic without understanding the previous one whilst asking as many
questions as possible to keep their head above water. The use of code examples will be engaged upon with a constant search for clarifying examples that contain a desired level of explanatory power which can result in a number of core concepts being 'set in stone'. A problem that may arise for the learner is when examples presented to clarify a concept are ineffective. This issue can also manifest itself at a micro level where the code examples use single letter variables like 'i' and 'j' and shortcuts like 'sum ++' instead of 'sum = sum + 1'. In terms of a code example, the learner often 'prefers if it was related to things in the real world and not abstract things'. The net outcome of these difficulties is a slow struggle to accommodate programming concepts. The learner may not find the time allotted to each topic as satisfactory in order for it to sink in. Despite constant work efforts the learner may find himself/herself 'coming back' to a topic a number of times and still 'not grasping it'. For some the awareness of this lack of accommodation may only manifest itself when they sit their first assessment. For this type of learner they find that it may take 'ages' 'or 'longer than others' for key programming concepts like 'if' statements to click. Furthermore, this learner may hold out hope that 'somewhere along the line they'll get an understanding'. At a general level, it may take a long duration for the learner to get their head around the exactness and abstractness of programming. In this situation they describe experiences like not getting 'a full grasp' of programming.

The slogger may engage in both types of dealing strategies. However, in this regard, the nature of 'slogging' encountered in the data is typified by engagement in progression strategies. In particular, the slogger will go to various extents to understand the material. In terms of persistence the learner may use as many sources they can to make the material easier to accommodate. Furthermore, persistence will be typified by long durations where the learner 'keeps plodding along' going back over lecture notes looking for morsels of material that can help them overcome their lack of understanding. In some cases, for a given concept this persistence doesn't pay off where the learner ends up in 'still not grasping it' and they end up moving on to the next topic. The learner will also make use of assistance. In terms of lecturer assistance the slogger will more often than not require a full explanation of a topic or a concept especially in cases where persistence has not worked. In addition, they may frequently seek the same type of full explanation on a one-way basis from their peers. In terms of guessing the learner may use this strategy on a random basis by 'typing up
stuff and hoping for the best'. Here, rather than being a strategic/educated guess, the student just types in code without any apparent mental engagement in the material they type. In reality, this may only result in nothing more than electronic transcription. This finding is consistent with those of reported by Perkins et al. (1988):

These haphazard movers like stoppers were not genuinely engaging in the problem, but rather proceeding in an extreme trial and error fashion that evaded any genuine intellectual confrontation with the difficulties presented by the program. (p.159)

In addition, this learner may encounter levels of *accomplishment* which in relative terms may be small but based on the large amount of effort expended by them may appear significant. Finally, in terms of avoidance strategies, the slogger may *disengage* from certain parts of the syllabus that they cannot come to terms with. This may be caused by either the complexity of the material or the feeling of not being able to participate or interact with peers and lecturers. However, a more common avoidance strategy is that of *skipping over* certain material until help becomes available. The importance of the material (i.e. how much it is an intrinsic part of future topics) and the amount of time (duration) it is skipped over may determine whether or not the slogger will progress through the course.

This category of learner resembles the ‘serious struggler’ described by Jenkins and Davy (2000) where they describe them as learner types who will find the course extremely difficult, and who will not pass without significant additional support and encouragement. In this study the additional support manifested itself in the form of lecturer and peer assistance. In particular, the extent of the assistance they get from lecturers and peers is likely to be high where they will require most if not all of a solution to be explained to them on an incremental basis.
6.6.3 Worker

The centre of the continuum in fig 6.11 depicts the learner type described as worker. As with the slogger category, this is an in-vivo term used by a number of participants. In the data we see references like ‘it takes a good bit of work’, ‘it’s hard work but rewarding, at the same time it’s very frustrating’ and ‘I’ve had to work at it, the other subjects have sat OK with me’. Furthermore, fig 6.11 illustrates that the worker learner type may oscillate or alternate between positive and negative attitudes towards the challenging nature of programming (which will be referred to in the remainder of this section as the positive and negative zones respectively). For instance, whilst a level of success or accomplishment on a project or task may move them into the positive zone, a subsequent difficulty with a different task may result in them ‘drifting’ back into the negative zone. This alternation between positive and negative states will now be discussed in terms of the major phenomena.

As suggested above, this learner may be likely to encounter mixed states of mind. They may encounter confusion at some parts of the course where ‘if’ statements may ‘click straight away’ where as ‘methods’ and ‘classes’ and ‘inheritance’ may be ‘difficult to figure out’ and appear ‘crazy’ and a feeling of suddenly thinking ‘you can’t do this’ This confusion may alternate with understanding where some material ‘becomes a little bit clearer’ over time. The worker may experience bewilderment at various times during the course. They may be constantly battling the double-edge sword where they ‘like programming when it works, but if it doesn’t you are in big trouble’. In addition, the learner may be overtly aware of the tenuous nature of programming knowledge where one worker indicates that ‘semester one was alright, I got kind of lost in semester two and had to pick it up, but then coming into semester two it’s like the bar has been raised’. This tenuous nature of programming experience sees the learner alternating between positive and negative states and having to pick up their work rate in order to move back into the positive zone. In terms of self doubt the learner is likely to be aware of their own shortcomings e.g. as one worker indicated ‘I am still having problems with loops now and again, I have the gist but I wouldn’t be a great programmer now’. Another describes ‘I still don’t think it’s easy’ and a feeling of ‘not being comfortable going into my programming assessment’. A worker may find early material like algorithms ‘OK’ and can ‘work their way through them’ but
find that they get to a topic like loops (‘for’ and ‘while’) and find themselves suddenly asking the question ‘what’s going on?’. In addition, in relation to awareness of others the learner may be conscious of how others experience similar difficulties e.g. ‘a lot of people in the space of a few weeks got lost and myself, I’m just starting to come back into it’ and appears to provide a form of consolation and an incentive to continue working. In terms of pressure the learner may encounter initial material as manageable but subsequently ‘it gets more complicated when you start adding pointers and stuff and it gets worse’ and this may be exacerbated by ‘so much stuff to do in the labs’.

As might be expected, the worker is likely to find the perception of programming a less negative experience than that of the disengager or slogger. In terms of the alien nature of programming the learners find themselves ‘thrown in at the deep end’ and having to work harder at programming because ‘it takes a lot more brain power’. In addition they may find ‘sitting down and actually coding hard to get used to’. Here, one notes the less negative term of ‘getting used to’ which typifies a less defeatist attitude of this learner type. In addition, the learner tries to overcome the ‘hard’ nature of alien concepts like ‘switch statements’ by ‘doing more examples’ to ‘make it easier’. The experience of difficulty is also less negative than previous categories. For example, one worker indicates that ‘I am finding this semester tougher, the programming has upped the tempo’. This example crystallises the non-static nature of the programming subject where the learner may find it in a constant state of flux which may result in certain amounts of oscillation between positive and negative zones. In this regard, one learner provided a very interesting insight into how and why they find certain material difficult. Here, they describe ‘if’ statements and ‘for’ loops as being ‘OK’ because they can predetermine the control variables and incremental values of the ‘for’ loop and they can decide on the exact nature of the ‘if’ statement they want to implement themselves. However, they find that they have lost this feeling of being in control because of the perceived complexity with methods. This issue is emerged in the data where for some learners there is a point where they lose conceptual control of the material and how they deal with this after that point may

6 Here a ‘control variable’ is used to control the number of iterations in a loop e.g. for (index = 1; index <=10; index++). This loop will execute 10 times and exit when the control variable ‘index’ reaches the value 10. In programming this is known as a pre-determinable or ‘fixed’ number of iterations.
determine whether they remain in either the positive or negative zone. In cases where the learner is in the positive zone they view programming as interesting and enjoyable. In some cases the learner may feel intimidated by the workload but attempts to put it into perspective and come to terms with it even though it appears to be ‘daunting’.

In terms of programming organisation the learner may have some level of prior programming knowledge. In this case, they may experience some level of difficulty given the amount of conceptual material involved in object-oriented programming as opposed to Visual Basic (VB) which may be considered ‘more straightforward’ in comparison. This learner type may have a preference for examples used in lectures to have a high matching level with their existing schemes and this can prevent them from ‘drifting’ back into the negative zone. The learner is likely to find the pace of material delivery as being quite fast with a considerable amount of depth. This results in a necessity to work at as fast a rate as possible and to do ‘revision’. If this pace and depth isn’t managed properly the learner is aware that they can be ‘pretty much gone for a while’. Furthermore, it may be the case that the pace appears to be slower with the early material but as one learner put it ‘we just flew through arrays’. This may give the impression that ‘it was like all the hard stuff, we just flew through it’. The oscillatory nature of this category is evident when it comes to accommodation. Learners speak about certain concepts being ‘difficult to pick up in the first two weeks’. One worker describes ‘with almost everything at the start I didn’t have a clue what it meant, it involved a lot of work’. Here, self-application and hard work is used to transform the worker into the positive zone for each topic. In some cases, despite hard work, some topics will ‘not fully’ sink in with the learner. In addition, some topics like ‘functions’ and ‘pointers’ may ‘take longer to get the hang of’. The data also uncovered the type of worker for whom key material may click in too late when the assessment for that topic has passed and they indicate that ‘I understand it a bit too late but come to grips with it in the end’. Workers may often spend ‘a lot of time looking at code’ and then find key concepts just ‘click’, this may be matched with a sudden realisation of where they were going wrong, i.e. they see ‘why’ something was going wrong and eventually emerge successfully out of the negative zone. In addition, the learner may continue to have ‘problems in some areas’ and are happier than sloggers and disengagers to accept that they won’t get their head around certain
topics and move on. The oscillatory nature of this learner category can result in seeing 'the flow' of some topics but then coming to something like 'populating an array in a couple of lines of code' and so presenting yet another obstacle. In order to overcome a given task or topic the learner may need a tangible self example that crystallises the programming concept. Furthermore, the learner will attempt to access as many code examples they can get their hands on with respect to a specific topic. The data also suggests that the more of these examples worked through, the easier programming becomes 'in the long run'. In addition, the greater the work effort, the greater the chance of encountering clarifying examples in the midst of numerous ineffective examples.

Finally, in terms of dealing strategies the worker is likely to engage in activities that aid progression. The data suggests that one of the hallmarks of this learner type is that of persistence. In many cases, the learner will use sustained brute force to overcome accommodation issues in terms of programming concepts. Much of the terminology they use reflects a very diligent work ethic such as 'keep your head down and work through it', 'repeating and repeating until it hits home', 'use hard graft', 'find different ways of doing it'. Furthermore, the oscillatory nature of the worker is also evident where one worker highlights 'at the start it was OK, but then it got harder, the fact that I liked it and kept at it, it's OK again'. This is consistent with the experiences of Fincher (2006):

In our experience some students reach a ‘sticking point’ a few weeks into learning to program, where the amounts of new material is increasing yet they have not quite mastered the basics of programming. Persevering through this point is important, most of these students only get bogged down for two or three weeks, after which it seems to get easier! (p.46)

This learner type also avails of assistance. In terms of peer assistance they seek both one-way assistance e.g. ‘I ask other people how they did things’ or ‘turn to people in my class’ as well as reciprocal ‘we sit down and work on it’, ‘you’re helping them and they’re helping you’. This assistance may be partial or full depending on whether the learner wants help in a scaffolding-type manner or is at the end of their tether.
(having persisted for a long time) and wants to see how a task can actually be done. In a similar way, they may seek either full or partial assistance from their lecturer. One strategy that typifies the worker is initial utilisation of known bits of code in order to get some part of a task under their belt and instil a sense of confidence. In the data we see ‘I’d start of writing the bits I did know, I’d troubleshoot to see which part was wrong’ and ‘do the parts you can do first and then you look at the rest’. In some cases, the approach taken wouldn’t result in the most elegant\(^7\) piece of code where ‘it would be probably be a lot longer, I’d kind of stick to what I know’.

In cases where assistance may not be available the learner may incorporate the strategy of guessing. A common approach may be to ‘take out bits’ that they don’t understand and to use ‘trial and error to see what it does’. This incremental type of guessing is planned in nature where code segments from the lecture notes are examined on a ‘bit by bit’ basis in order to use them to accomplish a task in the laboratory. Finally, much of the hard work employed by the learner is driven by the desire for achievement or accomplishment. To a large extent, this factor seems to be lacking in the earlier learner types. Whilst the learner may be struggling on a number of occasions throughout the academic year to stay out of the negative zone, their interest in achieving a certain standard in programming may be quite high. In this regard, workers indicate ‘if you can’t get it done it’s frustrating, but when you get it to work it’s amazing’ or ‘it’s hard work but rewarding, the longer it takes the happier you are when it’s done. This notion of accomplishment can manifest itself in varying levels from getting a simple syntax error fixed to completing a method that performs a relatively complex task. The worker may also employ avoidance strategies to some extent. Here, they appear to have a more deliberate and planned approach to skipping over material. In terms of worksheets, the worker may ‘leave them for a while’ until solutions are posted by the lecturer. This tactic may entail reading through the solutions with the aim of ascertaining how and why they were unable to complete a task. An alternative approach is to leave a task temporarily with the aim of returning it to it later. The temptation for the worker (perhaps in the negative zone) to disengage was described very succinctly as ‘you want to work a lot harder and then as time goes by you get more lazy or something and say to yourself I’ll catch on to it later’. In this

\(^7\) Elegant programs are usually simple and require a few clean and easy steps to achieve the desired task.
regard, another participant indicates that he/she ‘picked it up again at the start of the
semester. I’m interested in it so it’s not that bad’ and this is done to ‘stop it fading
out’. This suggests a strategy to reactivate one’s interest in programming to stop it
‘fading out’. The professional practice of the author suggests that if programming
concepts are not practiced on a regular basis, the concepts themselves and the rigorous
syntax of the language may be quickly forgotten. Here, the learner may be engaged in
a constant battle between liking the subject and letting it get the better of them e.g. ‘in
the first semester it was going over my head, maybe I was letting myself do that but
I’m not now’. These numerous dalliances with the disengagement strategy in a sense
crystallise the oscillatory nature of the worker learner type as well as the double-
edged sword nature of the programming subject.

6.6.4 Manager

The manager learner type is an in-vivo category that reflects a student who, whilst not
achieving a totally full complement of programming skills is able to progress quite
satisfactorily on the course. In contrast to workers and sloggers they appear to be able
to cope or ‘manage’ better with the challenging demands of programming.
Furthermore, they seem to be able to cope with not knowing all the material presented
in first year and attain a good solid base of programming that can be built upon in
subsequent years.

In terms of state of mind the learner may experience some level of confusion but is
able to put it into perspective by being able to ‘accept that there will be stuff that you
won’t understand’ and ‘it was easier to accept structure as that’s the way it was’. In
this regard, this type of learner who is more comfortable with programming can deal
with the confusing material in a different way to those who struggle. They seem to be
able to accept syntax and slightly confusing layout as nuances of the language that
just have to be accepted. In addition, they seem to be more willing to tackle the
‘intricate’ ways that some concepts like ‘methods’ actually work. Whilst the learner
may experience instances of self doubt they seem to be able to put it into perspective.
They speak in terms of being initially ‘awestruck’ by the challenge of programming
but through diligence ‘it comes’ together. In addition, they are acutely aware of their
deficiencies but can accept them e.g. ‘I’m not invincible when it comes to the subject but I’m steady enough’ and with relation to a project ‘there are always a few bugs that you never get fixed at the time’. Whilst being aware of others who are struggling in their class, managers are conscious of it but do not let it impact on them and. In fact, they suggest that this struggle can be made more manageable if diligence and work ethic are adhered to e.g. ‘it’s not too bad if you attend all the classes’, ‘it’s fair enough, manageable and enjoyable’. Furthermore, they may be able to use the achievements of others to spur themselves on e.g. ‘you want to get over it and you have to get over it because you know that other people go through the same problems’. Finally, the manager doesn’t seem to experience any great level of pressure and indicates that if one ‘keeps on top of things and not letting them build up, that’s the biggest thing really’.

The perception of programming of this learner type is primarily positive. They may find themselves ‘struggling’ with the alien nature of programming but eventually ‘figure it out in their head’ or ‘it was hard at the start but I started to get used to it’. Similarly, they seem to be able to adequately cope with the difficult nature of programming where they may find it ‘tough but doable’ or ‘tricky’. On the negative side, the manager may refer to specific parts of the subject that is causing difficulty e.g. ‘recursive functions’ but in the overall scheme of things this is quite small. Given these factors the learner is likely to find the majority of the programming course as enjoyable and interesting.

In terms of programming organisation the leaner may have prior programming knowledge. In cases where they haven’t been exposed to object-oriented concepts before they seem to be able to take it in their stride as they consider it ‘something new’ or ‘different’ that can be overcome. Whilst the learner may find the pace and depth of material delivery quite challenging they are likely to ask relevant questions at the appropriate time to avoid moving on to the next topic without a satisfactory level of understanding. In order to facilitate this process the learner may develop various types of self examples. In this regard, when they encounter a new topic they may ‘think how is the best way this is going to be applied’ or ‘what’s the use for it’. This strategy is consistent with self-regulated learning (SRL) which refers to the process whereby learners, in a systematic manner, direct their thoughts and actions.
towards the achievement of their learning goals (Schunk 2000). In addition, the learner has a positive approach towards using code examples to accommodate a concept where they find programming tasks 'tough but doable' and can be overcome by starting off with 'smaller examples instead of being thrown in at the deep end'. Using this strategy the learner builds up a set of clarifying examples that either 'sticks' with them or enables understanding of the concept to be 'set in stone'. In terms of programming material sinking in the manager seems to be able to cope with concepts not being understood in a short time window and realise that with work and diligence they may be understood over time. When this occurs, the learner may use reflection e.g. 'now you can see at this stage how loops are applicable, whereas first it was confusing' to give them confidence to persist with more difficult topics. This persistence, in many cases pays dividends for the student (where the concept(s) clicks) e.g. 'just sitting it out, working it over and over until it clicks, there is a moment when it just clicks and you understand it'. These strategies in turn may enable the learner to gradually get their head around many of the concepts in programming despite them appearing 'daunting' at first.

In terms of dealing strategies the manager may employ skipping over tactic in situations where the topic is of less critical importance e.g. 'memory allocation' is of less critical importance than 'if' statements where not understanding the former will have fewer repercussions than the latter. In cases like this, the learner may 'come back to it' if they have time, but if not they will not get unduly concerned. In the main, the manager is likely to employ progression strategies. In terms of persistence they may find that perseverance at a task (that initially appeared quite difficult) may result in very positive outcomes, e.g. 'the more you work on them the more you find how similar they are, it's only a few lines of code that are different' or 'once you have done all the examples you will be surprised how far easier the course is going to be for you in the long run'. This type of experience may serve to instil greater confidence in persisting with perseverance and application. The net result of this may build up a sequence of skills that can be used to incorporate new transferable skills e.g. 'the more you practice, the more you can link to previous things, the more you build up, the more you can link back to'. In addition this learner type appears to engage in peer learning where in a reciprocal manner, a group of students sit down and 'bounce ideas off each other'. This may also entail sitting down and collectively analysing a project
in order to facilitate ‘getting a rough outline so that everyone’s on the right track, instead of one person going off on a tangent’. In terms of progressing the learner may implement guessing in a more planned approach than workers or sloggers. For example, as one participant indicated ‘sometimes I might not get a full grasp of how exactly everything works but I’ll understand how to use if and after a couple of different programs using the same thing differently I’ll understand it in time eventually’. This ties into the persistence strategy where material that was initially guessed and was not fully assimilated over a period of time with eventually sinks in after continuous reimplementation in other programs. Over a period of time, this continued diligence in implementing and re-implementing difficult code chunks may result in the development of a significant library of known bits of code. The learner may break a task down into manageable ‘chunks’ or transfer known bits of code from simpler programs into methods of functions as required in larger projects. The perseverance with these strategies is likely to be driven by the desire for accomplishment where whilst the learner may find a task ‘frustrating’ they find ‘when you get it to work it’s amazing’ and this results in a sense of ‘pride’ given the initial perceived complexity of the task.

6.6.5 Achiever

The achiever learner type is one that reflects a student who achieves a high level of competence in programming and progresses without experiencing regular difficulty with programming concepts.

In terms of state of mind the learner is unlikely to experience very much confusion or bewilderment. However, one issue that became very evident was how aware achievers were of the difficulty experienced by others where they refer to instances of ‘my classmates struggling’ or ‘some have been taken aback and left the course’. It may be the case that it is this very overt awareness of the capability of achievers that makes the less capable students even more aware of their perceived inadequacies. It is interesting to see how the same code/concept can be uncovered in two totally different ends of the continuum. Furthermore, despite being aware of this struggle of their less capable peers it doesn’t seem to affect them in any way in terms of their own
progression. This confident state of mind is consistent with the notion of self-efficacy as which is defined by Bandura (1994) as 'people's beliefs about their capabilities to produce designated levels of performance that exercise influence over events that affect their lives' (p.71). Furthermore, according to Bandura (1994) students possessing a high level of self-efficacy approach difficult tasks as things to be mastered, as opposed to, threats to be avoided and this in turn 'fosters intrinsic interest and deep engrossment in activities' (p.71). In this regard, it is evident from the data that achievers exhibit a high level of self-efficacy.

In terms of their perception of programming the achiever sees this in a positive light except in some rare cases where they may become bored with the material. Whilst the learner may experience some level of difficulty this is limited to a certain number of concepts that they haven’t seen before, i.e. alien concepts from object-oriented programming that they haven’t encountered previously in the Visual Basic (VB) language and are thus challenging. This is consistent with the ‘mover’ category of Perkins et al. (1989) who see mistakes ‘as part of the process of programming, part of the challenge’ (p.267). The learner may for the most part enjoy programming once it doesn’t entail covering ‘boring’ or trivial material that ‘you will never use’. Whilst the less progressive learner categories may find much of the programming material alien the achiever identifies specific topics (in the data it is solely object-oriented topics). For example, learners refer to matters such as ‘the whole thing about encapsulation and keeping the different parts of the program separate from the other parts’ and ‘Java syntax has a lot of abstractness’ or in the case where prior VB knowledge exists ‘classes were a major difference to what I had seen beforehand’. Here the learners are isolating classes and other object-oriented issues as a singular alien concept. This differs dramatically from interviews with weaker learner types like disengagers and sloggers where the participants don’t refer to actual programming concepts but rather speak in general terms. It emerged during data collection that students who are more comfortable with programming are less reluctant to explicitly mention actual programming concepts like classes, inheritance etc. In addition, encountering terms like ‘abstractness’ and ‘inheritance’ during interviews served as a beacon to the author that indicated a level of comfortableness with the programming subject. Conversely, this can be contrasted to the lack of confidence of weaker students to use certain terms for fear it would be wrong or misused in the incorrect context.
In terms of **programming organisation** the achiever is likely to have **prior knowledge**. In this regard, achievers may indicate the usefulness of **prior programming knowledge** e.g. ‘at the start I found it very easy because it was mostly stuff I’d seen before and I applied what I knew already to Java’. From a slightly negative perspective they may (in stark contrast to their less able learner types) find the pace and depth of **material delivery** as non-challenging during certain parts of the course especially at the beginning of the year e.g. ‘because I have a background in programming, it’s a bit slow at the beginning’. In this case the student may implement other organisation strategies like engaging in **self examples** e.g. ‘if I find stuff boring I go off and to my own stuff’.

In terms of **dealing strategies** the achiever is likely to engage in solely **progression strategies**. They enjoy the sense of **accomplishment** that can be gained from getting a large challenging project working and contrast it to other subjects that involve rote learning where they find ‘there is no real kick’ out of them. In addition, they appear to utilise **persistence** in situations where they come across something new and given their level of ability it doesn’t take them ‘too long’.

It should be noted here that viewing learning issues in terms of each continuum category is consistent with the approach of Naps et al. (2003) who indicate that ‘each learner falls somewhere on the continuum in each dimension, and their position can considerably affect their learning when different learning methods and tools are used’ (p.147). The implications of these learner categories for teaching practice will be discussed in chapter 7.
6.7 VALIDITY AND RELIABILITY

What are needed are not formulaic approaches to enhancing either validity or trustworthiness but understanding of, and respect for the issues that underlie these terms. We must grapple with them, doing our best to increase our ways of knowing and avoiding ignorance, realizing that our efforts are quite small in the larger scale of things. (Seidman, 1998, p.19)

The concepts of validity and reliability in a qualitative study have been discussed in a variety of ways in the literature (e.g. Chiovitti and Piran, 2002; Dixon-Woods et al; Onwuegbuzie, 2002; Miles and Huberman, 1994; Cohen et al., 2000). This variety can leave the researcher as unsure about which subset to use. In this regard, validity and reliability issues presented in this section will initially focus on how the grounded theorist can validate the theoretical scheme as recommended by Strauss and Corbin (1998) followed by other measures of validity, reliability and rigour as suggested in the literature. It should also be noted that given the fact that validity and reliability should be implicit in a grounded theory study, every effort was made to make the methodological and analysis decisions taken by the author manifestly explicit in the various chapters of this thesis.

6.7.1 Validity

In terms of qualitative research Cohen et al. (2000) define internal and external validity. Internal validity attempts to demonstrate that the explanation for a specific event or a series of phenomena can be sustained in the data. In simple terms they assert that ‘the findings must accurately describe the phenomena being researched’ (p.107). External validity refers to the generalisability of the study to other settings. From a logical perspective, given that grounded theory is a bottom-up inductive research method, one would assume that if methodological rigour and adherence to the demands of inductive enquiry are utilized then internal validity should be apparent in the study. In terms of a grounded theory study, this research has found that internal validity mechanisms are an intrinsic part of the data collection and analysis process.
External validity on the other hand appears to be less important to the grounded theorist who focuses instead on the rigour of data collection and analysis process at the research site(s). It may be the case that generalisability in the form of 'transferability' (Miles and Huberman, 1994, p.279) may be possible if the data collection and analysis is performed across multiple data collection sites. This issue will be discussed shortly in terms of substantive versus formal theory.

6.7.2 Validating and Evaluating Theory

According to Strauss and Corbin (1998), validating theory is not about testing it in the quantitative sense but rather that ensuring that the theory that is generated emerged from the data and represents 'an abstract rendition of raw data' (p.159). This is also referred to as the notion of 'fit'. Furthermore, at the simplest possible level they assert the following:

However, once an analyst explains in detail how he or she arrived at a conceptualization, other researchers, regardless of their perspective should be able to follow the analyst's path of logic and agree that it is one plausible explanation for what is going on. (p.146)

Given the fact that qualitative data analysis entails data reduction the researcher must ensure that no salient material was omitted from the theoretical scheme. At a high level, Strauss and Corbin (1998) recommend a number of ways to validate the theoretical scheme. Firstly, the researcher should go back and compare the scheme against the raw data, in the form of a 'high-level comparative analysis' (p.159). This approach was an inherent part of this study where all instances of a category or phenomena were collected using the technique of 'conglomeration' as highlighted in appendix 2. This was matched by a constant checking of the open coding document over numerous iterations to not only ensure that the schemes matched the data but also that data reduction didn't result in phenomena being omitted. Secondly, they indicate that the researcher can ask the respondents to read the theoretical story and see if it fits their cases. This approach was not possible in this study given the close
proximity of data collection to the final examinations of the participants. In addition to these high-level activities, Strauss and Corbin (1998) also recommend more detailed criteria for both the research process itself and the empirical grounding of the theory. Both of these will now be discussed in the following paragraphs.

According to Strauss and Corbin (1998) it is important for readers of a theory who were not present during data collection and analysis to be able to judge accurately how the analysis was carried out. In this regard, they identify seven criteria which can assist in the evaluation of a grounded theory study (p. 269):

1. How was the sample selected? On what grounds?
2. What major categories emerged?
3. What were some of the events, incidents or actions (indicators) that pointed to some of these major categories?
4. On what basis did theoretical sampling proceed? How did theoretical formulations guide some of the data collection?
5. What were some of the hypotheses pertaining to conceptual relations (i.e. among categories) and on what grounds were they formulated and validated?
6. Were there instances in which hypotheses did not explain what was happening in the data?
7. How was the core category selected? Was this collection sudden or gradual and was it difficult or easy? On what grounds were the final analytic decisions made?

Each of these criteria will now be addressed in the following subsections.

Criterion 1 - Sample Selection

As described in chapter 3 the ‘open sampling’ technique as proposed by Strauss and Corbin (1998) was used. Here, sampling is open to those persons, places, and situations that will provide the greatest opportunity for the discovery of theory. In this study, the open sampling approach took account of actual constraints such as availability of students who volunteered to participate at a time which was least
disruptive to their studies (as described in detail in chapter 4). Given the fact that the research question was quite open and wasn’t preconceived i.e. looking at learning experiences of first year programming students, any student in a first year programming course at a third-level institution was a theoretically rich candidate where they would have represented a unique set of programming-related experiences. Furthermore, participants volunteered for the study after an initial contact briefing session was given by the author to their class during one of their programming lectures. Because the aim of the research was to examine student experiences of learning programming no pre-determined categorization was used e.g. selecting only those finding difficulty with the programming subject. In simple terms, those students who met the eligibility criterion of being on a first year programming course were invited to engage in one or more data collection activities (this is consistent with the approach of Chiovitti and Piran, 2003). In this regard, Strauss and Corbin (1998) advise that the researcher should never become upset by not being able to choose a site or obtain access to theoretically rich sites or persons. Instead they recommend that the researcher should ‘make the most of what is available to him or her’ (p.210). The net outcome in this research was access to a set of theoretically rich participants across four sites all of which provided data that was pertinent to the research question. The only potential drawback related to being limited by the voluntary nature of participant selection. With hindsight, this did not prove to be a limitation given that all participants provided information rich data about the experience of learning programming and as such contributed to the learner type continuum presented in this chapter.

Criterion 2 – Major Categories that Emerged

From the first interview onwards a number of major categories emerged across the four data collection sites. It became evident that the students experiencing ‘difficulty’ [D] were finding that it took a long time for programming concepts to ‘sink in’ [SI]. In terms of material not sinking in one of the first brief memos written by the author pertained to the student recounting a feeling of being unprepared for assessment tasks as a result of inadequate ‘sink in’ time being allowed for. In addition, they seemed to find much of the programming material as alien ‘words’ [W] or terms and described a
strategy of 'skipping over' [SO] stuff that didn’t make sense. The early memos written in relation to these were:

Memo – Here the participant seems to describe a state of bewilderment when encountering the alien nature of the programming subject. It will be interesting if further participants recount this type of experience and how they deal with it.

Memo – Here we see a very interesting strategy where the participant consciously puts material they don’t get first time to the back of their mind and focus on other parts of the course, i.e. no programming parts. This participant is consciously aware that the result or outcome of this strategy is falling behind.

Memo - do some programming students disengage from something if it doesn’t click with them within a certain time frame, it seems important to explore the relationship between persistence and disengagement that is starting to emerge in the data.

It also began to emerge that as the data collection progressed that a number of participants found programming ‘difficult but enjoyable’. In this regard, they seemed to be in a constant double-edged sword type battle with the challenging nature of the programming subject. Memos pertaining to this were:

Memo – In relation to my earlier memo, here we see the participating alternating between states of letting things consciously go over their head and knuckling down and not consciously skipping over material that they are not getting first time. It will be interesting to find if future participants drift between different strategies/approaches or mental states.

Memo – This is becoming a common experience, students finding the material tough but really wanting to understand it. It seems like at times they are struggling with this double-edged sword, where sometimes the toughness just gets the better of the interest and disheartens them. I am left wondering are they really interested in this subject or do they think they have to find it interesting in order to move on in the course or be accepted as a computer science student.
**Memo** – Here we have an example of the double-edged sword experience of programming where the student alternates between states of frustration and enjoyment.

**Memo** – Once again we see an example of this double-edged sword that programming seems to present, affability for the subject mixed with feelings of doubt and bewilderment.

As can be seen from the above theoretical memos and theoretical questions, successive data collection and analysis showed that a large number of participants found the programming subject very challenging but worked very hard using progression strategies like ‘persistence’ and seeking ‘assistance’. This oscillation between positive and negative perceptions of the challenging nature of programming that was witnessed across all four research sites became a significant part of the theory.

**Criterion 3 – Major Events that Pointed to Categories**

The major events that pointed to these emergent categories are clearly discussed in detail in the axial coding section within the causal conditions, context and intervening conditions presented within the paradigm frame for each phenomenon. Furthermore, the conglomeration of appendix 2 illustrates the association between these categories and other categories outside of the phenomenon boundary e.g. fig 6.3 shows a cross-phenomena relationship between ‘persistence’ and ‘code examples’.

In particular the paradigm frames in the axial coding section of this chapter indicate the major events frequently occur when the student encounters a particular point in the syllabus or a particular task on a laboratory sheet. Subsequently, the way they deal with this concept or task in the form attempting to accommodate it may determine their state of mind at that point as well as their overall perception of the programming subject. Successful outcomes in terms of these events may serve to see the challenge posed by programming in a positive light whilst unsuccessful outcomes may have the opposite effect.
Chapter 6 Data Analysis

Criterion 4 – How Theoretical Sampling Proceeded

As indicated above, this criterion relates to how theoretical formulations guided some of the data collection. Whilst open sampling was adhered to, in instances where a category that was referred to in an open-ended interview had emerged in earlier data collection, subsequent data specific questions were asked to gain the maximum variation in the properties and dimensions of that category. For example, after the category ‘skip over’ emerged in initial data collection and properties and dimensions had been applied to it, references to it in subsequent interviews would result in follow-up probes that pertained to the emerging properties. For example, the question ‘how long would you skip over a topic for?’ would help in saturating the ‘duration’ property. Another follow-up question like ‘would you come back to that topic?’ would assist in determining whether the ‘duration’ property had a dimension value of ‘indefinite’ or ‘permanent’. It is important to reiterate at this point that this study progressed on the premise of ‘natural’ theory emergence where questions pertaining to categories and their properties/dimensions were only used if the participant made explicit reference to them. Here, if the participant did not mention skipping over material as part of open ended questioning, questions pertaining to this category (that may have emerged in previous interviews) were not asked. It was decided that this approach of ‘fishing’ for categories would fly in the face of inductive theory generation and natural theory saturation as described in chapter 4. In this regard, the author was conscious to produce theory that was naturally emerging in the data and not forced through a series of closed questions. If this forced questioning is taken to its logical conclusion then the last participant would have to be asked a series of closed questions that related to all categories that emerged beforehand and this reverts more to a quantitative questionnaire approach than an inductive qualitative approach.

Criterion 5 – What the Hypotheses Were Pertaining to Conceptual Relations

In terms of this criterion it is important to clearly state what Strauss and Corbin (1998) mean by the term ‘hypotheses’. Here, they indicate that categories or phenomena will emerge in the data as collection progresses. During the initial data gathering in interviews the researcher may begin to notice how concepts appear to be
related to each other, referred to by Strauss and Corbin (1998) as ‘relational statements’:

In explicating these relationships, the researcher begins to link categories with their subcategories, that is, to notice that these seem to be conditions – these actions/interactions, these consequences. We call these initial hunches about how concepts relate “hypotheses” because they link two or more concepts explaining the what, why, where and how of a phenomenon. (p.135)

Furthermore, Strauss and Corbin (1998) are quick to point out that although hypotheses are derived from the data, they are ‘abstractions’ i.e. statements made at the conceptual level rather than the raw data level. In this study it was found that relationships emerged from the data very quickly e.g. at one early interview the participant articulated ‘I skipped over a lot of big words’. This suggested a relationship between unintelligible material and avoidance rather than tackling it head on. Over a period of time, ‘skipping over’ became subcategory of the ‘avoidance strategy’ phenomenon and ‘words’ was encapsulated by the higher order category ‘alien’. As data collection progressed, it was found that there was evidence emerging that supported the hypothesis and other evidence that didn’t. In this situation Strauss and Corbin (1998) indicate that ‘when a contradiction was found, it is important to note whether the data represents a true inconsistency or whether they denote an extreme dimension or variation of the phenomenon’ (p.135). In terms of hypothesis just mentioned the latter was the case where the apparent contradiction was attributed to a different learner type i.e. between a ‘disengager’ and an ‘achiever’. This is one example of the numerous inter-category relationships that are discussed in detail in previous sections of this chapter.

Criterion 6 – Instances in Data not Explained by Hypotheses

The approach taken in open coding was to document every possible relationship that emerged in data collection. Instances where these relationships were isolated to one
interview were noted. In any research project, it is likely that isolated incidents could emerge that might provide the source for future research. One example of an isolated relationship was that of 'regret' at engaging in 'avoidance strategies'. Whilst the latter category was saturated the former wasn't. In this case, exploration of this in a future study might provide interesting results in terms of preparing students for their approach to the programming subject. Similar issues will be discussed in the conclusion chapter.

Criterion 7 – How the Core Category Was Selected

Section 6.5 of this chapter provides a detailed account of how and why the core category was selected. Suffice to say here that as the research progresses and the analyst iterates between data collection and analysis they begin to see the small fragments of the jigsaw blending together. Furthermore, in this study the central or core category ‘the challenging nature of programming’ was an in-vivo concept that ascended to the top of the abstraction hierarchy in a similar way to how an air bubble rises to the water’s surface. Given the fact that ‘challenging’ appeared as an in-vivo concept, detailed re-analysis of the data was conducted to ensure that it encapsulated all of the phenomena and their respective sub-categories.

As indicated earlier, Straus and Corbin (1998) also provide a list of evaluative criteria to assess the ‘empirical grounding of a study’ (p.270):

1. Are concepts generated?
2. Are concepts systematically related?
3. Are there many conceptual linkages and are the categories well developed?
4. Is variation built into the theory?
5. Are the conditions under which variation can be found built into the study and explained?
6. Has process been taken into account?
7. Do the theoretical findings seem significant, and to what extent?
8. Does the theory stand the test of time and become part of the discussions and ideas exchanged among relevant social and professional groups?
Criterion 1 – Are concepts generated?

According to Strauss and Corbin (1998), concepts 'are the building blocks of theory' (p.101). In this study, from the first interview onwards, a large number of concepts were generated. In the early data these were tentative concepts in the form of codes as illustrated in fig 6.1. The term ‘tentative’ is used here to represent the fact that the author would document all concepts that manifested in the data irrespective of whether they eventually became full-blown categories or not. This type concept development is the output of line by line microanalysis of the data. In many cases, some tentative concepts were subsumed into others to form categories. For example, fig 6.12 illustrates two tentative concepts ‘unintelligible stuff’ and ‘inadequate time’. As the data collection proceeded similar concepts emerged and resulted in the development of a category. In many cases the label of the category was based on an in-vivo term that appeared in subsequent data collection. For example ‘unintelligible stuff’ was subsumed into the higher order category ‘words’ and ‘inadequate time’ was encapsulated in the in-vivo term ‘sink in’ time. As these concepts started to emerge they were developed into categories in the form of properties and dimensions. These properties and dimensions were useful in developing follow-up questions to saturate the emerging categories e.g. ‘how long would it take a concept to sink in?’. A question like this would emanate from the time property of the sink-in code. In addition, emergence of the quantity property would prompt the question ‘would it all sink in after that time had elapsed?’. As mentioned earlier in the thesis, concept and category development was a laborious and time-consuming process. However time spent on this activity was invaluable in extracting a set of categories that accurately represented the data.
Unintelligible Stuff [US]

'I could never remember the stuff at the start, public void static' (P5)

'I take out bits I don’t understand and see what it does [TE]' (P5)

Trial and Error [TE] – In-Vivo

'I kind of go through it, take out bits and see what it does, like trial and error, just to see what it does' (P5)

Sink In [SI] – In-Vivo + [CLK]

'Well usually, I don’t understand it at the start, it kinds of clicks in at the end, I keep on doing it' (p5) (Overlap between persistence and SI) (P5)

Persistence [P]

'I keep on googling it until I find something, always looking for code' (P5)

Inadequate Time [IAT]

'The project was difficult because we weren’t very far into applets before we did it' (P3)

Memo - Feeling of being unprepared?

Sink In [SI]

'They were difficult to pick up for the first two weeks because of the difference between applets and applications'

'You just can’t take it all in, what an applet actually is, even the “hello world” applet is still longer, you have to import stuff, albeit 8 lines'. (P3)

Persistence [P]

'The more you work on them the more you find how similar they are, its only a few lines of code that are different' (P3)
Chapter 6 Data Analysis

Criterion 2 – Are concepts systematically related?

Engagement in the open coding process as highlighted in fig 6.12 as well as conglomereration illustrated in appendix 2 resulted in an awareness early on in the project that many concepts (and subsequently categories) were involved in relationships with other concepts. In terms of theory development, this manifested itself in many ways e.g.:

\[
\text{[SI] + [PACE]}
\]

\['When I start new concepts like methods, constructors, classes, inheritance etc, all of like in 2 weeks or something, so much to take in and try to put them together' (P19)\]

In the above data extract one witnesses a relationship between 'sink in' time and a tentative concept like 'pace' which was an abbreviation for pace of delivery. As data collection progressed the tentative concept 'pace' was subsumed into a property of the 'material delivery' category. Thus, in this small extract from the data one can begin to appreciate the iterative nature of theory development where the author had on many occasions to constantly revisit data and decide whether concepts would become full-blown categories or alternatively, properties of categories. Furthermore, using abstraction, relationships can be inferred from the data that indicates a relationship between the 'sink in' time and 'material delivery' categories (both described in detail in appendix 2). Specifically, one can infer from the data that it may be possible that the amount of time it takes for material to sink in and the quantity of the required material that actually sinks in are influenced by the perceived depth and pace of delivery. As fig 6.8 in this chapter indicates these two related categories were encapsulated by the phenomenon programming organization. This simple example has shown that whilst there may not appear to be explicit relationships between concepts in the data, relationships between concepts that are properties of different categories highlight a possible relationship between these categories that needs to be explored in subsequent data collection and analysis. This example is only one instance of many inter-concept relationships that emerged during data analysis. Furthermore, this constant iteration between theory building and data collection, looking for
properties, dimensions and relationships and their subsequent abstraction into phenomena was found to be a major task of grounded theory.

Criterion - 3 Are there many conceptual linkages? Are the categories well developed?

In this regard, Strauss and Corbin (1998) indicate that categories should be tightly linked which applies to both categories and their subcategories and the core category. Specifically, this criterion requires that categories should be theoretically dense with many properties that are dimensionalised. Furthermore, they indicate that 'it is the tight linkages and density of the categories (many properties and dimensional variations) that give a theory its specificity and explanatory power' (p.271). As was highlighted in the previous section, constant iteration between theory building and data collection, looking for properties, dimensions and relationships and their subsequent abstraction was a major feature of this study. This is also known as the approach of constant comparison where new instances of categories are compared with existing ones in order to uncover the widest possible variation in terms of properties and dimensions. The more this happens, the richer the category becomes in terms of variation and saturation. This criterion therefore has two requirements to be satisfied. Firstly, each phenomenon that results through abstraction should encapsulate a logical grouping of inter-related categories. Secondly, in so far as the data will allow, each category should have the maximum number of properties and dimensional variation. In terms of the first requirement, this study resulted in four abstracted phenomena being generated namely, 'state of mind', 'perception of programming', 'programming organisation' and 'dealing strategies' respectively. Each of these (and the subset of their sub-categories that apply in any given situation) are influenced by how a student experiences the 'challenging nature of programming' i.e. the core category. In further fulfilment of this requirement, each phenomenon encapsulates a set of subcategories in the form of logical groupings. This was an essential part of axial coding where categories are 'fractured' or fragmented and re-grouped logically around the axis of a category which becomes an abstracted phenomenon. It should be noted that these were all natural groupings with no forced or contrived relationships between a given phenomenon and its subcategories. For example, fig 6.9 is a prime example of a group of logically related 'states of mind'
that were encountered in the data. The linkages between the central category and the
phenomenon in fig 6.10 theory was found in this study to represent a macro-level set
of interacting phenomena that were a clear and accurate representation of the major
issues at play in the data. In terms of the second requirement, analysis of the
subcategories and their properties and dimensions was found to be a micro-level
endeavour where the analyst, in a not to dissimilar manner to the ‘data miner’ is
trawling the data for low-level properties and dimensions. A prime example of a
logically related dense set of categories and sub-categories is illustrated in the
programming organization phenomenon in fig 6.8. Here density is represented in two
ways. Firstly, macro-level strategies and experiences with regard to ‘programming
examples’ has to two lower levels of abstraction. These lowest level categories
‘clarifying example’ and ‘ineffective example’ represent the conceptual variation of
the ‘programming example’ category where students, whilst in a constant search for a
clarifying example to crystallize a programming concept may encounter a lot of
ineffective examples along the way that fail to explain a concept in a satisfactory
fashion. Secondly, density in the form of the maximum number of properties and
dimensional variability can be found in examples like the ‘sink in’ code which has
four properties namely ‘level’, ‘time’ ‘awareness’ and ‘quantity’ (see appendix 2 for
detailed explanation of these). Furthermore, this rich variation in properties and
dimensions facilitated the learner categorisation presented in the previous section of
this chapter.

Criterion 4 - Is variation built into the theory?

Strauss and Corbin (1998) indicate that ‘variation is important because it signifies that
a concept has been examined under a different series of different conditions and
across a range of dimensions’ (p.271). In this regard, they emphasise that some
studies only report a single phenomenon and establish only a few conditions under
which it appears. This results in limited actions and interactions and variations being
presented by the study. The grounded theory presented in this study in both the
conglomeration document of appendix 2 and the learner type classification described
in this chapter exhibits a wide range of variation with student learner types along their
properties and dimensions. For example, if we revisit the ‘difficulty’ category that
exists under the ‘perception of programming’ phenomena we see it contains three properties that vary considerably in terms of dimensions. Nature relates to whether difficulty has a positive or negative impact on the student (a student in the middle ground of this continuum in the data would describe it as ‘just OK’). The data indicates that a number of participants refer to programming as ‘enjoyable but tough’. This seems to be a common experience for students who enjoy programming even though it makes considerable demands on them. Extent refers to how much of the material on the programming syllabus the student finds difficult. In the data we see participants referring to ‘so many’ and ‘an awful lot’ and ‘hundreds of different ways’ etc. Duration relates to how long this sense of difficulty remains with the student. In this regard we see participant referring to things like ‘at the start’ ‘you are gone for the rest of it’, ‘since the beginning’ etc. Here the variation on the perception of difficulty as positive (where it presents a positive ‘challenge’, keeps students on their toes and avoids boredom) or negative (where it can result in feelings of exasperation and self doubt over a sustained period of time which may result in the student giving up) is presented. In terms of variation at a higher level of abstraction, we can see that the learner type continuum in fig 6.11 enables us to analyse any category e.g. in this case ‘difficulty’ in terms of a given point on the continuum. This can be done with any category developed during open and axial coding. It should also be noted that a given category within a phenomena may not exist at certain points on the continuum e.g. the feeling of ‘intimidation’ may not exist within the ‘achiever’ range of the scale and this is also applicable to other categories e.g. ‘disengagement’.

Criterion 5 - Are conditions under which variation can be found built into the study?

According to Strauss and Corbin (1998) any explanation of a phenomenon should include the conditions under which it can be found. This should include broad or ‘macro’ conditions as well as ‘micro’ conditions (i.e. those that have an immediate bearing on the phenomenon under consideration). In essence, the theory should present an interweaving of phenomena, causal conditions and interaction strategies at both a macro and micro level. They go on to state that these conditions may exist as external factors such as economic factors, organizational policies etc. As described in the last criterion, this variability has been described clearly in this study. For example,
if we examine the ‘prior knowledge’ category we see that variations of it are influenced by the policy of the Irish Department of Education not to provide computer programming as a subject on the second level curriculum. The conglomeration document in appendix 2 also illustrates ‘micro strategies’ that are engaged in by students such as developing ‘self examples’ and seeking out ‘clarifying examples’ that will ‘set a concept in stone’ and can be reverted back to in situations where that particular concept e.g. ‘parameter passing’ presents problems. In this regard, whilst higher level phenomena like ‘programming organisation’ may represent macro-level theory in terms of their association with the central category, lower level categories like ‘self examples’ or ‘peer assistance’ illustrate the theory at a micro-level. This issue will be revisited in chapter 7.

Criterion 6 – Has process been taken into account?

As well as analyzing data for properties and dimensions, Strauss and Corbin (1998) also require that the grounded theorist be aware of process. Here, the analyst presents how changes in sequences of evolving action/interaction and changes can be traced to structural conditions. In particular, they indicate that changes may be anticipated, planned for and predicted. Alternatively they can occur quite unexpectedly and require some on the spot problem solving. This notion of process thus is inextricably linked with the coding paradigm in axial coding where a phenomena is described in terms of its causal conditions, context, intervening conditions, action strategies and consequences. For example, a certain action strategy may result in changes in context and thus ‘framing’ the next action strategy. Furthermore, Strauss and Corbin (1998) assert the variable nature of process where individuals respond to situations in which they find themselves. In this study, the notion of process has been implicit in terms of the detailed analysis of phenomena in terms of context, causal conditions etc (e.g. see table 6.5 as an example). In addition, the presentation of the learning continuum in fig 6.11 and its subsequent discussion of the various learner types highlights in detail the dynamic and ever-changing nature of programming experience. For example, it shows how ‘accomplishment’ can result in overcoming ‘self doubt’ and how a student falling into the ‘worker’ learner type category can oscillate between positive and negative attitudes towards the challenging nature of programming. Furthermore, analysis of the
Chapter 6 Data Analysis

data shows many dynamic interactions between categories in the form of positive feedback loops as illustrated in fig 6.13.

![Positive Feedback Loop Diagram](image)

Fig 6.13 Positive Feedback Loop

Here, we see that an increase in persistence may increase the likelihood of uncovering a clarifying example. This likelihood will be increased if the extent the student goes is full (where they exhaust all possible sources like text books, code examples, internet etc) over a long duration (where extent and duration are properties of the ‘persistence’ category). Furthermore, the greater the chance of uncovering a clarifying example, then, the greater the likelihood of gaining accomplishment in relation to the task. The net result of this is for the student to increase their persistence because increasing it in the past has worked i.e. positive feedback loop. This can vary if a clarifying example is not found which may in turn increase a level of ‘self doubt’ which may in turn cause a net decrease in persistence because it didn’t pay off as a strategy and the student ‘disengages’ (i.e. a negative feedback loop). This is only a small example of the implicit nature of process in this study.

Criterion 7 – Do the theoretical findings seem significant?

To the author’s knowledge, apart from the recent study phenomenographic by Stamouli and Huggard (2006) this is the only qualitative study conducted on novice programming in the Irish Republic. In terms of the significance of the results of this study, this chapter has shown that many issues of importance have arisen. In addition, the theory developed in this study across four sites was considerably aided by the researcher’s significant level of theoretical sensitivity in the area of programming education as well as engagement in an immense amount of iterative analysis. In terms
of creativity as described by Strauss and Corbin (1998) the author attempted to strike the optimal balance between imaginative analysis and technical correctness. In this regard, the author asserts that the central category and its association with the major phenomena is an accurate portrayal of the various experiences of the participants in the study.

**Criterion 8 – Does the theory stand the test of time**

This is largely an unknown quantity at this stage. However, suffice to say that many of the findings in this study overlap with findings of other studies in the literature as indicated both in this chapter and the literature review. Only future studies of programming in Irish third-level institutions will determine how representative these findings are with the actual phenomena experienced by the Irish novice programming population.

### 6.7.2 Other Evaluative Criteria

Miles and Huberman (1994) also present the qualitative researcher with a list of ‘relevant questions’ under a number of headings that can assist in the evaluation of a qualitative study. These will be presented in subsequent paragraphs.

**Objectivity/Confirmability**

Are the study's general methods and procedures described explicitly in detail? Do we feel that we have a complete picture including ‘backstage information’?

In this study, rigorous adherence to proper inductive grounded theory was engaged upon and as such it was decided at an early stage that a detailed description of all steps in this research was an inherent attribute of this thesis. Furthermore, no backstage information was omitted from this thesis with all steps being made explicit, from the selection of participants to the selection of a core category
Can we follow the actual sequence of how data were collected, processed, condensed/transformed and displayed for conclusion drawing?

The chapter on data collection presents a clear description of the data collection that was conducted over a number of weeks across four separate research sites. In addition, where data reduction in the form of merging categories into one single category occurred, this is clearly mentioned in the thesis e.g. the merging of ‘trial and error’ and ‘guess’ or ‘self awareness’ and ‘self doubt’ into one category. Furthermore, some concepts that were initially thought to be categories ended up as being action strategies of categories e.g. ‘knuckle down’ or ‘modify existing code’.

Are the conclusions explicitly linked with exhibits of condensed/displayed data?

The conclusions and phenomena illustrated are a direct reflection of the data as detailed in the conglomeration document in appendix 2. Care was taken not to derive any non-data related concepts or conclusions.

Is there a record of the study’s methods and procedures, detailed enough to be followed as an audit trail?

This chapter in conjunction with chapters 4 and 5 on data collection and analysis process respectively provided a detailed step-by-step account of the entire data collection and analysis procedures conducted in this study.

Has the researcher been explicit and as self-aware as possible about personal assumptions, values and biases, affective states and how they have come into play during the study?

The author as clearly indicated his background as computer programming lecturer in and Irish third level institution for over a decade. In this regard, the researcher made extensive use of theoretical memos and reflection. Given his background, he used a
deliberate strategy of not jumping in too quickly with assumptions when participants raised certain issues. As highlighted in chapter 3 a deep understanding of the subject material may cause the researcher to disengage analytically as soon as they hear a certain term being used by the respondent and fail to gain an insight into how and why a student experiences a certain situation. Only in extreme circumstances where a student was experiencing serious discomfort and embarrassment in not articulating a concept e.g. ‘parameter passing’ did the author mention it. This was done to ensure a comfortable atmosphere for the remainder of the interview. In addition, a primary aim of this research was to engage in pure inductive theory generation which constantly entailed bracketing one’s knowledge in order to allow the data to emerge in the voices of the participants.

Reliability/Dependability/Auditability

Are the research questions clear and are the features of the study design congruent with them?

As indicated in this thesis the research question was open and straight-forward and pertained to the learning experiences of first year Irish programming students at third-level institutions. As such, any student engaged in a programming course would possess a set of valuable personal insights into the programming learning experience. For this reason, all those who volunteered to participate across the four sites were accepted as participants. Furthermore, in advance of engaging in the study the author engaged in a detailed study of the grounded theory method to assess its suitability (as part of paper 3 on the Ed.D. course work). In this regard, one issue in the Strauss and Corbin (1988) text crystallised the suitability of the method to the research question. In particular, applying grounded theory to this type of research context enables respondents to describe their experiences and problems in their own voices and this can enable the researcher to ‘discover the issues that are important or problematic in the respondents’ lives’ (Strauss and Corbin, 1998, p.38). This detailed study of grounded theory resulted in the author being completely satisfied as to its suitability to the research question.
Chapter 6

**Data Analysis**

*Is the researcher's role and status within the site explicitly described?*

As clearly specified earlier, despite being a programming lecturer, the author had no link with any of the participants and accessed the sites with the cooperation of on-site lecturers who acted as gate keepers. In addition, anonymity was assured and this resulted in honest and detailed accounts of participants’ experiences.

*Do findings show meaningful parallelism across data sources?*

The data show considerable overlap across data sources and sites. Despite the fact that different teaching methods were used at each site the learning issues that were uncovered were consistent across the four data sites. In fact, no new categories actually emerged in the last tranche of interviews. This is evident from the participant numbers in the conglomeration of appendix 2 where the participant number appears after the reference to the category in the data i.e. (P19). As an example, the category ‘skip-over’ appears in the interviews 3 and 24.

*Were data collected across the full range of settings, times and respondents and so on as suggested by the research question?*

Data was collected across all four research sites. On a slightly negative note, focus groups were only held at two sites as no participants volunteered for a focus group at the other two sites. In addition, only a small number of participants submitted a diary/journal. As mentioned earlier, data was collected within a number of weeks close to the end of the Spring term/semester where participants would have had close to equal exposure to most programming topics in first year. This timing was deliberately chosen to maximise the variation in experiences across the full scope of a first year programming syllabus.
Internal Validity/Credibility

How context-rich and meaningful are the descriptions?

The research sites were specifically chosen because they contained first year programming students attending a computer science course as opposed to an engineering course with programming as a subject in year one. In this regard, the data, mainly in the form of interview transcripts show a rich variation in experiences from what could be described as a homogeneous population across four research sites. This context richness with abundant variation is evident in section 6.6 of this chapter in terms of learner type categorisation.

Does the account ring true, make sense, seem convincing or plausible, enable a 'vicarious presence' for the reader?

Despite being a lecturer in computer programming, interviewing participants and subsequent reading of the transcripts enabled the author to see the learning experiences of students through their eyes. In addition, many of these experiences are congruent with research conducted in other studies outside Ireland.

Did triangulation among complementary methods and data sources produce generally converging conclusions?

As mentioned earlier, there was a slightly disappointing uptake on focus group and diary/journal participation. However, the focus groups did allow further elaboration on categories that emerged in the interview data. In particular, the focus group allowed the benefit of a number of participants talking about the same issue in quick succession. This proved very valuable in both gaining category variation and saturation. In terms of divergent conclusions, the focus group data did not generate any new categories that hadn't been presented in the interview data. Because of the task-reporting nature of the diary, some of the data pertained to micro issues on various programming tasks and this proved valuable at the sub-category level.
Chapter 6

Data Analysis

Are areas of uncertainty identified?

Areas of uncertainty were not uncovered per se. However a number of issues which didn’t reach saturation were noted during open coding such as ‘fear of future’ and ‘regret’. These will be discussed in the conclusions chapter.

External Validity/Transferability

Are the characteristics of the original sample of persons fully described enough?

A detailed description of the participants and the voluntary nature of participation in terms of multiple data collection methods is presented in chapter 4.

Is the sampling theoretically diverse enough to encourage broader applicability?

The theory developed in this study is described as a substantive theory where it doesn’t attempt to explain outside its immediate field of study (Goulding, 2002). This can be contrasted to a formal theory, which, according to Goulding (2002) ‘has explanatory power across a range of situations’ (p.46). However, given the fact that data was collected from four separate sites it could be argued that the theory generated can be applicable in some way to the novice programming population in Irish third-level institutions. However, given the substantive nature of this study, no claims outside the scope of the data are made.

Are the findings congruent with, connected to, or confirmatory of prior theory?

The literature review in chapter 2 highlighted that there exists considerable variation between ‘stoppers’ and ‘movers’ (Perkins et al. 1989). Perkins and Martin (1986) refer to the ‘fragile’ nature of computer programming. Jenkins and Davy (2000) refer to the ‘difficult’ nature of novice computer programming with very few students achieving the status of ‘rocket scientist’. Analysis of the findings of this chapter show
considerable overlap at both a macro level (in terms of the various phenomena) and a micro-level (in terms of low-level concepts like 'disengaging', 'self doubt', 'guessing' etc). This overlap and synthesis of prior theory with theory emerging in this study is interspersed in the axial coding section of this chapter (section 6.4).

Are the processes and outcomes described in the conclusions generic enough to be applicable in other settings?

It may be the case that many issues like encountering a subject for the very first time as found in this study will exhibit overlap with similar problem solving type subjects at third-level on a first time basis e.g. physics, chemistry. Anecdotal evidence suggests that many Irish students only see physics and chemistry for the first time as part of a third-level course in general science or engineering. In addition, some Irish third-level institutes incorporate the physics subject into the first year computing course schedule. It will be up to readers of this thesis to determine whether or not the issues discussed exhibit congruence with their respective disciplines.

6.8 CONCLUSION

This chapter has presented the major findings of this study. In particular, it has shown how four major phenomena have emerged from the data namely, dealing strategies, state of mind, perception of programming and programming organisation. These phenomena have emerged out of abstraction and of logical groupings of emergent categories. In terms of selective coding it shows how the aforementioned phenomena revolved around the central category in a similar way to how sub-categories revolve around phenomena at a lower level of abstraction during axial coding. It has illustrated in detail how the central category 'the challenging nature of programming' encapsulates the major phenomena. Furthermore, it has explicated this theory using a 'storyline memo' as prescribed by Strauss and Corbin (1998) and a categorisation/taxonomy of programming learner types. This taxonomy or categorisation illustrates that each learner type on a continuum represents a complex
and dynamic set of interacting categories that cross phenomenon boundaries. Furthermore, it has shown that the dynamic nature of the challenging nature of programming can result in a given student experiencing many different points on the learning continuum as they progress through a programming syllabus. Furthermore, the student may oscillate between negative and positive states in terms of the challenging nature of programming. Finally, this chapter has discussed in detail how this study has conformed rigorously to the requirements of the Strauss and Corbin (1998) methodology in terms of validity and reliability.
Chapter 7

Conclusions

• How The Research Question Was Answered
• Original Findings in the Study
• Implications for Teaching Practice
• Methodological Clarity
• Methodological Originality
• Limitations of the Study
• Future Policy and Practice
• Conclusions and Future Research

7.1 HOW THE RESEARCH QUESTION WAS ANSWERED

The introduction chapter presented the user with the two main interconnected aims of this study. The first aim was to develop a substantive grounded theory based on the learning experiences of first year programming students across four Irish third-level research sites. The second aim was to use this theory to develop a categorisation/classification of novice programmer learner types. Chapter 6 illustrates in detail how each of these aims was achieved.

In terms of the first aim, a grounded theory with the central category pertaining to the challenging nature of programming emerged from data collection and analysis. Fig 6.10 clearly indicates that this challenge can be viewed by a student as either a
positive or negative experience. In addition, an interesting issue that emerged was that in some cases this experience can oscillate between feelings of positivity and negativity in terms of the challenge. For example, enjoyment and achievement at an early part of the syllabus may be counterbalanced by difficulty and confusion with regard to a later topic on the course.

In terms of the second aim, a learner type categorisation in the form of a learning continuum as depicted in fig 6.11 was developed. The interconnectivity of these aims pertains to the fact that a detailed analysis of each learner type identified on the learning continuum was developed in terms of the four major phenomena that comprised the theory namely dealing strategies, programming organisation, state of mind and perception of programming. Variation amongst category types was achieved by analysing the varying dimensional values of the sub-categories of the phenomena. In particular, isolation of any point on the continuum is represented by a specific set of inter-related categories (with a particular set of dimensional values for the properties of each category) across all four phenomena as depicted in fig 2.1. Furthermore, detailed analysis resulted in the finding that a student may experience a number of learner type categorisations as well as oscillating between them as they progress through the programming subject over the academic year. In essence, this study has found that a student’s positioning on the continuum may be in a regular state of flux depending on their exposure to new programming concepts, teaching techniques, laboratory assignments etc. In this regard, we may find students constantly engaging in macro-level activities such as progression strategies or programming organisation and micro-level strategies such as peer assistance or seeking out clarifying examples in an attempt to reposition themselves on the positive end of the challenging continuum. These issues present implications for teaching practice and, as such, are discussed later in this chapter.

7.2 ORIGINAL FINDINGS IN THIS STUDY

In the course of this research it has been found that the notion or originality has two dimensions in a grounded theory study. Firstly, given the inductive nature of
Chapter 7 Conclusions

grounded theory study conducted with a specific group of participants in a particular context, originality is an inherent property of the resultant substantive theory. The second dimension pertains to ascertaining whether the emergent theory makes original contributions to the research domain, in this case, novice computer programming. In terms of the first dimension, a grounded theory of this nature has not been conducted with Irish third-level students. In this regard, the substantive theory that emerged incorporating four major phenomena encapsulated by a central category is intrinsically original. The second dimension must consider whether the categories that emerged in this study and their interrelationships provide original contributions to the domain of novice programming at large (i.e. not just within the Irish third-level context). This second dimension will now be discussed in terms of the two main aims of the research, namely theory relating to the learning of programming and the development of a programming learner type categorisation based on the theory.

Reflection on the theory generated in this study and the subsequent classification of programming learner types has resulted in two interrelated issues that have potential implications for both understanding how learning programming is actually experienced as well as teaching practice. Firstly, closer examination of theory presented in fig 6.10 indicates the existence of two main factors in dealing with the challenging nature of programming. Analysis of the top half of the diagram indicates that the phenomena dealing strategies and programming organisation pertain to how the student approaches the programming subject and thus could be referred to as approach strategies. Likewise, analysis of the lower half of the diagram indicates that the phenomena state of mind and perception of programming relate to the emotional state of the student and their attitudes and this could be referred to as emotions/attitudes. The logical progression of this abstraction suggests that the programming educator may need to deliver the programming subject to a group of students with varying abilities (as indicated in fig 6.11) in a way that optimises the strategies they use in approaching the subject and, in as far as possible, instils positive emotions and attitudes about the programming subject where the challenge posed by it is something to be embraced and not feared. In addition, these two abstractions may form useful reference points for researchers conducting programming learning research whereby each one could be researched individually or collectively in future studies in order to ascertain what effect (if any) each one has on the other. For
example, future studies could investigate the role of scaffolding as an approach strategy to instil positive emotions/attitudes about the programming subject.

The second issue uncovered through reflection relates to the emergence of both 'macro-level' and 'micro-level' theory uncovered in the study. Macro-level theory relates to general issues as portrayed by phenomena and higher level categories. In a similar way to that just described in the previous paragraph, a detailed analysis of the phenomena presented in chapter illustrates the tacit separation of approach strategies and emotions/attitudes. In particular, the emotion/attitudes phenomenon encapsulates higher order categories only (i.e. they do not encapsulate lower level categories, apart from the alien category which subsumes alien concepts and words as illustrated in fig 6.7). In contrast, analysis of the approach strategies phenomenon uncovers a plethora of sub-categories that this research will refer to as micro-level theory, that when investigated, uncovers the low-level activities engaged upon by students whilst working through their programming subject. The implications of macro and micro theory for teaching practice will be discussed in this chapter.

In terms of the second aim of this research, original findings also emerged in the development of the categorisation of learner types in the form of a learning continuum. The first issue that differentiates the learner continuum from those developed in the past e.g. the 'stopper-mover' continuum as developed by Perkins et al. (1989) is that it is based on the central category of the theory and its underlying phenomena as illustrated in fig 6.11. In this regard, each isolated position on the continuum in fig 6.11 e.g. disengager is presented in terms of a subset of interacting categories that emerged in the theory. For example, the disengager is likely to engage in avoidance strategies, may view the majority of code examples supplied to him/her as ineffective examples, may encounter self doubt and confused states of mind and their perception of programming may view it as excessively difficult and intimidating. In addition, depending the location of a point on the continuum, one will witness variation in terms of the dimensions of the properties of a category. For example, the slogger and worker learner types may both encounter confusion, but the worker, given its position further right on the continuum is likely to experience lower extents of confusion over a shorter period of time than the slogger. Here extent and duration are properties of the confused/confusion category which have the dimensions of partial-
full and short-long respectively (as highlighted in detail in the conglomerate document of appendix 2). This represents an original approach to categorisation whereby categories of learner can be richly described in the form of a network of interlinked categories with varying dimensional values that emanate from a substantive theory. If this continuum could be visually simulated one would see subtle changes in properties of categories and their dimensional levels as one moved from left to right with some categories being activated and subsequently deactivated depending on the respective location. For example, as one moved from left to right in a simulation of the continuum, the values of the properties of the confused category would successively reduce until one reached a point where it was no longer evident and became deactivated perhaps somewhere between the manager and achiever categories. Fig 2.1 gives a simple example of this where the activated categories on a certain point on the continuum are shaded. Furthermore, the labels attached to the learner categories using in-vivo terms from the data where possible are also original contributions of this study.

As well as originality in terms of the two main research aims, a number of original programming learning experiences emerged in the theory. Figures 6.10 and 6.11 depict a major finding in this research whereby a student’s experience of the challenging nature of programming can oscillate between positive and negative states. In addition, this oscillation can also take the form of alternating between positive and negative learner categorisations on the learning continuum. This oscillation area on the continuum was developed based on what was described in this study as ‘the double edge sword’ perspective on the challenging nature of programming. In this regard, some students may temporarily oscillate between negative and positive states of all of the phenomena depending on how they experience the material at any given time. In this regard, participants recounted how they ‘keep on top of things’, ‘come back into’ the work ethic and experience a resurgence of interest after periods of difficulty and disillusionment. This notion of the double-edge sword emanated from the description of programming in the data as ‘enjoyable but tough’. This indicates that whilst students may enjoy programming and work at it a way that progresses them along the continuum, they may encounter a difficult or tough topic in the syllabus that results in them drifting back towards the left side of the continuum as illustrated in fig 6.11. This drifting backwards may be a result of the perceived
toughness of material which results in some lessening of work ethic as small levels of disillusionment and futility set in. As such, the student in the oscillating zone may regularly face challenge after challenge and in this regard they may attempt to engage in strategies that position them on the right half of the continuum. For example, to overcome a confusing task they may combine micro-level strategies of implementing known bits of code to start it off and follow this up with peer assistance to aid understanding of the remaining parts. This issue of oscillation was developed in chapter 6 where the oscillating ‘worker’ may reach a point in the syllabus where they lose conceptual control of the material and how they deal with this after that point may determine whether they remain in either the positive or negative zone. In addition, the typical learner may experience a varying set of zones on the continuum. For example, they may find themselves ‘managing’ the earlier material and having to ‘slog’ through the more abstract material such as ‘polymorphism’ in object-orientation.

Furthermore, the use of grounded theory has enabled the uncovering of the specific contextual conditions, action strategies and outcomes that Irish students experience in terms of the challenging nature of programming. In this regard, it has discovered a number of fascinating insights. For example, it has shown how the concept of ‘accomplishment’ transcends the learner category divide with each learner type experiencing it at different levels. This may vary from getting a simple arithmetic expression completed to successful implementation of a complex object-oriented program. It has shown how the challenge posed by programming can both spark and retain interest across a number of the learner categories. For example, whilst the ‘worker’ may see this as an incentive to remain on the positive side of the challenging continuum, the ‘achiever’ may see it as a motivating factor in attaining perfection and accommodation of all concepts on a programming syllabus. In addition, it has uncovered other phenomena that have not been reported in literature. For example, it has shown how ‘bewilderment’ can be both an inhibitor and an enabler of progress in terms of the programming challenge. In particular, bewilderment, mixed with other negative emotions such as ‘confusion’ and ‘intimidation’ may serve to disillusion the student and subsequently disengage from the subject. On the other hand, ‘collective bewilderment’ as reported in the data may serve to motivate a group of weaker
students to persist with the attitude that they are not the only ones experiencing difficulty with programming.

Analysis of lower level categories in chapter 6 also uncovers original findings. For example, the study found two distinct types of avoidance strategies namely, *skipping over* and *disengagement*. The reason for two separate categories can be found in detailed analysis of the data, where, whilst both are conscious strategies to avoid current programming material, chapter 6 discusses a subtle difference between the two. In particular, detailed analysis of the data indicates that *disengagement* has to do more with lack of application of the student in the lecture or laboratory where they may become 'lazy' by losing initial enthusiasm or allowing themselves to 'drift', become 'disinterested' or 'switch off' from the particular topic or task. On the other hand, when *skipping over* material or concepts, rather than allowing themselves to 'drift', participants are mentally engaged in a state of working through the programming material and deliberately skip over the parts that confound them. Rather than 'drifting' and disengaging completely, they may often work on something else in the mean time that they do understand and revisit the difficult topic when either help becomes available or solutions are posted on the network. Obviously, if the skipped material is not returned to it may be considered to be a form of disengagement. This subtle dissection of the data using microanalysis and constant comparison was a feature of this study. In the same, way *accommodation issues* were subdivided into three separate sub-categories as illustrated in fig 6.8. Presenting this subtle variation explicitly in the form of category association diagrams and paradigm frames is also unique to this study. Using these approaches provides the researcher with a new approach to asking more focussed theoretical questions in terms of addressing the exact nature of relationships between categories as well as the potential sub-categorisation of a given category.

This study also made a deliberate effort to present the original macro or low-level strategies in an explicit fashion. In this way, it attempted to develop as much granularity as possible within the categories by teasing out properties and dimensions that might have been subsumed into higher level categories. For example, it has used the *nature* property to both reflect the positive and negative nature of the programming challenge as well as the different dimensions of a given property. In
Chapter 7

Conclusions

terms of the former, it has show how the nature of difficulty may be seen as a positive challenge to be overcome or a negative one that is best avoided. In terms of the latter, it has shown how the nature of 'assistance' may be 'one-way' or 'reciprocal' and 'guessing' may be 'random' or 'planned'. This approach enabled greater specificity with respect to the properties of categories. Whilst there may be a degree of conceptual overlap between the many low-level categories and results published in previous studies, this theory presents an original set of granular abstractions that have not been presented in other work. As discussed in chapter 2, this originality was achieved by conducting data collection and analysis in advance of a literature review.

Finally, chapter 4 highlighted how the teaching method varied across the four data collection sites, with two using the traditional lecture and laboratory-based approach and the other two using a problem-based learning (PBL) approach. In terms of percentages this equated to approximately a 50%-50% split. Data collection and analysis indicated that the concepts and categories that emerged transcended the teaching approach divide with no individual categories materialising that could be directly attributed to a specific teaching method. This is consistent with the findings in the literature where common problems and issues in programming have persisted over time (as well as languages and environments). This is most likely accounted for by the fact that the fundamental tenets of computer programming i.e. sequence, selection and iteration as well as the significant demands imposed by exactness of syntax and problem solving have outlived the passage of time.

7.3 IMPLICATIONS FOR TEACHING PRACTICE

A noticeable facet of a number of the studies presented in the literature review in chapter 2 related to how the researcher(s) assessed the implications of the respective studies for teaching practice. Chapter 1 outlined how one of the motivating factors for engaging in this study was to gain deeper understanding of the actual issues experienced by students, and thus, pave the way for more student-focussed teaching practices and material provision. In this manner, the implications of this research for teaching practice will now be discussed.
From a macro perspective, this study suggests that acute lecturer awareness of the emotions/attitudes of students pertaining to the programming subject is very important. Whilst a surface level many of these emotions/attitudes have negative connotations e.g. confusion, bewilderment, self doubt etc., the lecturer may be able to prepare students for these emotions in advance by assuring them that they are natural feelings to experience when dealing with new concepts. Reflection on chapter 6 has illustrated how the central category relating to the challenging nature of programming can be viewed as negative or positive. In this regard, rather than avoiding these emotions, programming educators should deal with them by instilling the notion that would-be negative perceptions like difficulty and confusion are challenges to be embraced. In addition, educators could possibly engage in the delivery of specifically designed lessons whereby some task that appears at a surface level to be difficult and intimidating can be broken down into a set of manageable steps on an incremental basis. In addition, if each step could be illustrated in a number of possible ways that appeals to different learning needs of students, this might lead to an even greater success. In this regard, the educator is engaging in teaching activities that address the issues generated by the theory as opposed to speculating what these issues of concern actually are. In this way, the theory may assist the educator in proactively shaping teaching practice.

Furthermore, where the theory suggests the existence of negative emotions, programming educators can analyse each one with the aim of neutralising, or minimising it to the fullest extent possible. For example, high levels of self doubt may be reduced by one-to-one tutorial sessions using scaffolding (as described in chapter 6). In addition, given the fact that the theory suggests that many of these negative emotions appear early in a programming course due to the ‘alien’ nature of the subject, the educator may attempt to pre-empt this by building assisted learning mechanisms into the teaching process. Woolfolk (2001) defines assisted learning as ‘providing strategic help in the initial stages of learning, gradually diminishing as students gain independence’ (p.49). For example, chapter 6 discussed how the awareness of others state of mind could be negative where the student perceives the progression of others as far superior to that of theirs. It may be possible that assisted learning techniques could be devised to transform this situation into a positive one where the strength of stronger performing students could be harnessed in the form of
scaffolding. Here, 'pair programming' (Bryant et al., 2005), a technique used in industry could be used where students are paired up so that attempts are made to get the weaker students into the zone of proximal development which is described by Woolfolk (2001) as the area where the student 'cannot solve a problem alone but can be successful under guidance or in collaboration with a more advanced peer' (p.50). In order to minimise the negative impact of personality clashes, the pairs could be changed after each task is complete. These examples are only a subset of the potential ways of addressing negative emotions as emerged in the study and may or may not be successful. What is more important is that the educator is aware of these problematic areas and attempts to devise strategies to address them. This is consistent with the recommendations of Perkins et al (1989) who indicated that ‘with appropriate instruction and encouragement students can learn not to give up so easily’ (p.267) and that ‘the result may be enhanced learning and enjoyment for many students’ (p.267). It seems plausible to suggest that any attempt to neutralise negative emotions and attitudes will be a step in the right direction.

From a micro-perspective, investigation of the approach strategies uncovers a host of low-level strategies employed by students to deal with the challenging demands of programming as well as accommodation-related issues. One of the major unexpected categories that emerged in this study pertained to accommodation issues. In this regard, a number of participants described how certain programming concepts took a longer time to sink in than perhaps anticipated by the lecturer. In the worse case scenario, participants recount being assessed on material that hasn’t been given adequate time to sink in (i.e. a student term for ‘accommodation’ as defined in table 6.1). In this regard, a student might accommodate a concept e.g. ‘for’ loops some time after the assessment on that topic has taken place, which may in turn lead to obvious frustration. A potential response strategy to this situation might be to develop automated environments with data banks of questions that are constantly updated whereby a student can take a test on any topic during a supervised laboratory session. In this case, the student could re-take an assessment e.g. a series of multiple choice questions on ‘for’ loop theory on an occasion when they feel they have accommodated the material (i.e. when they feel it has sunk in) in a similar way to re-taking the driving test.
Another type of accommodation takes the form of the *get head around* category. Here, rather than not understanding specific constructs in programming, the student fails to come to terms with the whole concept of programming e.g. its abstractness, syntax and exact demands of compilers. In order to deal with this, programming may have to be introduced at second-level in an introductory fashion at least in order that students are not put off by abstractness and exactness, but rather introduced to it on a phased basis. Another argument for introducing programming into the second-level system is based on the emergence of the *prior knowledge* category. The theory suggests that prior exposure to programming can facilitate the development of mental schemes that can be built upon at third-level. On the other hand, exposure to programming at second level may convince students who are not suited to the subject to avoid choosing it as a third-level career and thus doing them a valuable service. For example, students who do not take to the physics or chemistry subjects at second-level are unlikely to choose them as third-level career options. The argument for this provision of introductory programming also emerged in the *alien* category under the *perception of programming* phenomena. Here, participants recount programming as something they haven't seen before and consisted of a whole new set of *words* that were basically unintelligible. Here, exposure to programming and to such basic words as ‘variables’, ‘data types’ ‘sequential processing’, ‘iterative processing’ etc. might go a significant way in demystifying these terms. One interesting issue that emerged in a focus group discussion was where a group of participants suggested that the development of a simplified dictionary/glossary that explained programming concepts in a straightforward fashion would be useful. In addition, provision of this type of material might have the effect of minimising the amount of times students have to resort to negative dealing strategies such as *skipping over* material or *disengaging* from programming tasks. It should also be noted that these types of *avoidance strategies* were another unexpected category that emanated from the data. Subsequent analysis of the literature illustrated how they overlapped with the ‘stopper’ mentality as described by Perkins et al. (1989). In this regard, Perkins et al. (1989) indicate that educators must try and see out teaching approaches that instil the attributes of ‘movers’ and negate the attributes of ‘stoppers’.

Another important micro-level issue that emerged from the theory related to the types of *programming examples* students encounter in both lecture and textbook material. In this regard, the theory suggests that students clearly distinguish between *clarifying*
examples and ineffective examples. Here, the former enables the student to accommodate a concept, whilst the latter does not. The data suggests that ineffective examples are either overly complex or use metaphors or analogies that the student is not interested in such as ‘chess’ or ‘sodoku’. The worse case scenario of this situation was recounted by a participant who thought that two-dimensional arrays could only be associated with ‘chess’ and not football league tables. This issue presents an important challenge for educators in that it requires them to desist from making assumptions about what analogies or metaphors actually clarify a concept. In this regard, it seems that a data bank or library of examples that can be added to on an incremental basis might reduce the ineffective nature of presumptuous analogies in lecture notes.

7.4 LEARNING THEORY AND PEDAGOGY

Having discussed the theory developed in the context of its potential implications on teaching practice, it is also important to analyse how it relates to existing learning theory and subsequently, the issues it raises for computer programming pedagogy.

One of the most noticeable facets of the theory presented in this study through the learner type categorisation pertains to the varying learner types that engage in a computer programming course. In particular, the programming organisation phenomenon suggests that programming students engage in strategies that try to utilise what they know already in the form of existing schemes and prior programming knowledge as well as seeking out programming examples that aid learning in the form of clarifying examples and self examples. A detailed analysis of this phenomenon and the categories it encapsulates uncovers considerable resonance with constructivist learning theory where each individual student creates idiosyncratic versions of knowledge (Ben-Ari, 1988) based on their current or past knowledge. In addition, constructivist learning theory requires that learning must be active whereby ‘the student must construct knowledge assisted by guidance from the teacher and feedback from other students’ (Ben-Ari, 1988, p.258). This fact illustrates further overlap between constructivism and the theory developed in this thesis whereby
students attempting to progress engage in peer assistance and lecturer assistance on a persistent basis until they accommodate various programming concepts.

Chapter 2 has presented the reader with the issues relating to the use of metaphors and analogies in computer programming education and the reservations of some commentators (Du Boulay, 1989, Halsaz and Moran, 1982) with respect to these. Ben-Ari (1998) discusses the utilisation of this approach in the context of constructivism and presents an example where the notion of a 'box metaphor' to represent variables may be considered inadequate because students believe that it can simultaneously contain two values (when in fact they can only ever contain one at a time, a fundamental notion in computer programming). In this regard, he contends that whilst the students have constructed a consistent concept (i.e. the notion of a box for storage) their interpretation of it renders it non-viable for the particular programming construct. This has considerable conceptual overlap with the ineffective examples category uncovered in this study and described in chapter 6. In relation to this category, Ben-Ari (1998) makes particular reference to it in the context of constructivism in computer programming education where he indicates that 'the lack of an effective, if flawed, model of a computer is a serious obstacle to learning' (p. 259). This is an important issue witnessed in lectures and laboratories where we can have the situation that whilst students may have a slightly inaccurate model of a programming concept it still enables them to effectively write code to implement it. This model constructed in an idiosyncratic fashion may relate to the existing schemes that they already possess. From a constructivist pedagogic perspective it seems that facilitating students to utilise and subsequently refine their personal models (so that they can be aligned with the construct in a more accurate fashion) of programming concepts may be a more effective approach than assuming 'all students will construct the model that the instructor has' (Ben-Ari, 1998, p.26). This may lend weight to the viewpoint of some researchers that traditionally, computer programming is taught in an ego-centric fashion. Felder (1993) lends weight to the existence of this traditional egotistical approach when he points out that very often teachers tend to favour their own learning style, in part because they instinctively teach the way they were taught and that if this personal learning style is imposed on the student, it may prove to be counter-productive. Godleski (1984) indicates that when teachers rely on their own cognitive style preferences, many students are less engaged and perform poorly.
Chapter 7

Conclusions

Culver (1996) also identifies the limitations of this egocentric type approach in computer science education and warns against generating ‘our own preconceived notions of how students should learn’ (p.3). He goes on to state that in many cases ‘we teach the way we learn....limiting the educational opportunity of those who do not learn well the way we learn’ (p.3).

In fact, many programming educators and researcher are starting to question this traditional approach. McCracken et al. (2001) have observed this need for change emanating from the teaching profession when they indicate that ‘many computing educators have voiced concern of whether students are learning the necessary skills’ and furthermore, that the level of skill achieved by students was not ‘commensurate with their instructors expectations’ (p.125). Furthermore the ACM curriculum guidelines (ACM 2001) highlights that a common teaching approach of focussing on purely ‘mechanistic constructs’ of computer programming, does not serve the student well in terms of their overall learning needs. If one is to approach this from a constructivist perspective, Ben-Ari (1998) contends that if the student does not bring a preconceived model of the computer or computer programming to the class (in this study these may take the form of existing schemes) then educators must ensure that a viable hierarchy of models is constructed and refined as learning progresses. In particular, he indicates that these models ‘must be explicitly taught and not left to haphazard construction and not glossed over with facile analogies’ (p.260). Finally, in terms of the sink-in category uncovered in this study Ben-Ari (1998) indicates that ‘constructivism suggests that programming exercises should be delayed until a class discussion has enabled the construction of a good model of the computer’ (p.260).

This proposal is consistent with the findings of this study where participants recounted not having enough time for accommodation of programming concepts and often found themselves sitting an assessment even though the topic (e.g. arrays) hadn’t sank in. From a pedagogical perspective this suggests that development of a wide variety of explanatory models of both the computer and computer programming are required for a diverse set of learners in order that this sink-in time can be reduced.

Analysis of education literature (Woolfolk 2001, Schunk 2000, Kolb 1984) uncovers the notion of learning style as an important dimension of pedagogy design. In this regard, Woolfolk (2001) differentiates between ‘deep’ and ‘surface’ learning styles. In
the former, students have a deep-processing approach to learning and view learning materials as a means for understanding concepts. In the latter, the learner focuses on memorising learning materials, for the purpose of achieving grades. These varying learning styles have been uncovered in this study where as reported in chapter 6, the achiever learner type category will often retrospectively analyse a piece of programming code that they guessed with the aim of fully understanding why it actually worked. In this case their need to fully understand all programming concepts they encounter drives this type of strategy. In contrast, this study also uncovered phenomena that exhibit attributes consistent with surface learning. For example, skipping over certain material because it didn’t overly impinge on the overall programming grade was a disengagement strategy employed to certain extents by some learner types positioned somewhere between the disengager and worker learner type categories. In addition, as discussed in detail in chapter 6, these learner types may engage in one-way peer assistance whereby they are happy to acquire assistance in the form of their peer going through the entire solution with them and thus relieving them from the task of having to mentally engage in any level of depth with the task at hand. In contrast the deep learner would differ whereby the extent of the assistance would be much less and the nature of the assistance would be reciprocal (see appendix 2 for description of the nature and extent properties of the peer assistance property).

In further exploration of learning styles Kolb (1984) differentiates between ‘abstract’ and ‘concrete’ learners, where the former like to think their way through new concept, whereas the latter like to feel their way through with tangible examples they can relate to. These learning styles were also uncovered in the course of this study where higher achieving students recounted a level of comfort with abstract nature of some programs such as polymorphism and abstract classes. In contrast, concrete learners were found in this study to frequently seek clarifying examples that were aligned with their existing schemes, for example, the notion of a football league table to conceptualise a two-dimensional array as indicated in the previous section on teaching practice. In some cases, if these were not encountered some learners tended to disengage from material that appeared alien in nature given the ineffective nature of the examples used in the classroom.
In a recent research project on learning styles in undergraduate computer science conducted by Layman et al. (2005), the researchers looked for the existence of personality types using the Myers-Briggs personality test. In terms of personality test they found that many computer science instructors teach in a way that suits the intuitor personality type (those who are imaginative and concept-oriented). They contend that this approach is unsuitable for the sensor personality type (those who prefer details, procedures and practice as a means of absorbing information). These findings have overlap with those of this study where achievers displayed a level of comfortableness with abstract concepts to the extent that they accepted them as necessary nuances of computer programming. On the other hand, sloggers exhibit some level of overlap with sensors where they indicate a level of discomfort with abstract notions and prefer it where examples given have meaning in terms of real world concepts. In addition they welcome the use of methods and techniques that eliminate the abstractness of programming e.g. as indicated in chapter 6 a dictionary that described programming concepts in lay man’s terms in a sufficient level of detail. In this regard the researchers conclude that a balanced approach to programming pedagogy is required that appeals to the needs of the differing personality types in a given cohort. This contention is supported by Thomas et al. (2002) who claim that active, sensing and visual learners may be particularly disadvantaged by current methods of teaching programming.

In the context of learning theory the notion of self-regulated learning (SRL) (presented in chapter 6) is also being addressed in computer programming education research. In a recent Irish study, Bergin et al. (2005) found that students with high levels of task value and intrinsic goal orientation use more self-regulated strategies than their less successful counterparts. As described in chapter 6 this finding is consistent with the achiever learner type category uncovered in this study where they exhibited high levels of self-efficacy. In particular they engaged in activities such as setting themselves extra work that increased task value (and in some cases reduced potential boredom) and, whilst they did show awareness of the lesser progression of their counterparts who were struggling it did not result in them deviating from their goal of high levels of achievement in the programming subject. In response to this situation, Bergin et al. (2005) indicate that from a pedagogic perspective, specifically
designed tools that assist programming students in self-regulating their learning would be of potential benefit.

Finally, this section has shown how the findings from this thesis study exhibits considerable overlap with other studies conducted in the area of learning theory and computer programming. A strong message that seems to resonate from this research is that pedagogical advancements are required to transform computer programming education in a way that deals with student cohorts with diverse sets of learning styles and needs. Given the many advancements in computerised tools in recent years there still seems to be a need for pedagogical techniques that facilitate 'exemplary' programming teaching as called for by Linn and Dalbey (1989). To this end, it appears that tools such as MUPPETS (Phelps et al., 2003) that facilitates visual learning, student engagement and peer knowledge transmission have the potential to overcome some of the pedagogical deficiencies experienced by many programming students. However, as indicated in chapter 2, it is apparent that a panacea for programming pedagogy has yet to emerge and that one potential way forward may be to achieve incremental improvements and to disseminate them in a way that benefits the computer science discipline as a whole. This may prove to be a successful strategy given the universality of computer programming theory where concepts such as sequence, selection, iteration, parameter passing and algorithm development permeate all programming languages and environments. If these incremental improvements do not emerge then educationalists and policy makers may have to consider the implementation of some form of filtering mechanism in terms of computer science courses. This may be achieved in two separate yet very different ways. The first approach may entail incorporation of computer programming as a subject at second level where potential students have the opportunity through coursework and examinations to have their programming aptitude assessed. This has the added benefit of giving students a flavour of what a programming subject actually entails and the cognitive demands it imposes on them. The second approach would be to implement a programming aptitude test as described by Dehnadi and Bornat (2006b) with only those reaching acceptable levels of aptitude being admitted to courses. Future research and discussion is required to determine which approach will most benefit the computer science profession as a whole.
7.5 METHODOLOGICAL CLARITY

During detailed data collection and analysis it became obvious that there is an obligation on the grounded theory researcher to possess methodological clarity. Methodological clarity, as proposed here, has two dimensions. Firstly it relates to the researcher possessing a clear and unambiguous understanding of the chosen research method and its methodological implementation. Secondly, it relates to the requirement on the researcher to report in a clear and precise manner the exact methodological steps they undertook in their data collection and analysis.

This study has found that it is a useful exercise for the researcher to constantly reflect back on their methodology on an iterative basis in a way not too dissimilar to reflection-on-action as highlighted by Farrell (1998) where after recalling one’s teaching after the class, the educator gives reasons for his/her actions or behaviours. It has found to that it is useful to document each stage of methodological significance and to reflect on its correctness with the purpose of reporting it in a clear and precise manner. For example, with the Strauss and Corbin (1998) approach, the central category is usually an abstract category (a higher order category that describes a number of concrete categories uncovered in the data). Subsequent data analysis found that the central category may in fact be an in-vivo code within the data. As described earlier, an in-vivo code is a code based on a verbatim term uncovered in one or more data sources. For example, during the early analysis of the first sets of data, potential in-vivo codes uncovered were ‘skip-over’ and ‘sink-in time’ and these resulted in becoming actual categories in the theory. Furthermore, early analysis suggested that it could be possible that an in-vivo code uncovered in the data may rise to the surface to the highest level of abstraction once it can pull all the other categories together. In retrospect, this intuition proved correct where the in-vivo code ‘challenging’ eventually emerged as the central category in the theory. In terms of issues encountered like this, where methodological clarity becomes clouded, the researcher is required to constantly revisit the methodology issues in relevant textbooks or literature to gain clarification on proper methodological procedure. This somewhat laborious and time-consuming activity was conducted in this study to ensure proper methodological implementation. This clarity was aided by developing a technique...
Chapter 7

Conclusions

referred to as methodological memoing in chapter 5. Here, there the author describes in a clear and accurate manner methodological issues pertaining to theoretical sensitivity and sampling strategies.

7.6 METHODOLOGICAL ORIGINALITY

In the context of grounded theory, Strauss and Corbin (1998) define diagrams as 'visual devices that depict the relationships among concepts' (p.217) and thus play an important role in enabling the analyst 'to use creativity and imagination' (p.220) which is seen as vital in phenomena abstraction from raw data. In addition, Strauss and Corbin (1998) indicate that in many cases 'each analyst develops his or her own style for memoing and diagramming' (p.220). This original approach to diagramming and memoing was developed in the study and is described in detail in chapter 5. The two original techniques used in this study were association diagramming and conglomeration. Association diagramming enabled two important tasks in this study. Firstly, it used labelling in a way that illustrates the relationship between categories with directional arrows to show the order of the relationship i.e. from category to sub-category. These labels take the form of optionality e.g. 'may involve' or 'may be driven by' which clearly indicates that this relationship may or may not occur at any given time. Secondly, it uses a directional line with the label 'is a kind of' to illustrate sub-categorisation. As was discussed in chapter 5 this labelling approach is taken from sub-categorisation in object modelling where, for example, a part-time employee 'is a kind of' employee. The versatility of this diagramming approach is shown by the fact that it can be used for both axial coding (fig 6.2) and selective coding (fig 6.10). In addition this versatility enables two additional functions. Firstly, its intuitive labelling enables the reader of the various diagrams to generate mini-storylines for each phenomenon (i.e. smaller versions of the storyline memo as developed in section 6.5). Secondly, in terms of selective coding, they facilitate the easy addition of new phenomena (where a new slot is simply created) and their sub-categories as new theory begins to emerge in subsequent data collection and analysis.
In addition, data analysis was considerably aided by developing the conglomeration technique as illustrated in appendix 2. In particular, aggregating all instances of a category into one location significantly aided the identification of properties and dimensions and dimensional variation in the form of ranges e.g. ‘low-high’, ‘short-long’ etc. In addition, explicit identification of relationships in the ‘associated categories’ section enables the analyst to get a more detailed insight into the complex interrelationships between categories.

Future work on this extending diagramming technique could entail developing different icons/shapes for the different component types e.g. sub-category, category (with in-vivo categories being highlighted in some way using a special symbol), abstract category, phenomenon and central category. In addition, computerised tools, similar to Computer Aided Software Engineering (CASE) tools described by Hoffer et al. (2002) could be developed to enable dragging and dropping these components onto the desktop as well as building back-end code to create a data dictionary or central repository of each component in the theory. It is envisaged that this central repository would represent an automated and dynamically updateable version of the conglomeration tables in appendix 2.

7.7 LIMITATIONS OF THE STUDY

Whilst the study was successful in terms of reaching its initial aims of theory generation and programming learner categorisation, a number of identifiable limitations were encountered. The first of these pertained to category saturation where a number of concepts emerged in data collection that did not reach saturation point. In particular, these concepts had particular resonance with the author based on exposure to them in professional teaching practice and as such were noted during data collection and analysis (open coding) as potential categories. The first concept was labelled in open coding as ‘fear of future’ where a student who is struggling with early programming tasks and concepts looks ahead to subsequent material in lecture notes, text books and laboratory sheets and is fearful of the perceived enormity of this challenge. This form of ‘looking ahead’ may instil an even greater sense of negativity
than they already possess and doesn’t take cognisance of the fact that programming is based on a building-block approach where subsequent tasks are built up to in a progressive manner. For example, basic notions pertaining to index positions of characters in strings can be reused when dealing with the subsequent topic of arrays.

Another unsaturated concept was labelled in open coding as ‘theory to practice’. Here a participant recounted understanding what was covered in the lecture ‘in theory’ but when it came to implementing this in a practical laboratory situation they encountered considerable difficulty. Once again this is a common issue encountered by the author whilst working in laboratory sessions. An additional concept also uncovered amongst two participants was ‘regret’. In one instance, the participant regretted the fact that key programming concepts didn’t sink in at an earlier stage. This was attributed to a lack of work ethic in persisting with difficult material. In the other case, the participant regretted skipping over a topic that was crucial in subsequent projects. Whilst reflecting on the theory generated in this study it is likely that the first manifestation of ‘regret’ (if it were saturated) would be encapsulated by the ‘state of mind’ phenomena as illustrated in chapter 6. In terms of the second instance, it is likely that it would be viewed as a negative outcome of engagement in an avoidance strategy.

Another unsaturated concept also emerged in a low-level programming organisation strategy which was labelled as ‘exact example’ in open coding where the participant recounted a strategy of ‘looking for exact pieces of code’. This is also a common experience whilst teaching in programming laboratories where weaker performing students look for an exact piece of code that will complete an entire task or problem. This is contrasted to the actions of better performing students who look for pieces of code that carry out a similar task and subsequently modify it in such a way that it works in terms of the task at hand. In the former, one can see no attempt at knowledge transfer and thus is a very ineffective strategy where the student simply pastes in a fully working code chunk into their program. This concept of knowledge transfer is important computer programming where students must learn to apply concepts like ‘loops’ and ‘if’ statements to new tasks as they arise. In terms of the fragile nature of computer programming learning described in chapter 2, a recent Irish study by O’Kelly and Gibson (2006) indicates that learning mechanisms are required that
transforms fragile knowledge into a concrete transferable skill that can be reapplied in new situations. This finding, combined with those of this study may act as a catalyst for future Irish research in the area of knowledge transfer in novice programming.

The common denominator amongst these unsaturated concepts is that they have all been witnessed by the author in programming laboratory sessions. In terms of data collection, the main disappointing factor in this study was that only three participants submitted diaries/journals. These diaries (illustrated in appendix 1) required the participant to describe the experiences they had whilst working on programming tasks. In hindsight, if this study were to be conducted again a process whereby journals would be given to participants at the beginning of the year and collected at regular intervals (i.e. once a month) would have been implemented. The approach of asking participants to fill in journals close to the end of the year might have been perceived by them as an onerous chore. This effect might have been reduced had the workload been spread over the year. In this regard, the participants might have adopted journal reporting as a regular part of their programming studies where it gave them the opportunity to reflect on their work.

As was indicated earlier in this thesis, only those concepts that reached saturation were presented in this thesis. Practical experience gained in this study suggests that at some stage, the researcher has to draw a line in the sand and present the saturated categories in the form of a theory. Both variations of grounded theory indicate that this is a common situation that has to be dealt with by the researcher. Strauss and Corbin (1998) highlight that the researcher will be faced with practical constraints and he/she just has to make do with what they have. Glaser (1992), on the other hand approaches this situation in a more philosophical manner and indicates that ‘the theory itself should not be written in stone or as a “pet”, it should be readily modifiable when new data present variations in emergent properties and categories’ (p.15). Furthermore, he goes on to state that ‘the theory is neither verified nor thrown out, it is modified to accommodate by integration the new concepts’ (p.15). If one adopts the philosophical assertions of Glaser (1992) in this regard, one can suggest that further studies can be conducted to saturate former unsaturated categories like ‘theory to practice’ as described earlier. In addition, given the ever-changing nature of computer programming languages and environments, it may be the case that in the
future some parts of this theory may be rendered as inapplicable and may ‘fall off’ as new emergent theory is added. This is viewed by the author as a satisfactory facet of qualitative study where all studies, if conducted rigorously, produce valuable contributions to knowledge that can be added to at a later stage.

Finally, whilst chapter 6 produced a detailed account of validity and reliability issues in this study, Glaser (1992) presents the researcher with four simple properties (which were originally presented in Glaser and Strauss, 1967) that determine how well any grounded theory is actually constructed. The first property is that of ‘fit’ (p.15) which relates to the fact that if theory is carefully induced from the substantive area, its subsequent categories and their properties will fit the realities under study ‘in the eyes of subjects, practitioners and researchers in the area’ (p.15). As mentioned already in this thesis, the literature was not analysed until data collection and analysis had taken place. Given the significant amount of overlap between this study and previous work in the area suggests that this theory will be accepted in the computer science education and research communities. Secondly, ‘work’ (p.15) relates to the fact that if a grounded theory works, it will explain the major variations in behaviour in the area with respect to the exposure of the main concerns of the subjects. In this study, variation has not only been produced in the paradigm frames of chapter 6 but also in terms of the learner type categorisation, with the variance in properties and dimensions being produced for each learner classification. The third property, ‘relevance’, according to Glaser (1992) is satisfied if the first two properties, namely ‘fit’ and ‘work’ are applicable to the resultant theory. The final property, ‘modifiability’, described earlier in this section, requires that the theory be modifiable when new data emerges. Figure 6.10 illustrates that this property is applicable to the theory presented in chapter 6 where the unsaturated candidate category ‘regret’ could be facilitated by adding a new category slot to the ‘state of mind’ phenomena. In fact, the presentation of 6.10 illustrates that new categories can be facilitated either by slotting them into the relevant phenomena, or, if necessary, by adding new slots to accommodate emergent phenomena. The only limitation on this expansion of theory is that all new candidate phenomena or categories must be linked to the central category i.e. ‘the challenging nature of programming’.
Chapter 7

In summary, despite the existence of a number of unsaturated concepts and a smaller number of completed journals than initially desired, this section has shown that the theory presented in this study accurately reflects the domain-specific data collected across the four research sites. In addition, its diagrammatic structure, in the form of category association diagramming (as illustrated in fig 6.10) exhibits a flexibility that facilitates the addition of newly emergent categories and phenomena. This is similar to the concept of scalability in computer systems development where a system is designed in such a manner so that it has the ability to scale to support larger or smaller volumes of data and more or less users. Finally, this section has shown that despite some unavoidable limitations, this study conforms to the four desirable properties of a grounded theory as initially prescribed by Glaser and Strauss (1967) when they formulated and developed the methodology.

7.8 FUTURE POLICY AND PRACTICE

This section has shown that the theory developed in this study has raised a significant number of issues at both a policy level and a teaching level. In terms of policy, it suggests that the lack of provision of introductory computer programming at second-level is something that needs to be addressed. The worrying aspect here is that the new Leaving Certificate ‘Technology’ syllabus due to be introduced in September 2007 does not incorporate a programming section. It is the assertion of this research that this policy, if not revisited, may continue to contribute (at least in part) to the problems encountered in computer programming at third-level. This often results in situations where some students who enrol on a computing course are unaware of both the nature of it and the demands it will impose on them. From a teaching perspective, this study has enabled a significant insight into the day-to-day learning issues experienced by students in terms of the emotions they experience and the strategies they employ. Whilst there were many overlapping experiences and strategies amongst students the resounding message to educators that emerged from this study was that each student will encounter a unique experience (i.e. from a theory perspective a personal set of inter-related categories with specific values of properties and dimensions) and this individuality must be attempted to be addressed in some way.
This individuality suggests a need for an ideal learning chain of activities akin to the chain of cognitive accomplishments as described by Linn and Dalbey (1985) whereby a student can be presented with a sequence of material that meets their learning needs and improves their chances of progression. Given the rapid advancements in teaching technologies such as virtual learning environments, on-line assessment and advanced simulation software there appears to be a call to educators and educational institutions to devise mechanisms that cannot only identify the ideal learning chain for programming students but also provide a unique combination of programming material (in the form of code examples, simulations, visualisations etc) that are aligned with the ideal learning chain of the student.

7.9 CONCLUSIONS AND FUTURE RESEARCH

This study has developed a grounded theory of the experiences of novice programming students in Irish third-level institutes. It has shown that the challenging nature of computer programming impacts on both the way students approach the programming subject and their emotions/attitudes about it. In addition, it has used this theory to develop a continuum of programming learner types and indicates that a student not only experiences different categories of learner type but they may oscillate between these different categories. It has shown that in a number of cases many of the emergent categories from the study resonate with those uncovered in previous studies. This suggests that many of the issues experienced by programming students are not specific to a particular location or time. This may be due to the fact that whilst programming environments and languages have changed over the years, the basic tenets of programming i.e. sequence, selection and iteration, like the laws of physics have remained unchanged.

Given the fact that this is the first major Irish qualitative study in the area of novice programming across a number of third-level institutions, it is hoped that this has paved the way for future research. In particular, qualitative studies highlighting topics that students have found 'difficult' in the Java language have been recently conducted in Irish third level institutes (Hyland and Clynch, 2002; Chambers and Madden,
Chapter 7 Conclusions

2002). Future research engaging in mixed method research could entail analysing the results of these studies in light of those emerging from this grounded theory study. For example, areas of difficulty from these qualitative studies e.g. ‘arrays’ could be analysed to see if this difficulty was caused by categories emerging from this study such as ‘ineffective examples’ or inadequate ‘sink in’ time. It may be possible that mixed-methods studies of this nature will allow researchers to not only identify what the main topics of difficulty experienced by Irish students actually are but also gain an insight into answering the why question. In this way, incremental knowledge may be built up pertaining to the actual phenomena that exist in the Irish novice programming context. This knowledge may then be used to refine teaching practice and educational policies that address these issues at a cross-institutional level. In addition, from a global perspective, implementation of quantitative studies on cooperative learning like that conducted by Beck et al. (2005) (where they define cooperative learning as ‘students working together as a part of a collaborative effort to understand material or complete a task’, p.471) in novice programming could be combined with an analysis of the categories peer assistance and self examples that emerged in this study. These examples are an indication of the extensive range of possibilities that are available to extend the knowledge that emerged from this study.

In conclusion, this study has provided a means for Irish novice programming students to describe their actual experiences whilst learning computer programming. A consequence of its substantial as opposed to formal nature is that it may result in it not presenting all of the learning issues encountered by Irish students, but given the fact that data collection and analysis was conducted across four sites, it is reasonable to suggest that has provided an insight into a large number of issues experienced by Irish programming students. Finally, by allowing students to recount their experiences in their own voices it has attempted to do something similar to what all good professional programmers do, i.e. consult the users in advance of writing the software system. Perhaps this is a good starting point for any future programming education reform.
References


References

Computer Science Education (SIGCSE). Norfork, St. Louis, Missouri. (pp. 470 – 474).


References


263
References


References


Appendix 1 – Journal / Diary

Programming Journal

Study Title: Learning Computer Programming in Irish Third-Level Institutions: A Study of First Year Students' Experiences

Instructions to Participants

1. The journal consists of sections for 5 days of programming activity/task where you record in your own words what you did, how long it took you approximately and your comments on the activity.
2. The comments section is where you document in your own words how you experienced the task(s)
3. Please note that if you carry out more than one task in a day you can discuss and comment on all of them. Don’t worry about exceeding a page etc as the page breaks will ensure that each new day starts on a new page.
4. NB Please note that it doesn’t have to be 5 consecutive days, it is any 5 days that you have worked on programming task(s). So, for example, it could be Monday, Wednesday, Friday, Monday, Wednesday.
5. There is also a section at the end where you can make general comments on your experience of programming. Here, as in the other sections you document your own experience of programming in your own words. There is nothing too insignificant to mention.
6. In order to save expense please fill in the form electronically and e-mail it back to me at the below address as soon as you have it filled in.

Many Thanks

enda.dunican@itcarlow.ie

NB: Finally, below fill in your e-mail address and student ID

Student ID Number:

E-Mail Address:
Day 1

What I did: i.e. programming tasks/activities/study

Duration: How long it took me.

Comments: Here your record your experience with the tasks/activities/study
## Day 2

**What I did:** i.e. programming tasks/activities/study

**Duration:** How long it took me.

**Comments:** Here your record your experience with the tasks/activities/study,
Day 3

What I did: i.e. programming tasks/activities/study

Duration: How long it took me.

Comments: Here your record your experience with the tasks/activities/study.
# Day 4

**What I did:** i.e. programming tasks/activities/study

<table>
<thead>
<tr>
<th>Duration: How long it took me.</th>
</tr>
</thead>
</table>

**Comments:** Here your record your experience with the tasks/activities/study,
Day 5

**What I did:** i.e. programming tasks/activities/study

**Duration:** How long it took me.

**Comments:** Here your record your experience with the tasks/activities/study,
General Comments

In this final section please record in your own words any general comments and issues you experience in programming, noting that every experience you have is worth documenting:
Appendix 1 – Consent Form

Learning Computer Programming in Irish Third-Level Institutions: A Study of First Year Students’ Experiences

The following information explains the purpose of this research project. You should be aware that you can withdraw from this study at any time and this will have no effect on your relationship with your department, lecturer or Institute.

The purpose of this study is to examine the experiences of students in a first year programming course. Data will be collected in the form of an interview and/or focus group and/or journal.

Please do not hesitate to ask any questions about the study either before participation or during the time you are participating. I am happy to share with you the findings of the research after it has been completed. Please note that complete anonymity is assured in this research and your name and course will not be referenced at any time.

It is hoped that this research will serve to improve the teaching and of computer programming for first year students. The results of this research may be submitted for journal or conference publication, and as mentioned above no reference will be made to the individual students or the class group.

Please sign your consent with full knowledge of the nature and purposes of the procedures. A copy of this consent form will be given to you to keep.

Signature of Participant Date

Enda Dunican, Researcher, Computing Department, IT Carlow.
Appendix 1 – Information Sheet

Enda Dunican
Institute of Technology Carlow

Study Title: Learning Computer Programming in Irish Third-Level Institutions: A Study of First Year Students’ Experiences

Purpose of Study

As the research title suggests, the objective of this research is to get an in-depth insight into the experiences of first year computing students in computer programming in Irish third-level institutions for my Doctorate thesis. This will be achieved through various data gathering techniques namely:

- Interview
- Focus Groups
- Journals / Diaries

Interviews

Here the participant (student) will be asked open-ended questions on their experiences of programming. During this interview the participant will be asked to discuss in their own words, their own personal experiences of programming. Here, there are no right or wrong answers, and all that is required is for the participant to talk about in an open and confidential manner how they experience the programming, where everything they say is of interest to the project. NB – the interview will not be like an assessment in terms of asking the students what they know, this is not its purpose and will not happen. Nothing of what they say will be discussed in anyway with anyone else. Also, unlike other studies, this study gets the students’ views on how they experience the learning of programming.

Focus Group

Here participants are interviewed as a group where open questions are posed to them and they discuss their experiences in groups. The group sizes are usually about 6-8 participants.

Journals/Diaries

Here participants are given a diary consisting of 5 days, where they can document their experiences in programming in a typical week. Here they can document anything that is relevant to them in a typical week and are asked to write as much as possible. This diary can be provided in either electronic format or hard-copy format. At the end there will be an open section where they can document anything else they find important in terms of how they experience programming.
Appendix 1 – Participation Form

Participation Form

I am interested in participating and would like to participate in the following (you can participate in any one or more of the following):

- Interview □
- Focus Groups □
- Journals / Diaries □

Student ID *

________________________________________

E-Mail Address

________________________________________

Male / Female

________________________________________

* Note: For confidentiality I will not need your name, no reference will be made in the study to your ID as I will be giving you a random, confidential participant ID e.g. P1, P2 etc.

I will contact you by e-mail regarding the time and dates to meet you at your institute and will e-mail you an electronic journal / diary.

Regards

Enda Dunican

enda.dunican@itcarlow.ie
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code : [ACC]</th>
<th>Long Name : Accomplishment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Reported by participants who experience a sense of great satisfaction on completion of a task. The nature of this task can be from as little to fixing a syntax error to completing a large project. For many participants this sense of euphoric satisfaction is one of the key positive experiences in programming.</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Level</td>
<td>Low – High</td>
</tr>
<tr>
<td>Amt of Effort</td>
<td>Low – High</td>
</tr>
<tr>
<td>Interest</td>
<td>Low – High</td>
</tr>
<tr>
<td>Time Duration</td>
<td>Short – Long</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>Level relates to the level of accomplishment and satisfaction actually achieved. This property is directly linked to the other properties, namely amount of effort put in by the student, the interest they have in mastering programming and the amount of time they spend on lab and study work.</td>
</tr>
<tr>
<td>Associated Categories</td>
<td>[SE] [D] [P]</td>
</tr>
</tbody>
</table>

‘It takes a good bit of work for some programming, you get stumped at them, but you just keep at them [P] then when you get something displaying to the screen that’s supposed to be there and you are putting maybe 20 minutes (time) or more and then you get your goal, you get a good feeling’ (P8)

‘When you’re actually successful at doing a hard program, you’re quite relieved and you just love it because it took so much effort (effort) to get it working. If you are not interested (interest) you won’t like it’ (P1)

‘Java is great when you actually get the code working, you feel kind of good as well but the overall outcome when you do it is a good feeling, but its tough [D]’ (P9)

When you are doing something and it works for you, you feel like you’ve achieved that task’ (P14)

‘You’re doing something and when you succeed there is a reward at the end, if you get something to work it’s amazing (level). If you can’t get it done it’s frustrating, but when you get it to work, it’s amazing’ (P19)

‘I try to write my own programs [SE]. We are doing maths problems and just to practice, I wrote a program to do it. I find the tasks I am given, I don’t know about them. I prefer an application you know yourself, you get a lot more satisfaction about it’ (P22)

‘It has been satisfying, writing programs that work. It is an unbelievable feeling when it works. It makes it feel worthwhile’ (P22)
‘The first time you get a project you want everything to work. You want a proper program, doing something you are proud of’ (P37)

‘Labs are more interesting and it’s more rewarding to see everything coming together on the screen’ (P39)

‘It’s hard work but rewarding. At the same time it’s very frustrating. The longer it takes, the happier you are when it gets done’ (P39)

‘I enjoy project work, especially now we are working on stuff that not small and stupid, which you’ll never use. It’s something you can say afterwards “I did that! I’m proud of it, it works”’ (P36)

‘If you’re just learning (rote learning) something, like history or something, there’s no real kick out of it. But in programming when you get something right, you have been working at for an hour or something [P], you say to yourself “hey I’ve got it working”’ (P35)
**Appendix 2 Conglomeration**

<table>
<thead>
<tr>
<th>Code : [ALC]</th>
<th>Long Name : Alien Concept</th>
<th>In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Reported by participants who find computer programming and its associated abstract concepts something they are totally unused to. A key issue that emerges in the data is the absence of computer programming from the Irish education system at both primary and secondary school levels.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Low – High</td>
</tr>
<tr>
<td>Extent</td>
<td>Partial - Full</td>
</tr>
</tbody>
</table>

**P & D Overview**  
*Level* relates to the level of ‘alieness’ experienced by participants in terms of the programming concepts covered in first year. For example they may find parameter passing more alien than sequential processing. This appears to be high when students have had absolutely no exposure to programming of any form. The *extent* property pertains to the extent of the programming material/concepts within the syllabus that the participants find alien. For example, if we look at the data below some participants exhibit a low level of extent where they have done programming before. For those participants who have programmed in Visual Basic (VB) before the low level of extent they exhibit pertains to object-oriented concepts of classes, methods, encapsulation etc.

| Associated Categories | [BW] [D] [PB] [SLFDBT] [PPK] [P] |

'I found it very tough [D], it was completely (extent) alien to me and when I started looking at programs, sitting in lectures scratching my head and I wasn’t really sure what was going on [BW]'  (P8)

**Memo** - *Here the participant seems to describe a state of bewilderment when encountering the programming subject. It will be interesting if further participants recount this type of experience and how they deal with it*

'As I said programming was completely alien to me. I’ve a better understanding of it now. I’ve been keeping on top of things and not letting them build up [PB], that’s the biggest thing really’  (P8)

'They (loops, ints, booleans etc.) were something you never done before, and you are sitting there like, am I doing that right [SLFDBT], I wanted to be sure’  (P14)

'It gets tough [D], at the start you are like trying to learn a whole new language which I’m really bad at [SLFDBT]’  (P19)

'At the start I was struggling because it’s all new and I kind of figure it out in my head’  (p19).
Classes were difficult to understand, not classes, but the concept of classes. As I hadn’t seen it in Visual Basic before, it was something new, apart from that I didn’t have any problems. Classes were a major difference to what I had seen beforehand’ (P20)

Memo – once again we see a different dimension to the alien concept code. Here the student is isolating classes as a singular alien concept. It appears that this student is doing well in programming. This differs from past interview where many of the participants don’t refer to actual programming concepts but rather speak in general terms. It seems to be emerging that students who are more comfortable with programming are less reluctant to explicitly mention actual programming concepts like classes, inheritance etc.

‘Java syntax has a lot of abstractness, it is not so similar to the way we speak’ (P18)

Memo – I feel compelled to contrast the use of the term ‘abstractness’ to the term ‘words’ used by students struggling with the programming subject. It is emerging that beacons like this indicate to you how comfortable and successful a student is with the programming material

‘For someone who doesn’t have any experience at all it was kind of hard because people are not taught how to think logically’ (P18)

‘When you are writing Java code it is totally alien’ (P24)

‘It was hard at the start, trying to get used to something I hadn’t done before. I started to get used to it’ (P21)

‘With most subjects in college, they kind of continue from secondary school, but there was nothing like this in secondary school. Everything is so different’ (P29)

‘Well classes were definitely new. That was the first major thing that was different for me. It was a totally new way of thinking about it. The whole thing about encapsulation and keeping the different parts of the program separate from the other parts. The whole thing about object-oriented programming really because I hadn’t come across it’ (P28)

Memo – ‘Here we see a different dimension to this code. The student describes not programming itself as an alien concept but rather the object-oriented approach to programming with classes, methods and encapsulation. This is in start contrast to finding programming itself as alien. Once again we see that the students more comfortable with programming are happy to discuss their experiences in terms of programming concepts.

‘It’s good to have examples you are familiar with and you say “oh yeah I’m familiar with that”. That doesn’t happen in programming so it requires a lot more sitting down and slogging through the work [P] (outcome), whereas in maths you are expanding on knowledge you already had, because you have been doing maths for years’ (P41)
'It was a steep learning curve. I had never done any programming before and I was thrown in at the deep end. It's been tough. I won't lie, it's been tough. Everything else I was expecting, the maths etc but the programming just threw me completely' (P42)

'It's difficult [D] because it takes a lot more brain power. The rest of the subjects came naturally to me but the programming hasn't. Programming has been something 'I've had to work at it. The other subjects have sat OK with me' (P42)

'Programming is so new like, you've never done anything like it before. Like maths and stuff you can learn theory but in programming there is only so much theory. Coding, that's the big difference. Getting used to sitting down and actually coding is hard to get used to' (P43)

'Programming is new stuff in general. In other subjects you do a lot of maths that you've done in your leaving certificate, so you have seen a bit of it before. You should so a bit of programming in school' (P45)

'Completely new concepts, I'd done nothing like that before' (P36)

'I've done maths before since I've been in national school. Programming is a totally new thing for me this year and I've never done it before, I'm not sure what to do when I get stuck' (P44)

'The first time I saw "switch statements I found it hard [D]. It was different to what we did before. Doing more examples, using it more made it easier [P]' (P19)

'The introduction to the fundamental concepts of software development with emphasis on problem solving and computer programming was a new experience to me, something quite new (level)' (P4)
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code : [A00]</th>
<th>Long Name : Awareness of Others</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This relates to participants recounting overt awareness of the ability or progression of others on the programming course.</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td><strong>Dimensions</strong></td>
</tr>
<tr>
<td><strong>Nature</strong></td>
<td>Negative – Positive</td>
</tr>
<tr>
<td><strong>P &amp; D Overview</strong></td>
<td>Nature relates to whether this overt awareness of others on the course is positive or negative. It appears to be negative for those who see others as progressing in an easier fashion than they are. It seems to be a positive experience for those who see others as being in the same boat as themselves and therefore is an incentive to work harder. It appears neutral for those who are happy with their own satisfactory progression.</td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[SI] [P] [D] [BW] [SLFDBT]</td>
</tr>
</tbody>
</table>

'I seem to get errors that everyone else doesn’t, you need static here, a colon here. So it’s just going wrong, it’s just going wrong for me, “disaster!”' (negative) (P2)

'You realize that you’re not the only person who is struggling with it. Everyone’s struggling, trying their best' (P19).

'Some people just hear it once and can do it straight away, I have to sit down and study it for an hour, 10 pages of notes, studying them for an hour to get it embedded into my brain, so I could use it properly’ (P19)

'I’m conscious that people need to learn and haven’t had the same amount of experience as me’ (P20)

Memo – Here we see two new and interesting angles on pace and awareness of others. Because this student has had considerable experience in programming they find the initial pace slow and their awareness of others is on the other end of the continuum, where they are overtly aware of the difficulties and struggles of others rather than seeing their own ability as being less than that of others

'Starting with Java seemed a bit hard especially seeing my classmates struggling with Java. For someone who doesn’t have any experience at all it was kind of hard because people are not taught how to think logically’ (P18)

Memo – once again with this student we see instances of codes with different dimensions than before. Here the student is overtly aware of the difficulties others are having. It may be the case that it is this very overt awareness that makes the less capable students even more aware of their perceived inadequacies. It is interesting to see how the same code/concept can be uncovered with two totally different ends of the continuum.
‘A lot of people have been taken aback, some have left the course’ (P10)

Memo – It is interesting to see in the data that students progressing satisfactorily in programming are not phased by the negative experiences of others.

‘A lot of people in the space of a few weeks got lost and myself, I’m just starting to come back into it (positive)’ (P10)

Memo – this student seems to be able to come out of temporary glitches successfully

‘When you can’t do a problem, you don’t know what to do in a lab. A few fellas might have programmed before and they’d be well up for it, but the general consensus in the class would be sitting there looking at the screen [BW], thinking “what do we do?”’ (P10)

‘I have seen a couple of people in the class already sinking fairly fast, but it’s not too bad if you attend all of the classes’ (P37)

‘Some people go in and see the course content and are absolutely terrified. They flick onto some of the advance code and they see what they’ll be doing in a couple of months or in a year or two or something like that and they are absolutely terrified. A couple have left the course for that very reason. If you go in with an open mind and realize it’s a subtle build up and that you are not going to be dumped with everything then its fair enough and its manageable and enjoyable’ (P41)

‘If other people are having similar problems you can listen into their same problem’ (P45)

‘You want to get over it and you have to get over it because you know that other people go through the same problems’ (P22)
Appendix 2

Code : [BW]  Long Name : Bewilderment

| Description | This relates to a sense of being lost in the programming subject and not knowing how to continue. Participants experiencing this phenomenon seem to be aware that remaining in this state catches up on them and they have to get out of this state if they are to satisfactorily progress. |
| Properties | Dimensions |
| Duration | Short – Long |
| Intensity | Low - High |

P & D Overview: Duration relates to how long the participant stays within this state of bewilderment. Intensity relates to the effect of this state of mind which seems to vary in terms of its impact on the student.

Associated Categories: [AOO] [PB] [D] [ACC] [CHALL] [CONF] [P] [SLFDBT] [SI] [W]

‘When I started looking at programs, sitting in lectures scratching my head and I wasn’t really sure what was going on’ (P8)

‘I think it was hello world, I was like, this is going to be grand, you know, and then we started getting into methods and arrays and I was lost, in a world of my own’ (P9)

‘The next week builds on what you didn’t understand anyway so you are really lost (intensity) [PB]’ (P24)

‘I like programming when it works but if it doesn’t you are in big trouble, you have no way of catching up’ (P24)

Memo – Once again we see an example of this double-edged sword that programming seems to present, affability for the subject mixed with feelings of doubt and bewilderment

‘The general consensus in the class would be sitting there looking at the screen, thinking “what do we do?” [AOO]’ (P10)

‘Semester 1 was alright, I got kind of lost in semester 2 and had to pick it up, but then coming into semester 2 it’s like the bar has been raised’ (P10) (Properties duration & intensity) (P10)

‘You are sitting there and wondering what is the lecturer talking about and that’s hard [D]’ (P42)

‘Really make sure you know you’re going to be doing a lot of programming. It’s not easy and you can get caught up so easily if you don’t know what you are doing. You really have to know what it’s going to be about’ (P43)

‘With Java I don’t know what’s behind the scene’ (P24)
'It doesn't get any easier. If you are finding the programming hard [D] what do you do? Do you keep going ahead or do you stop? (duration) You get more disillusioned if you don't have the backup when you're working on things' (P24)

Memo – Here we see the student experiencing bewilderment and also a sense of isolation with no apparent backup mechanisms to assist them
<table>
<thead>
<tr>
<th>Code : [CHALL]</th>
<th>Long Name : Challenging In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This is probably one of the most common experiences exhibited by participants (even though they do not explicitly use the term ‘challenging’). In particular it appears to be either a positive or negative issue with participants.</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Nature</td>
<td>Negative – Positive</td>
</tr>
<tr>
<td>Intensity</td>
<td>Low – High</td>
</tr>
<tr>
<td>Extent</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>Nature refers to whether the student finds challenging as either a positive or negative experience. Intensity is an interesting property to the extent that low intensity may not be a positive factor for the high-achieving student who requires a certain level of intensity to avoid boredom (e.g. P5 below). As in a previous category, the extent property pertains to the extent of the programming material/concepts within the syllabus that the participants finds challenging. For the weaker students, they seem to find this as leaning towards the full end of the continuum. Whilst this may have a negative impact on them, a significantly partial extent for stronger students may also have a negative effect to the extent that it instills a sense of boredom.</td>
</tr>
<tr>
<td>Associated Categories</td>
<td>[MD] [P] [BW] [GH] [D] [PB]</td>
</tr>
</tbody>
</table>

‘The pace [MD] is challenging’ (P14)

‘It's challenging, a lot of new concepts and ideas. I find myself making silly little errors, leaving out curly brackets and stuff’ (P18)

“In programming you have to learn how to make your own stuff up, your own programs. Other subjects are just notes that you have to memorise and write down’ (P18)

“It's something new and challenging’ (P19)

‘I knew the programming was going to be a massive (intensity) challenge, it’s kind of something that has to be worked at really hard [P]’ (P25)

‘For the most part it’s interesting and challenging. The methods caused me quite a bit of grief at the beginning. But overall it is fairly good (nature)’. (P29)

‘It is interesting. The way it is taught is interesting. It’s challenging enough as well to keep us interested all the way (avoid boredom)’ (P28)
Memo – This is a very interesting slant on the ‘challenging’ code. Here challenging is seen as very much a positive experience. In this experience it is challenging enough to keep the students interested.

‘Programming is definitely the most intense part of the course. It is the one subject where you definitely have to keep your head above water. If you start falling behind you’ve lost [BW]’ (P37)

‘Interesting, it started off not that hard. I kind of got my head around it pretty fast. I didn’t have any problems, whereas now as we’re getting on to harder stuff [PD] [D] it is getting challenging (extent). I enjoy project work, especially now we are working on stuff that’s not small and stupid, which you’ll never use.’ (P36)

Memo – Here we see another dimension to the challenging code uncovered in earlier interviews. This student is finding the programming very manageable but at the same time challenging. For the better performing students it seems this challenging element is one of the things that keep the students interested in the programming subject.
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code : [CLAREX]</th>
<th>Long Name : Clarifying Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This refers to an example that clarifies or crystallizes a programming concept e.g. a ‘for’ loop or ‘parameter passing’. This may take the form of a chunk of code on its own or with some sort of diagrammatic or simulated example e.g. counting from 1 to 10. For some of the participants clarifying examples were those that were couched in terms or examples that were familiar to them but this was not always the case, in fact it may be in the form of something new. From speaking to participants, one gets the idea that either describing in terms that is familiar to them or something new that sparks their interest is what transforms a clarifying example into a clarifying example.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Familiarity Level</td>
<td>Low – High</td>
</tr>
<tr>
<td>Size</td>
<td>Small – Large</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>As mentioned above familiarity pertains to how closely the example relates to something the student is familiar with. Size relates to the how big the code example actually is. From speaking to the participants they use terms like ‘small’, ‘easy’ ‘basic’ etc which indicates that a small discrete example that succinctly explains the singular concept is desirable.</td>
</tr>
</tbody>
</table>

| Associated Categories | [D] |

‘I have a couple of examples that stick with me’ (P37)

‘I’m always looking for an example somewhere, once I have an example it’s usually set in stone’ (P37)

Memo – this statement really crystallizes this code for me. The hallmark of a clarifying example is that when it clicks in it sets the concept in stone and the student can revert back to it any time they wish.

‘My lecturer sat down with me and had a nice little diagram that was like a tree basically where every branch came back to the main trunk of this tree. My lecturer then showed me what I had which looked like a car crash basically. That one little diagram stuck with me and that’s what showed me the light’ (P37)

‘At first I try to get the most basic use for whatever. Say it is a pointer or whatever. What is the most basic program you can write using a pointer and then just keep moving to more difficult ones [D]’ (P39)
‘It threw me until it was explained using an example I was familiar (familiarity level) with. I probably would have listened if that example was given in the lecture instead of drifting off’ (P42)

‘It’s good to have examples you are familiar with and you say “oh yeah I’m familiar with that”’ (P41).

‘It’s a lot easier if you have a smaller (size) program than the one you have to do. Actually seeing it done instead of just talking about it. I’d like to give a small example and then build a bigger one as in go through the stuff’ (P45)

‘An easy example of what we were trying to use, for example if it was a loop just an easy example counting to 10 or something like that, something basic and easy. Maybe a class that had 4-5 functions and how to call them and how to call them different ways, to explain it the simplest possible way’ (P44)

‘I saw a clear example of how it works’ (P24)

‘Clear simple examples of where it actually works’ (P24)

‘I’d prefer if it was related to things in the real world and not abstract things’ (P24)
Appendix 2 Conglomeration

<table>
<thead>
<tr>
<th>Code : [CLK]</th>
<th>Long Name : Click In In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>An in-vivo term used by participants to describe when a programming concept registers with them. The data suggests that this is a form of a ‘eureka’ moment when the student accommodates the concept under consideration. This type of instant occurrence is typified by the term ‘suddenly’ in the data.</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Extent</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>Speed</td>
<td>Slow – Fast</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>Extent refers to how well something clicks in. In the interviews, participants refer to programming concepts as ‘sort of’ and ‘kind of’ clicking in. Speed relates to how quickly something clicks in with the participant. In the interviews the participants use terms like ‘suddenly’, ‘taking ages’, ‘longer’ times than others.</td>
</tr>
</tbody>
</table>

‘Well usually, I don’t understand it at the start [SI], it kind (extent) of clicks in at the end, I keep on doing it’ (P5)

‘Spending a lot of time looking at the code, then it suddenly clicks and then you go “that’s where it was wrong”’ (P14)

‘If statements clicked with me straight away, looking at them can be confusing [CONF], but the clicked with me straight away for some reason, its probably because I practiced with them an awful lot (time)’ (P14)

‘Even though I couldn’t do it for a while, suddenly it just clicked and I did well in continuous assessments’ (P19)

‘We sit down and work on it, that helps. You’re helping them and they’re helping you and once you get it done it clicks in my head’ (P19).

‘I’m hoping that somewhere along the line I’ll get an understanding of this, when is it going to happen? When is it going to click in? When you are at home on your own, it’s very difficult [D] to do it. It feels like you’re beating your head off a wall or something’ (P25)

‘I wish I started the way I feel now. I’m sorry this hasn’t clicked earlier. Maybe it takes somebody a bit longer (speed)”’ (P25)

‘Just sitting it out. Working it over and over until it clicks [P]. There is a moment when it just clicks and you understand it’ (P29)
‘There have been a couple of topics where I’ve sat there and realized that I didn’t get that topic at all. I just read through the notes a couple more times, thinking about it and it sort of clicks a little bit. I’d say for the topic to fully integrate into my mind it would take ages for me. To get the initial sort of point behind a topic it normally only takes a day or so, not that often that would happen in lectures because the topics are so intense [SI].’ (P41)

‘Sometimes being told about it in lectures, it doesn’t click in your head. Whereas if you are actually sitting down and doing it yourself you kind of understand exactly what’s going on. Not all of the time, some topics might not have a problem at all but some things I don’t think you can really learn at all from notes. You just need the experience behind you to be able to do it’ (P36)

‘I’m struggling a bit, I got my head around most of it so just a few little bits and pieces which haven’t quite clicked’ (P36)

‘Mine is the same as yours, this is not working, and then they tell you (where yours is wrong) [PA] and you say ah, that helps a little bit, doing the next one and getting it right’ (P19).

‘When you get it working yourself and it just clicks in your head and you see why it works. Using them in a project and eventually they all come together’ (P16)
## Code : [CODEX1]

<table>
<thead>
<tr>
<th>Long Name : Code Examples</th>
<th>In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This refers to code examples provided in lecture notes, text books or from other sources (with the internet becoming an increasingly popular source of code example). A code example is usually provided in an attempt to explain how a programming concept works.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Easiness</th>
<th>Low – High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Small – Large</td>
</tr>
<tr>
<td>Explanatory Power</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>Modifiability</td>
<td>Low – High</td>
</tr>
</tbody>
</table>

**P & D Overview**

For the vast majority of participants an example with a high level of *easiness* i.e. an example that is easy to understand was seen as desirable. The *size* of the example also seemed to be an issue with as small an example possible to explain a concept coming over as desirable. In the majority of cases a high level of *explanatory power* is required in terms of getting the concept across to students. *Modifiability* is also seen as a desirable attribute of a code example with those with a high modifiability level useful in projects and lab work. It should be noted that the properties of *easiness* and *explanatory power* are separate given the fact that an easy to understand example may not have the requisite explanatory power to fully describe a more difficult programming concept e.g. polymorphism in object orientation.

**Associated Categories**

[D] [LEC] [P] [PA] [SE]

---

'I do the first question by using the example in the lecture notes [LEC]. I’d try and do the second question by referencing back to the first question’ (P3)

**Memo** – *Here the student appears to have a particular strategy in terms of lab questions*

‘If it goes in OK at the lectures I take notes. If I have difficulty [D] I go back over it again [P] try the example. If that works I move on, I’ve understood it’ (P24)

‘I look for examples anywhere in the lecture notes. Examples are far handier than getting a bit of theory in front of you. Try the example until you get it working [P]. If you get the example you’ve got it nailed down’ (P37)

‘Look at every example you can get your hands on. Once you have done all the examples you will be surprised how far easier the course is going to be for you in the long run’ (P37)

**Memo** – *here the student is really reinforcing their belief in doing as many examples as possible in order to get to grips with programming*
'I take notes and go off to the computer and do example programs and try them out. Usually some of them are fairly easy (easiness), the more you put it into practice the more you understand it' (P39)

'Give me an example, ask me to do a couple of things and then show me the solutions. That would be a lot better' (P42)

'Keep reading over and over the notes [P], if there are any examples at all then keep doing them until you understand [D], if they are complicated it is hard [D]' (P43)

'I check books and notes to see if there is anything similar to it. If I can modify that or depending on what it is I might check the internet to see if there is information on it' (P36)

'A well commented piece of code that explains everything (explanatory power). I find at the moment that they are going into very complex stuff, kind of [pause] it’s to do with it but not really what I’m looking for' (P36)

'We have examples of applets using arrays and not applications, I most likely implement the code from the applet into the application and try and modify (modifiability) it till it works [P]. If that didn’t work I’d turn to people in my class [PA]' (P3)

'If it goes in OK at the lectures I take notes. If I have difficulty [D] I go back over it again, try the example. If that works I move on, I’ve understood it' (P24)

'I find it tough, but I seem to get good results, it’s tough but it’s doable. I find this if you start off with smaller examples instead of being thrown in at the deep end’ (P10)

'You pick it up from the different little examples it’s being used in’ (P35)
### Description
This is a common issue reported by participants who indicate that they are confused by things in programming such as concepts, exactness of syntax, regimental nature of programming language and layout etc. It was interesting that the students more comfortable with programming seemed to be able to put confusing issues with the respective language down to nuances that had to be accepted.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extent</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>Duration</td>
<td>Short – Long</td>
</tr>
</tbody>
</table>

**P & D Overview**

*Extent* refers to how much of the material on the programming syllabus the student finds confusing. If this for example, is partial it might only pertain to abstract classes in object orientation. If this were the case it would have far less impact than a more fundamental concept like selection control (i.e. an “if” statement). In the data we see participants referring to things like having to know ‘everything’ and ‘so much’. *Duration* relates to how long this feeling of confusion lasts during the year. For example one participant indicates that ‘it becomes a little bit clearer’ and others refer to stuff that was confusing eventually ‘clicking’ with them.

### Associated Categories
[D] [BW] [ISO] [CLK] [DISENG] [IE] [SLFDBT] [SO]

“‘If’ statements clicked with me straight away (duration), looking at them can be confusing but the clicked with me straight away for some reason [CLK], its probably because I practiced with them an awful lot ” (PI4)

‘It has “if this, blah, blah” then switch does one of them and you have to figure out which one’ (P18)

‘It’s the Java language, you have to know everything (extent) to go and do it. In VB you have to know the code as well but it’s more straightforward’ (P18)

*Memo- it seems here that the all or nothing concept is a serious issue for this student*

“I get some errors and I go in the next day and say “what the hell is going on? [BW]”” (P19)

‘You can’t do this, you just can’t figure out why it is there, what’s the point of having a class separate to there, inheritance, just put it all into one class and do it like there, having separate classes, all doing inheritance, interfaces and all, its just crazy’ (P19) – quantity - property
‘When we started getting projects, I’m trying to imagine how to make the project. Get the problem, write pseudocode for it. It’s like a block and you are trying to imagine how you would make something work, how does it actually happen?’ (P22)

‘It becomes a little bit clearer but you are still saying, how does this work? You have a left side and a right side. You are supposed to put the constructors into this compiler. You can’t just throw in any stuff. It has to be set out the way the compiler understands’ (P25)

Memo – here the student appears to be confused and somewhat dismayed by the exactness of programming compilers.

‘Normally straight after lectures you are straight into labs and it’s fairly confusing. You have to sit there and work it out yourself [ISO]’. (P29)

‘Methods are more intricate in the way they work. The rest of the stuff before that you type one line and it will do most of the work, but with methods you have to be perfect’ (P29)

‘Lectures can be quite daunting, classes are what we are on now and the lectures they just seem crazy to me. There’s like set and get and there is so much happening (quantity), but you sit down in the lab, it’s hard to understand [D] just looking at notes’ (P29)

‘People get confused by all the brackets and semi colons. Also the way the classes and objects are separated by dots. It is a lot to understand at first when you are also trying to understand the concepts of iteration and selection’ (P29)

‘The dot stuff confuses people. The first time I saw it I said that’s just weird. I thought why isn’t it just print and then the line but then I got that, it’s like a method of a class of a package. It was off-putting at the start. I always want to know why something is that way. Anyone who’s like that found it off-putting wondering why it is that way. I suppose I’ll have to accept that there will be stuff that you won’t understand. You can kind of ignore the structure and just look at the statements inside it. It was easier to accept the structure as that’s the way it was’ (P29)

Memo – Here we see the student who is more comfortable with programming can deal with the confusing stuff in a different way. They seem to be able to accept syntax and slightly confusing layout as nuances of the language that just have to be accepted.

‘The last project, I’ve been racking my brains over that, just trying to figure it out, but I’m getting there slowly but surely’ (P37)

‘But when the lecturer is going through loops and stuff like that I try to follow but sometimes I just can’t. I say to myself “what’s going on?”’ (P42)

With functions I just didn’t get them at all and I said I’ll get it all sorted out later but just didn’t bother [SO] and it took a whole project to figure it out. The passing of functions and the parameters completely confused me’ (P42)
When you are going through it and there's no variable names, it's all "i", "j" and "k" where we have been using variables like count and number. I don't like the "i" and "j", they confuse me" (P42)

"Pointers are so complicated anyway, you lose track of them so easily. (P43)

'The fact that you run through the "if" only once, whereas the loop because you are used to seeing the program going line by line and then when you get to a loop, you just go over and over it again until you complete what the condition is. That sometimes might get confusing’ (P22)

'If you miss a day (context) and then you come in and see "i" and "j" and stuff and you don't not what the hell they are on about’ (P22)

'The best way to learn it first time, call it equal to the variable + 1. It makes it easier for people to understand. You are just adding one to the same value you had before. If you say the "variable++" you are just confusing people (alternative context, unintelligible layout/structure)' (P23)
Appendix 2 Conglomeration

<table>
<thead>
<tr>
<th>Code : [D]</th>
<th>Long Name : Difficult In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Reported by participants who find the programming difficult. A common Irish term for difficult is ‘hard’, e.g. ‘I’m finding the programming hard’. Other terms used are ‘tough’ and ‘tricky’. As indicated in the properties and dimensions of this category, this notion of difficulty may either have a positive or negative impact on the student. The pervasiveness of this category is evident in the amount of other categories it has relationships with.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
<td>Negative - Positive</td>
</tr>
<tr>
<td>Extent</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>Duration</td>
<td>Short - Long</td>
</tr>
</tbody>
</table>

**P & D Overview**

_Nature_ relates to whether difficulty has a positive or negative impact on the student (a student in the middle ground of this continuum in the data would describe it as ‘just OK’). In the data we see that many participants refer to programming as ‘enjoyable but tough’. This seems to be a common experience for students who enjoy programming even though it makes considerable demands on them. _Extent_ refers to how much of the material on the programming syllabus the student finds difficult. In the data we see participants referring to ‘so many’ and ‘an awful lot’ and ‘hundreds of different ways’ etc. _Duration_ relates to how long this sense of difficulty remains with the student. In this regard we see participants referring to things like ‘at the start’ ‘you are gone for the rest of it’, ‘since the beginning’ etc.

**Associated Categories**

[S] [ALC] [ISO] [MD] [SLFDBT] [CODEX] [ENJOY] [CONF] [GH] [DISENG] [IE] [P] [SE] [SO] [ACC] [AOO] [BW] [CHALL] [CLAREX] [CLK]

It’s a bit difficult, there’s a lot to learn like all those arrays and stuff, I don’t know I’m not catching it in lectures [SI]’ (P2)

‘Personally I found it very tough’ (P8)

‘It’s a great course but it’s tough, once you keep on top of things it’s grand’ (P14)

‘It’s hard but then again I haven’t failed anything, it is interesting. There are just so many different functions, methods and all that but then again it’s interesting, knowing what each one does separately (nature – positive)’ (P14)

‘There are so many different things in Java programming, it’s a lot’ (P14)

‘It gets tough, at the start you are like trying to learn a whole new language [ALC] which I’m really bad at [SLFDBT]’ (P19)

‘I am enjoying it, I found the first semester ok and did well with it, I am finding this semester tougher [ENJOY], the programming has upped the tempo’ (P24)
Appendix 2

‘You don’t have a lot of time [MD] to do stuff so it’s tough going’ (P19)

‘I find it tough, but I seem to get good results, it’s tough but it’s doable. I find this if you start off with smaller examples instead of being thrown in at the deep end’ (P10)

‘It comes in big chunks. If I had been sitting at a computer and been told this is a constructor and we were told how to do it. The lecturer explained how to do it but when you sit sown at the computer to do it, it’s a different story’ (P10)

Memo – This in my lab experience is a common issue for students. This student describes it very simply and accurately where the leap of faith required to actually do a new example on your own can be quite difficult and daunting

‘Arrays were too hard at the start to get to grips with. Two-dimensional arrays were hard.’ (P21)

‘If statements and for loops, I got to grips with fairly easily but the stuff we are doing now with the public and private methods. I find that a bit hard. It’s easier to understand the “for” and “if’s” because you can put in what they are doing. But the methods, you have to think about it, they are more complex. I’d like a few more examples [CODEX]’ (P21)

Memo – Here the student provides a very interesting insight into how and why they find certain material difficult. ‘if’ statements and ‘for’ loops are ok because they can control the control variables and incremental values of the ‘for’ loop and they can decide on the exact nature of the ‘if’ statement they want to implement themselves. However, they find that they have lost this feeling of being in control because of the perceived complexity with methods. This issue is emerging in the data where for many students there is a point where they lose conceptual control of the material and how they deal with programming after that point may determine how they progress with the programming subject.

‘I found the programming extremely difficult to get a grasp of’ (P25)

‘It has definitely been difficult, I wouldn’t say I was struggling with it, but it’s difficult to get into it. Overall it has been enjoyable [ENJOY]’ (P39)

Memo- Once again we see this double edged sword where the student experiences difficulty yet enjoyment

‘Everything else I was expecting, the maths etc but the programming just threw me completely’ (P42)

‘It’s alright, it’s the trickiest part of the course, easily the trickiest part but you really need to have a basic understanding. If you don’t get the basics on the topic you are gone for the rest of it (outcome). It’s OK, it’s not the easiest though it’s just OK’ (P43)
'You try googling stuff, if the lecturer isn’t there then there’s nothing that you can do [ISO]. The programs are really hard. (P43)

'Coding, that’s the big difference. Getting used to sitting down and actually coding is hard to get used to’ (P43)

'It’s one of the harder subjects, it takes up a lot of time. There’s a lot of errors and stuff like that you have to figure out. There is an awful lot to programming (extent). When you take one concept by itself for example doing an array and then you put pointers and loops and everything all together it gets very complicated [CONF]' (P45)

'There are hundreds of different ways to do things, just trying to get your head around it and also the programs are a lot longer and it’s harder to see the whole picture’ (P36)

'Difficult, I have been struggling at it since the beginning’ (P44)

'It’s tough, you are thrown in at the deep end. It’s not like you think it is, it’s much harder’ (P44)

'Parts are hard and parts of it have been easy’ (P24)

'How to form a method, how do I put arguments into it. The two arguments I had were arrays. Things like that are hard to get. How to create them, to do one that returns a Boolean with two array arguments, I haven’t a clue’ (P22)

'We continued using arrays but now we started using applets and it seemed a lot more difficult. We had to create an array with set values and show the following. Numbers that were greater than 5 and then show all numbers that were a multiple of 4. The general consensus in the class was that we couldn’t do it and we found this extremely difficult. I worked on the applet for an hour but to no avail [P]. These were the first applets we did since coming back from semester one and I felt that I wasn’t as sharp with applets as I should have been so I asked the lecturer could they repeat a class on applets. I believe that applets are extremely difficult and a lot of attention needs to be emphasized on these from the beginning and that they need to be revised regularly. Loops also need to be paid attention to because loops are used regularly in programming. I also find that in semester one I was really excited learning new things in java but in semester two it doesn’t seem as interesting and I am finding that I am not striving to learn more in java. If you could some how introduce something exciting in semester two I am sure it would keep students interested’ (P7)

'I found the whole topic of recursive functions very difficult to understand and this wasn’t helped by the way we were asked to write an iterative function in the middle of learning about recursive ones. I’m aware we need to know one before the other but not enough time was spent on the basics of this topic’ (P43)

*Memo* – Here we see the students who are progressing well with programming focus on specific topics that they find difficult, this as an excellent example of focusing on a recursive function/method
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Long Name: Disengagement</th>
</tr>
</thead>
</table>
| Description   | Reported by participants who consciously disengage from part or all of a programming concept. By disengage here is meant that they do not try to understand the concept/example/issue currently being covered in the programming course. As illustrated in the data this may take many forms from being a strategic decision in terms of focusing energy on other parts of the computer science course to that of a reaction to difficulty or confusion. Terms used by participants are ‘switching off’, ‘drifting’ and ‘fading’.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>Frequency</td>
<td>Rare – Often</td>
</tr>
<tr>
<td>Duration</td>
<td>Short - Long</td>
</tr>
</tbody>
</table>

| P & D Overview | Level relates to the amount of the programming material the student actually disengages from. In the data we see participants deciding to ‘pick it up again’ and putting some material to the back of their mind. Frequency relates to how often a student disengages from material. In the data we see ranges here from ‘sometimes’ to more permanent/absolute terms like ‘switching off’ for those students who find programming concepts too difficult to come to terms with. Duration pertains to how long they disengage from a particular topic, where participants mention they disengage ‘for a while’ or indefinitely by indicating getting it ‘eventually’ or ‘I’ll sort it out later’ |

| Associated Categories | [D] [CONF] [GH] [EXSCH] |

‘…. a lack of concentration in that lecture probably’ (P14)

‘When I came in first, you’d be kind of going “I want to do this!” like starting a new job, you want to work a lot harder and then as time goes on you get more lazy or something and say to yourself “I’ll catch onto it later”. So with a lot of the concepts I probably think it’s a lot harder [D] than it actually is, because you are not giving it as much effort as you did at the start. When you start something new like “if” statements, they were like........ really easy because it was at the start and everyone is taking all the notes and then you have loops or something and everyone is not getting it because they are not up for it as much as they were’ (P19)

“I picked it up again (level) at the start of this semester. I’m interested in it so it’s not that bad. You have to get your head around it to stop it fading out’ (P19)

Memo – Here we see a strategy to reactivate one’s interest in programming to stop it fading out. By this I am taking it to mean that if programming concepts are not
practiced on a regular basis, the concepts themselves and the rigorous syntax of the language will be quickly forgotten. This is consistent with my professional experience.

'I decided to prioritise on where I want to go and concentrate on other subjects. I’d be happy to pass programming with a lower grade' (P24)

'I switch off' (P24)

'I learn more in the labs that the lectures when I can see things being done. If I’m listening to it and I don’t get it I drift' (P42)

'Don’t let it go by the wayside saying “I’ll do it later”, because you are never going to do it later because you are always progressing. You end up missing the whole thing on functions and haven’t a clue what anyone is talking about. With functions I just didn’t get them at all and I said I’ll get it all sorted out later (duration) but just didn’t bother and it took a whole project to figure it out. The passing of functions and the parameters completely confused me’ (P42)

'I feel if I can’t interact in the class I drift, I start thinking about something else’ (P42)

'In the first semester it was going over my head, may be I was letting myself do that but I’m not now’ (P8)

'If you miss something in Java you just fall behind (outcome/consequence), I mean seriously behind and that’s happened to me from being absent or not listening in class’ (P9)

'I’m totally disinterested in a way sometimes (frequency)’ (P9)

'For a while (duration) I kind of put it to the back of my mind because there were so many other parts of the course and I said I’ll get my head around that eventually [GH] and I kind of left it on the long finger for a while and fell a good bit behind (outcome) and eventually had to approach the lecturer to put on an extra tutorial’ (P8)
## Appendix 2

**Conglomeration**

<table>
<thead>
<tr>
<th>Code: [ENJOY]</th>
<th>Long Name: Enjoyable</th>
<th>In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Reported by participants who find the programming course enjoyable. It must be indicated that for a number of participants this is mirrored with an overt awareness of the difficulty of the course also and therefore we see references to programming being ‘difficult but enjoyable’. This is referred to in a memo below as the notion of a double edged sword where students appear to be torn between the difficulty and enjoyment of programming.</td>
<td></td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td><strong>Dimensions</strong></td>
<td></td>
</tr>
<tr>
<td>Extent</td>
<td>Partial – Full</td>
<td></td>
</tr>
<tr>
<td><strong>P &amp; D Overview</strong></td>
<td><em>Extent</em> relates to how much of the programming course the participants found enjoyable. Given the fact that interviews were conducted towards the end of the second term there was a feeling emanating from the interview that the material got more difficult or tougher and as a result the sense of enjoyment was getting strained. It should also be noted that some of the better performing students as in (P36) find the later and more difficult parts of the course more enjoyable and less tedious than the earlier easier parts.</td>
<td></td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[D] [P]</td>
<td></td>
</tr>
</tbody>
</table>

'I am enjoying it, I found the first semester ok and did well with it, I am finding this semester tougher [D (extent), the programming has upped the tempo’ (P24)

'It has definitely been difficult [D], I wouldn’t say I was struggling with it, but it’s difficult to get into it. Overall it has been enjoyable’ (P39)

'If you go in with an open mind and realize it’s a subtle build up and that you are not going to be dumped with everything then its fair enough and its manageable and enjoyable’ (P41)

'I enjoy project work, especially now we are working on stuff that’s not small and stupid (extent), which you’ll never use.’ (P36)

'It’s the most interesting. You can see it being used, you compile something and it works’ (P35)

'I enjoy it, it’s interesting. I know a lot more than I did at the start’ (P28)

'The programming language looked to me first like a boring task, then when I got used to it I found it interesting and today I spend hours in front of my screen writing java and applet programs’ (P4)
Appendix 2

**Conglomeration**

<table>
<thead>
<tr>
<th>Code : [EXSCH]</th>
<th>Long Name : Existing Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Refers to participants who recount trying to relate programming concepts to those schemes they already possess.</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Matching Level</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td><em>Matching level</em> relates to how well the schemes the participant possesses actually matches the programming concept under consideration. If this is very partial the student may have to generate a new scheme to facilitate the concept</td>
</tr>
<tr>
<td>Associated Categories</td>
<td>[DISENG] [IE]</td>
</tr>
</tbody>
</table>

‘It would depend on how the lecturer had gone about explaining it in the first place, because I normally try and link that in with something that’s already there and just expand on something already inside my mind (matching level) rather than creating an entirely new subject. It really does depend on how the lecturer puts it. I try to think of it in the same terms as the lecturer would put it and just try to work with something similar or just try and integrate it with your own mode of thought.’ (P41)

‘Go through it yourself and figure it out in a way that makes sense in your head’ (P42)

‘It threw me until it was explained using an example I was familiar with. I probably would have listened if that example was given in the lecture instead of drifting off’ (P42)

‘It it’s over 60 or something you get a certain grade, you can relate to “ifs”. Ifs were used with ticket prices and things like that’. When you were doing arrays you have a single ‘i’ and “customer ++” (P24)
Appendix 2  
Conglomeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Long Name</th>
<th>In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GH]</td>
<td>Get Head Around</td>
<td>In-Vivo</td>
</tr>
</tbody>
</table>

**Description**

On first thoughts it was looking like this category should be merged with ‘Click-In’. However on reflection and further analysis of the data it was apparent that this category refers more to gradual coming to terms with a topic than the ‘eureka’ click in moment described earlier.

**Extent**

Partial – Full

**Duration**

Short – Long

**P & D Overview**

*Extent* refers to how much of the material on the programming syllabus the student has to get their head around. In the data we see participants referring to ‘some areas’ and ‘and bits and pieces’. *Duration* relates to how long this feeling of coming to terms with a programming concept/topic actually lasts. In the data we see participants referring to issues like ‘the last day’ ‘for a while’ and ‘at first’.

**Associated Categories**

[SI] [CLK] [CHALL] [D] [DISENG] [INTIM] [MD] [PA] [SLFDBT]

‘The language, I’m still having problems with some areas (extent). It’s alright once you get your head around it, but I still have problems. The way you have to organize everything. It’s the structured thing that gets me. Programming is too structured for me.’ (P42)

‘There are hundreds of different ways to do things, just trying to get your head around it and also the programs are a lot longer and it’s harder to see the whole picture’ (P36)

I’m struggling a bit, I got my head around most of it so just a few little bits and pieces which haven’t quite clicked [CLK]’ (P36)

‘I couldn’t get my head around it until the last day (duration) ’ [SI] (P2)

‘For a while I kind of put it to the back of my mind because there were so many other parts of the course and I said I’ll get my head around that eventually and I kind of left it on the long finger for a while and fell a good bit behind and eventually had to approach the lecturer to put on an extra tutorial’ (P8)

‘Right now I’m doing an applet where I’m doing arrays using a loop that displays the number that is greater than the one beforehand and I can’t get my head around it’. (P1)

‘At the start I had to get my head around it, then it was OK’ (P18)

‘Just getting your head around the whole idea of it at first. It can be daunting.’ (P39)

‘Sometimes I might not get a full grasp of how exactly everything works’ (P36)
'With other things you can see the flow, but then you come to this thing where you have to populate an array with a hundred spaces in a couple of lines. You just find, first off anyway, hard to get your head around' (P22)
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Long Name</th>
<th>In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GUESS]</td>
<td>Guess In-Vivo</td>
<td></td>
</tr>
</tbody>
</table>

**Description**
This is an approach described by participants during lab and project work. When they come to a problematic area they guess by typing in a piece of code, compiling it and see if it works. This is also referred to as 'trial and error'.

**Properties**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random – Planned</td>
<td></td>
</tr>
</tbody>
</table>

**P & D Overview**

*Nature* refers to the type of guess being made. For some students this is totally random where one participant recounts 'type up the stuff and hope for the best'. It should be noted that the data suggests that random changes can be employed by both weak and strongly performing students. One difference uncovered is that the stronger performing students tend to retrospectively assess the tested code as to why and how it works.

**Associated Categories**

[S] [GUESS] [LA] [KB]

'I know it is probably better to spec changes, it shows you actually have knowledge of the subject, but if you are completely lost it is better to do random changes. It worked for me with arrays and out of bounds errors' (P18)

'There is a big element of guesswork, put in a couple of lines and see what happens. Sometimes you can't remember what you have changed and why it works (random). In some of my programs I guessed around it and got it to work and then I had to sit back and print it off, read it and go back and finally figure out how I made it work’ (P42)

'I went to the lab and typed up the stuff and hoped for the best, but with that you don't learn very much. For pointers I didn't get it straight away, I just type in anything that would resemble what it should and just hope for the best. I kept getting errors. (P43)

*Memo* – We see a slightly different dimension of the code here. Here, rather than being a strategic/educated guess, the student just types in code without any apparent mental engagement in the material they type. I have seen this technique employed by students in my professional work. However the net outcome of this disengaged version of this code is usually as described by this participant, i.e. 'you don't learn very much'. In previous dimensions of this code we witnessed more educated guessing which entailed a strategy of backtracking when the guessed code chunk worked to uncover the reason why it worked. The guessing described above is very often in my experience no more than electronic transcription.
‘If I get it to work I usually try to test it to make sure it’s not just a fluke (retrospective assessment). Sometimes I might not get a full grasp of how exactly everything works but I’ll understand how to use if and after a couple of different programs using the same thing differently I’ll understand it in time eventually (planned)’ (P36)

Memo – This is an interesting slant on ‘sink-in’ and ‘guess’ where material that was initially guessed and was not fully assimilated over a period of time with reimplementation in other programs eventually sinks in.

‘I take out bits I don’t understand and see what it does’ (P5)

‘I kind of go through it, take out bits and see what it does, like trial and error, just to see what it does’ (P5)

‘Most useful is a hands-on approach. If you get a grip on it but you are not quite sure how to program it, see how it works and maybe by just making changes to code that has been shown in the notes, copy and paste it and make some changes to it bit by bit (planned). That kind of helps’ (P18)

‘It depends on how well I know the topic that I’m trying to get a grip on. If I don’t know what I’m doing I do trial and error line by line and see what this line does, go to the next one...’ (P18)

‘Usually with programming it’s trial and error. Starting off trying to get one thing to work and then another thing to work. Maybe they might work first time, just take it one step at a time’ (P39)

‘I put them together in whatever way makes them work (random). It would be trial and error, looking over notes and getting help as to why it’s not working from the lecturer’ (P42)

‘Trial and error mostly, it’s the best you can do sometimes. Go through notes, try and code, if you can’t then hopefully you keep trying until you get it’ (P43)

‘Tackle questions and play around with it, a lot of trial and error. It’s really been trial and error on the projects. I write a function for whatever and if that doesn’t work I might have to take it all back to scratch or if I see stuff in a chapter where I can use something else instead of where I was using a couple of variables, I’ll have to take back the whole program and edit it. Since I am playing around with it and I don’t have a full grasp of everything I know bits and pieces, I just try it and if it works “great!”, if it doesn’t I’ll have to mess around some more’ (P36)
<table>
<thead>
<tr>
<th>Code : [IE]</th>
<th>Long Name : Ineffective Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This refers to examples that may take the form of code segments, diagrams, illustrations, simulations etc that are used to explain programming concepts but with which participants find ineffective in getting a programming concept across to them. As can be seen in the data this can have a negative effect on the student.</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td><strong>Dimensions</strong></td>
</tr>
<tr>
<td>Level of Ineffectiveness</td>
<td>Low - High</td>
</tr>
<tr>
<td><strong>P &amp; D Overview</strong></td>
<td><em>Level</em> here refers to how ineffective the example actually is. A low level of ineffectiveness may cause the student to understand all but part of the concept. On the other hand, a high level as reported by some participants may have negative consequences such as not enabling them to understand a concept that might have been possible if a different or more numerous examples had been provided.</td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[D] [EXSCH] [CONF]</td>
</tr>
</tbody>
</table>

'I know we are given these problems to work out ourselves but sometimes from just reading the notes you'd imagine they'd have examples of different things. I remember when we were doing arrays. It just gave two examples to store information about a league table or something like that. Then I thought you could only do that sort of stuff with it, because I was just going by the examples in the notes' (P10)

'If statements and for loops, I got to grips with fairly easily but the stuff we are doing now with the public and private methods. I find that a bit hard [D] It’s easier to understand the “for” and “ifs” because you can put in what they are doing. But the methods, you have to think about it, they are more complex. I’d like a few more examples’ (P21)

'The example in the notes was to do with chess. I didn’t realize what the lecturer was talking about because I don’t like chess at all’ (P21)

*Memo – here we see an overlap with the ‘conscious disengagement’ code. Using an ineffective example in this case caused the student to disengage from trying to assimilate the concept of two-dimensional arrays*

For example, arrays were explained in the form of a grid as in an excel spreadsheet but that didn’t integrate very well with me which probably isn’t a good thing. You need to link with logic that’s already in your head [EXSCH]' (P41)

*Memo – I am reminded of a previous participant who referred to an inadequate example of a chess board to explain two-dimensional arrays. It is interesting to find so many overlaps across multiple sites*
Appendix 2

Conglomeration

'The format of the book is just wrong. There's a little bit of information about the topic and then it goes on to show you a program using the topic but there's just not enough information about it. Why you'd want to have that function or that call, whatever. You feel like there is something missing (level). When you are going through it and there's no variable names, it's all "i", "j" and "k" where we have been using variables like count and number. I don't like the "i" and "j", they confuse me [CONF]' (P42)

'The textbook is not detailed enough, it's just an overall look. They kind of expect you to know what they're talking about straight away. When people become professional they tend to forget what it's like not to know' (P33)

'I'd prefer if it was related to things in the real world and not abstract things' (P24)

'I hate when it has a letter, because I think the examples are much harder [D] to read and much harder to understand when it has "i's" and "+1" (P23)

'I don't know why it's "++", everybody knows "+1" (P24)
### Appendix 2 Conglomeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Long Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[INTIM]</td>
<td>Intimidating In-Vivo</td>
<td>This relates to the student being intimidated by the nature of the programming subject resulting from the quantity of programming material that is covered in a first year syllabus and the interrelatedness of this material in which one topic builds on previous topics.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Low - High</td>
</tr>
<tr>
<td>Duration</td>
<td>Short - Long</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P &amp; D Overview</th>
<th></th>
</tr>
</thead>
</table>
| Level                 | Here refers to how intimidated students become whilst engaging in the programming subject. One participant refers to being 'a little bit intimidated'. Duration relates to how long this feeling lasts. The participants here seem to see this as an initial experience and use terms like 'the first time' and 'at first'

| Associated Categories | [GH]                                                          |

'You are expected to interlink everything you have learned. The first time (duration) you get this it's intimidating' (P37)

*Memo – This is an issue I have experienced with students in the labs where the building block, integrated nature of programming hits home to students doing their projects. Whilst a syllabus can be broken down into discrete parts e.g. input/output, selection control, iteration control etc, the student is required to integrate all of these properly in a project. It reminds me of golf lessons where individual parts of the swing can be discussed individually i.e. stance, posture alignment, take-away etc, but at some stage you have to combine all of these discrete factors into one continuous & seamless swing.*

'When I see theory I get a little bit (level) intimidated. I have a big block of information (quantity) I have to digest now' (P37)

'Initially getting your head around the concept of the language, using arrays, loops everything. Just getting your head around [GH] the whole idea of it at first. It can be daunting.' (P39)
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Long Name</th>
<th>Known Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Description</td>
<td>This relates to a strategy described by students when tacking a programming task. Here they utilize small bits or chunks of code that they know and try and build on them to get the program working. The data illustrates that the effectiveness of this strategy varies from student to student.</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
<td>Level of Effectiveness: Low - High</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P &amp; D Overview: Level here refers to how effective this strategy is for the student. In the data we see different extremes from ‘doesn’t work very well’ to ‘a very good way to learn’</td>
</tr>
<tr>
<td>Associated Categories</td>
<td></td>
<td>[TE]</td>
</tr>
</tbody>
</table>

Since I am playing around with it and I don’t have a full grasp of everything I know bits and pieces, I just try it and if it works “great!” [TE], if it doesn’t I’ll have to mess around some more’ (P36)

‘I give it a bash. If I hadn’t a clue I’d try a roundabout way using if statements or something. It would probably be a lot longer, I’d kind of stick to what I know’ (P10)

‘I’d start off writing the bits I did know, I’d troubleshoot, I’d try and troubleshoot it to see which part was wrong’ (P1)

‘You do the parts you can do first and then you look at the rest’ (P10)

‘Try to break it down into its simplest chunks and what it all contains. Would you use static or not, it’s all the different combinations of classes or whatever you have to use. I try to get a layout or structure on the method’ (P25)

‘We had to create methods to do functions we had already written. I just basically transferred those into methods and that kind of made it much clearer, for example factorial using loops as we would have written those as a standard program. Then when it came to methods we split it up into different methods and that was a very good way to learn it because it made sense’ (P29)

‘By taking a bit at a time, when we have projects I get that bit working then the next bit working. I end up with five or six different functions and I have to put them all together which doesn’t work very well (level), but I get there’ (P42)

‘I’d like to give a small example and then build a bigger one as in go through the stuff’ (P45)

‘I see stuff in a chapter where I can use something else instead of where I was using a couple of variables’ (P36)
Appendix 2

We basically continued with arrays but we put the arrays into context. We had to put a film name, director, and running time in an array and display them as follows, Film Name, Director, Running Time. We had to do this in different contexts i.e. this was for a video, We had to do details on a shopping item and details on a computer. Once the first program was done it was just matter of changing the values for the next programs' (P7)
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code : [LA]</th>
<th>Long Name : Lecturer Assistance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Refers to students seeking the assistance of their programming lecturer. This may entail a full or partial explanation of how to go about a particular programming task or of a concept covered in lectures</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td><strong>Dimensions</strong></td>
</tr>
<tr>
<td>Extent</td>
<td>Partial - Full</td>
</tr>
<tr>
<td><strong>P &amp; D Overview</strong></td>
<td><em>Extent</em> here refers to how much help the student gets from the lecturer on a particular programming task. As indicated in the data, partial may involve the student doing most of it themselves and then asking the lecturer when they are stumped. On the other hand, full will entail the lecturer going through the entire task/code bit by bit.</td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[TE] [PA] [P] [MD] [PA] [SE]</td>
</tr>
</tbody>
</table>

‘If that fails it’s the lecturer or a forum on the internet’ (P37)

‘I wait for the lab with the lecturer, call them over and ask them for help. There is a lot of stuff on the internet but it’s not explained very well’ (P39)

‘If there is a specific problem and there’s someone from the class [PA] or the lecturer is there I’ll ask them but most of the time it’s just sitting there’ (P41)

‘I put them together in whatever way makes them work. It would be trial and error, looking over notes and getting help as to why it’s not working from the lecturer’ (P42)

‘If it works you read back over it to see what went where. If you don’t understand it you grab the lecturer for a second to ask how you did it or else you just try and repeat it again later’ (P43)

‘My lecturer just went through it bit by bit instead of a long paragraph of stuff in the notes’ (P43)

“I ask question in lectures, I try and figure it out first, but if I couldn’t, I’ll ask for help from the lecturer, or ask a friend’. (P14)

‘I try and program it myself, see the error, where did happen, if it’s a problem of concept I go back to the lecturer and ask questions’ (P20)

‘I talk to the lecturer. They kind of sit down with you until you get it (extent) [P]’ (P35)
<table>
<thead>
<tr>
<th>Code : [MD]</th>
<th>Long Name : Material Delivery</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This refers to how the student experiences the delivery of the programming material during the academic year</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td>Dimensions</td>
</tr>
<tr>
<td>Pace</td>
<td>Slow – Fast</td>
</tr>
<tr>
<td>Depth</td>
<td>Low – High</td>
</tr>
<tr>
<td><strong>P &amp; D Overview</strong></td>
<td>Pace refers to how fast the material is gone through in the course. The data suggests that a fast pace for some students may appear to be slow for others who either have prior experience or are picking up programming at a quicker rate. Depth refers to the level of detail gone into with a programming concept/topic. Once again this is viewed differently by each student</td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[LEC] [CLK] [LA] [GH] [D] [SO]</td>
</tr>
</tbody>
</table>

'It says you start off from scratch, but it moves very quick, each week something different, its up to yourself to get on with it’ (P9)

*Memo – here there seems to be evidence of self-realisation, where the participant is astutely aware of their own lack of diligence*

'It is very fast paced (pace), you take in a lot (quantity) in each class and if you don’t understand it you move on to the next class without asking questions [SO], you fall behind (consequence) so you have to ask questions, I ask every class. The pace is challenging [CHALL]’ (P14)

'When I start new concepts like methods, constructors, classes, inheritance etc, all of like in 2 weeks or something, so much to take in and try to put them together’ (P19)

'You don’t have a lot of time to do stuff so it’s tough going [D] (P19)

'Because I have a background in programming, it’s a bit slow at the beginning’ (P20)

'The pace, the other course was at a slower pace. I did VB there, but after 6 weeks here I was into more detail (depth) than the whole year at the other college. This course is more in-depth.’ (P24)

'It goes off really quickly. You think you are on top of it and all of a sudden you are saying “what’s going on?”. You just have to keep on top of it’ (P42)

'You have to cover the stuff so fast, you’re constantly moving. Unless you do revision yourself there is no time to go back over something you miss, even the slightest detail, you are pretty much gone for a while (consequence)’ (P43)
Appendix 2

Conglomeration

'I found the start quite easy when we were doing the “ifs” and I really got the hang of them, but after that we just flew through arrays. It was like all the hard stuff [D] we just flew through it' (P23)

'You’d end up leaving it [SO] because by the time you come to the next lecture you’ve moved onto something else (consequence)' (P24)
**Appendix 2**

**Conglomeration**

<table>
<thead>
<tr>
<th>Code : [LEC]</th>
<th>Long Name : Lecture Notes In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Lecture notes presented to the students throughout the year. These often take the form of theoretical descriptions of programming concepts with relevant code examples. They are often accompanied by associated lab sheets which present lecture material related tasks to the student to be worked on.</td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[W] [PA] [CODEX] [MD] [P]</td>
</tr>
</tbody>
</table>

'I have the lecture notes beside me and keep going through and referring back' (P2)

'I key it in and if it doesn’t work I’d go to the lecture notes. When I get errors I copy and paste them into Google and see why they are not working’ (P2)

'I go to the lecture notes and everything is just big words [W], just jumping out at me’ (P2)

'Just knuckling down and going back over the notes [LEC] as much as I could [P] and asking how other people did things [PA]' (P8)

'It means you are reading back through notes [LEC] more to find answers to your questions, stuff that you could have missed you’re now picking up on’ (P8)
## Appendix 2

### Conglomeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[P]</td>
<td>Persistence</td>
</tr>
</tbody>
</table>

**Description**
This is a common strategy reported by participants in terms of how they approach the programming subject. As is clearly indicated in the data the extent and duration of persistence varies amongst students. Furthermore, the data illustrates that it is one of the most commonly used strategies by both weaker and stronger programming students.

**Extent** Partial – Full

**Duration** Short - Long

**Type** Random - Planned

**P & D Overview**
- **Extent** refers to the different lengths the student goes to, i.e. lecture notes, books, internet, peer and lecturer assistance. One participant refers to this as ‘trying different ways of doing it’
- **Duration** refers to the amount of time the student invests in their persistence. In the data we see students ‘staring at it for an hour’ and trying out things ‘over and over’. **Type** refers to how planned or unplanned the persistence actually is. Some students appear to have a prescribed strategy of persistence from trial and error and looking for any help at all to an apparently defined strategy of starting with the lecture notes and moving on to examples.

**Associated Categories**
- [CODEX] [PA] [LEC] [SI] [ACC] [BW] [ENJOY] [D]
- [CLK] [ALC] [AOO] [CHALL] [LA] [P]

*I keep on googling it until I find something (duration), always looking for code [CODEX]’ (P5)

*The more you work on them the more you find how similar they are, it’s only a few lines of code that are different’ (P3)

**Memo** – This is a situation where students who initially find something difficult, realize when they persist and get there that it actually wasn’t a major departure from what they would have done already. This is consistent with my experience of dealing with students in labs where in some cases when they get something working e.g. extracting a surname from a string, they almost kick themselves for not seeing the solution earlier (which involves changing a parameter or two in a method call in an example where they have already been shown how to extract the first name). This is also matched with a sense of happiness and achievement when it does work especially when the ‘eureka’ moment is encountered.

*I keep trying different stuff until it works’ (P3)

*We have examples of applets [CODEX] using arrays and not applications, I most likely implement the code from the applet into the application and try and modify it till it works [P]. If that didn’t work I’d turn to people in my class [PA] (extent)’ (P3)
 Append" 2 Conglomeration

‘After the lecture I understand what’s been said, in theory I understand but when it comes to keying it into the computer it’s not coming straight out of my head that’s for sure, I’m constantly going back through the notes [LEC]’ (P8)

‘It takes a good bit of work for some programming [SI], you get stumped at them, but you just keep at them [P] then when you get something displaying to the screen that’s supposed to be there and you are putting maybe 20 minutes or more and then you get your goal, you get a good feeling [ACC]’ (P8)

‘You look over the notes and have a go at it again, if you are still stuck you can as a friend or the lecturer to see if you can suss out the problem’ (P1)

‘Keep your head down and work through it’ (P1)

You just have to spend more time (duration) [time investment], any spare time you get, we’d be staying in during lunch breaks doing Java, helping each other out [PA]’ (P9)

‘I keep doing it, repeating, repeating and repeating it until it hits home’ (P2)

‘I just asked questions until I got enough answers to make up my mind and I kept reading the notes, it didn’t take me too long’ (P20)

I’m sitting looking at notes and they are not making any sense to me. I’m there following the flow and still not grasping it, then coming back to it again and still not grasping it’ (P24)

Memo – here we see an instance of where persistence is not paying off for the student.

‘Keep plodding along, go back over the notes and try and see if you can make any sense of them. If not, go on to a new section and just work on that, maybe try and get that end of it right (planned), hopefully what you didn’t know you will be able to overcome it’ (P24)

‘When you see all of this stuff thrown at you at first you say to yourself what’s going on here [BW], but if you stick with it, it’s OK. I feel like “where do I start” but if you go through it step by step it’s OK’ (P10)

‘At the start it was OK but then it got harder, but the fact that I liked it [ENJOY]and kept at, it’s OK again’ (P22)

Memo – here the student’s persistence is driven by his/her liking for the programming subject. We also see that they are alternating between states of enjoyment and not being comfortable with programming, i.e. an instance of the double-edged sword again

‘I’ll still keep practicing. Programming is something difficult enough that I need to keep practicing’ (P22)
‘Just sitting it out. Working it over and over until it clicks [CLK]. There is a moment when it just clicks and you understand it’ (P29)

Definitely practice, it’s the only way that it sinks in. If you get lost you need to find out somehow, from someone or a book, the internet or whatever. If you don’t understand something you are lost from there really’ (P29)

‘Try the example until you get it working [P]. If you get the example you’ve got it nailed down’ (P37)

‘I stare at it for about a half hour. Nine times out of ten I’m stubborn enough that I’ll get it’ (P37)

‘I use hard graft, try to find different ways of doing it. Usually you get a way of doing it that does work’ (P39)

‘I get a copy of the notes for my laptop, sit down and start to read them from scratch and I look at the examples. I read through them again until it starts to sink in (SI)’ (P41)

‘Trial and error mostly (random). It’s the best you can do sometimes. Go through notes, try and code, if you can’t then hopefully you keep trying until you get it’ (P43)

‘Keep reading over and over the notes, if there are any examples [CODEX] at all then keep doing them until you understand, if they are complicated it is hard’ (P43)

‘I just do my best, just keep trying until I can’t understand it, go over it, get help, do everything I can because you can’t really advance if you can’t understand it’ (P44)

Memo – Even though this student is experiencing difficulty in programming they are still overtly aware of the building block nature of programming and that progression is dependent on understanding the foundations/basics

‘Repetition, just studying it and doing it, repeating those over and over again. If that doesn’t work just get help (assistance) just to clarify it so I can repeat it’ (P44)

‘The more you practice, the more you can link to previous things. The more you build up, the more you can link back to’ (P28)
<table>
<thead>
<tr>
<th>Code: [PA]</th>
<th>Long Name: Peer Assistance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Relates to students seeking help from their peers (classmates)</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Extent</td>
<td>Partial - Full</td>
</tr>
<tr>
<td>Nature</td>
<td>One-way - Reciprocal</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>Extent refers to how much help the student gets from their peer(s) on a particular programming task. This varies from partial where they do most of the task themselves to full where their peer goes through the entire solution/explanation. Nature refers to whether this assistance is one-way or a reciprocal situation where assistance involves helping each other in a synergistic fashion.</td>
</tr>
<tr>
<td>Associated Categories</td>
<td>[LEC] [CLK] [SI] [LA] [GH] [CODEX] [P]</td>
</tr>
</tbody>
</table>

'We have examples of applets using arrays and not applications, I most likely implement the code from the applet into the application and try and modify it till it works [P]. If that didn’t work I’d turn to people in my class’ (P3)

'You have to tackle them head on and don’t be afraid to ask somebody’ (P8)

'Just knuckling down and going back over the notes [LEC] as much as I could and asking how other people did things’ (P8)

'I’d check old programs, ask a friend, read the notes’ (P1)

You just have to spend more time, any spare time you get, we’d be staying in during lunch breaks doing java, helping each other out’ (P9)

"I ask question in lectures, I try and figure it out first, but if I couldn’t, I’ll ask for help from the lecturer, or ask a friend’. (P14)

'Mine is the same as yours, this is not working, and then they tell you (where yours is wrong) and you say ah, that helps a little bit, doing the next one and getting it right’ (P19).

'We sit down and work on it, that helps. You’re helping them and they’re helping you and once you get it done it clicks in my head [CLK]’ (P19).

'If you work with other people, you get kind of get their ideas as well as especially if you think to yourself “I’m lost”. It’s good to get people working together, by listening to others and what they are saying, it kind of sinks in with yourself [SI], you kind of learn it from each other (reciprocal )’ (P10)

'We get a group of friends together and bounce ideas off each other as to how the theory is best applied’ (P37)
'If that fails I get one or two of my friends who are more than competent programmers to look at it. If that fails it's the lecturer [LEC] or a forum on the internet' (P37)

'If there is a specific problem and there's someone from the class or the lecturer [LA] is there I'll ask them but most of the time it's just sitting there’ (P41)

'If I had someone there beside me and I had a problem, unless they were really busy and I didn't want to disturb them I'd ask them straight off (one-way)' (P41)

Obviously it's getting more difficult as we are moving along. As a class we are good at helping each other out. We break it down in terms of the variables we have to get in, what form of float, double etc. We don't want to go into too much detail, i.e. it has to be an "if" statement, or it has to be a "for" loop. It's more like getting a rough outline so that everyone's on the right track, instead of one person going off on a tangent and two weeks later realizing they were completely gone the wrong way about it and they'd have no time left to come back and fix it' (P41)

'I spend a couple of hours at it myself and if I can't get it my first place to go is my friends, because all our projects are roughly the same. If they can show me how or what way they are doing it (partial), I might be able to get my head around it [GH]' (P36)
Appendix 2

## Conglomeration

<table>
<thead>
<tr>
<th>Code: [PB]</th>
<th>Long Name: Pressure Build-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Relates once again to the building block nature of computer programming. Given that each concept builds on the previous ones, if earlier concepts are not understood then the students feel a gradual pressure build up over the academic year.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Properties</strong></th>
<th><strong>Dimensions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
<td>Negative – Positive</td>
</tr>
<tr>
<td>Intensity</td>
<td>Low – High</td>
</tr>
</tbody>
</table>

**P & D Overview**

Nature relates to whether the existence of pressure is positive or negative. For some, it is part of the positive challenge of programming whilst for others it becomes too much and the student may feel a sense of being ‘overloaded’ with difficult tasks. Intensity relates to how much pressure build up the student is experiencing. In this regard, students recount ‘getting lost’ and material going ‘over their head’. In order to try and counteract this one participant highlights that the student must somehow ‘keep on top of things’ (dealing strategy).

**Associated Categories**

[D] [ALC] [BW] [CHALL]

---

I’ve a better understanding of it now. I’ve been keeping on top of things and not letting them build up, that’s the biggest thing really (positive)’ (P8)

‘Honestly, the further you go on, the harder it gets [D], at the start it was fine, but it takes a lot of time [time investment], you really have to keep up with it’ (P9)

‘It’s not like other subjects where you can learn a bit and it will be OK, with this it just seems to be all at once, but before that it completely over your head (negative)’ (P29)

‘At the start it’s OK but then it gets more complicated and you start adding in pointers and stuff and it gets worse (intensity)’ (P45)

**Memo** – In my professional experience this is a common occurrence where students suddenly find the material getting difficult at an early stage when they are required to combine concepts. The main problem for student at this stage is that the material doesn’t get any easier conceptually. For example if they are finding arrays of strings difficult using loops, then adding in methods, objects and classes adds more confusion to an already confusing intermingling of concepts.

‘The next week builds on what you didn’t understand anyway so you are really lost’ (P24)

‘It’s time, there’s not a lot of time to practice, you’re in college so much. Just so much to do as well with the labs, you always have problems. If there was one where you
could work on stuff you are not sure on that would be good. It's all about time, there's too much to cover (P23)

'We had 1 hour lecture and 4 hours laboratories per week. Tasks were given to be done during the tutorials time and this work was accomplished for Java Portfolio. The problem was that I could hardly finished before the next tutorials which means that I found myself overloaded by tasks and studies of the weeks' (P4)

'I think that certain areas of java programming need to be studied in great detail and if this isn't done then it makes it very hard to do other tasks because a lot of programming comes hand in hand' (P7)
<table>
<thead>
<tr>
<th>Code : [PPK]</th>
<th>Long Name : Prior Programming Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This relates to the programming knowledge that some students have prior to undertaking the programming course</td>
</tr>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Extent</td>
<td>Partial - Full</td>
</tr>
<tr>
<td>Usefulness</td>
<td>Low - High</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>Extent here refers to how much of the programming syllabus material students may have seen before. For examples a number of students have Visual Basic (VB) experience and in that case do not have prior exposure to objects, methods and classes. Usefulness pertains to how useful the prior knowledge is in understanding the material on the programming course. This varies amongst participants.</td>
</tr>
<tr>
<td>Associated Categories</td>
<td>[D] [ALC] [SI]</td>
</tr>
</tbody>
</table>

‘Classes were difficult to understand, not classes, but the concept of classes. As I hadn’t seen it in Visual Basic before, it was something new, apart from that I didn’t have any problems. Classes were a major difference to what I had seen beforehand’ (P20)

‘At the start I found it very easy because it was mostly stuff I’d seen before and I applied what I knew already (usefulness) to Java’ (P29)

‘I did VB there, but after 6 weeks here I was into more detail than the whole year at the other college (extent). This course is more in-depth.’ (P24)

‘It’s the Java language, you have to know everything (quantity) to go and do it. In VB you have to know the code as well but it’s more straightforward’ (P18)
Appendix 2  
Conglomeration

<table>
<thead>
<tr>
<th>Code: [SE]</th>
<th>Long Name: Self Example</th>
</tr>
</thead>
</table>

**Description**

Reported by participants who come up with their own tasks or applications to complete using a computer program. As is evident in the data, this can be done for a variety of reasons from self-explanation to fitting in with current schemes to avoidance of boredom.

**Properties**

<table>
<thead>
<tr>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Rationale</td>
</tr>
</tbody>
</table>

**P & D Overview**

Size relates to how small/simple or large/elaborate the examples are and this may be determined by the rationale property. Rationale relates to the driving impetus behind the self example. The data suggests that is on a continuum. Explanatory may entail the simplest possible example to explain a concept. Current schemes may entail creating an example that fits in with something the student already understands, i.e. a current scheme. More interesting, as the data suggests entails the student doing an example that appeals to them (e.g. one of their hobbies) in order to prevent boredom and to make the material more appealing.

**Associated Categories**

[LEC] [CODEX] [SI] [D] [ACC]

‘If I find stuff boring I go off and to my own stuff (rationale – more interesting)’ (P5)

Followed by long narrative on converting golf example to data base

‘For example, an “if” statement, I wouldn’t copy it but make my own thing of it’ (P18)

‘I did my own notes based on the lecture notes examples [LEC]’ (P19)

‘I try to write my own programs. We are doing maths problems and just to practice, I wrote a program to do it. I find the tasks I am given, I don’t know about them [D]. I prefer an application you know yourself (current scheme), you get a lot more satisfaction about it (outcome)’ (P22)

_Memo – this is an interesting issue, is it perhaps that difficulty is exacerbated by using examples in lecture notes and lab sheets that students do not relate to?_

‘Practice a lot, just try and write your own little (size) programs and get them to work, add stuff to your programs from lecture notes [LEC]’ (P21)

‘I come up with problems or try and think of applications to test everything. I find it harder to remember something if I couldn’t see a practical application for it’ (P29)
'If you gave me the same thing a thousand times it wouldn’t mean a thing unless I applied it in some way. Once I have an example in front of me I can use it [CODEX]' (P37)

'I think of how is the best way this is going to be applied. For example, your functions, they are going to tidy up your code and make it easier to read. You can go down and check the function and come back up again. Basically how is the best way it can be applied to your own overall code. That’s the first thing I do when I try to integrate something new’ (P37)

Memo — Here when the student discusses integrating something new we see an example of ‘accommodation’ where the student either alters existing schemes or develops a new one to digest this new concept they encounter in programming

'I like to know what’s the use for it’ (P39)

'I go over the topic in a lab class and I try to do a simple (explanatory) test myself involving that topic, it sinks in a lot faster that way [SI]. I find it faster to learn it if I’m doing an example myself as I’m going along. You are better off trying examples and fixing errors rather than waiting until the exam and finding out that you haven’t a clue at all’ (P41)

'When I discovered functions I made a silly text based game and was just jumping from function to function and I learned from doing that. When it came to the first project which involved functions I was ahead of the class. I was showing my friends and I was comfortable (outcome) with it’ (P36)

'I do a lot of extra little things that you set yourself to do, imitate a calculator or something’ (P35)
Appendix 2

**Code**: [SLFDBT]

**Long Name**: Self Doubt

**Description**
Relates to participants doubting their ability for programming and at different stages questioning whether or not they should be on the course.

**Properties**

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Level</th>
<th>Extent</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Low - High</td>
<td>Partial - Full</td>
<td>Short - Long</td>
</tr>
</tbody>
</table>

**P & D Overview**

*Level* relates to how much doubt the student is experiencing, in the data we see terms like ‘not a clue’ and ‘not comfortable enough’. *Extent* relates to how much of the material throughout the year they get this feeling with. Here some refer to the ‘whole coding’ process, whilst some are ‘fairly comfortable’ with some of the material e.g. loops. *Duration* pertains to how long this self-doubt actually lasts. One participant recounts not progressing at all and being ‘stuck’, another asks themselves ‘when is the understanding going to happen’.

**Associated Categories**

[D] [CLK] [ALC] [CONF] [BW] [AOO] [GH]

'I'm still having problems with loops now and again, I have the gist but I wouldn't be a great programmer now' (P9)

'They (loops, ints, booleans etc.) were something you never done before [ALC], and you are sitting there like, am I doing that right, I wanted to be sure' (P14)

'It gets tough [D], at the start you are like trying to learn a whole new language [ALC] which I'm really bad at (P19)

I'm getting nowhere with it, I don't have a clue (level) what is going on after doing the lecture today. You are at home sitting in front of your computer and then you start to think you are on the wrong course. You can't do this, you just can't figure out why it is there, what's the point of having a class separate to there, inheritance, just put it all into one class and do it like there, having separate classes, all doing inheritance, interfaces and all, its just crazy [CONF]' (P19)

**Memo**

Here we see an good example of cause and effect where self doubt and difficulty in the material sinking in cause the student to doubt whether or not this is the right course for them. Further on in the data we see examples of how the various Java and object oriented concepts leave the student feeling exasperated. It seems that not only is it the concepts but the amount of separate concepts (quantity) that exacerbates this feeling of bewilderment and confusion.

332
‘I still don’t think it is easy. I didn’t feel comfortable going into my programming assessment. I know I didn’t know the stuff well enough, even though I had tried, I don’t see the logic of it and am not able to learn it’ (P24)

‘I don’t feel personally as progressing, I’m stuck in this area. The area is trying to grasp the whole coding and how you lay it out and things like that. Even that public class thing with main’ (P25)

‘I’m hoping that somewhere along the line I’ll get an understanding of this, when is it going to happen (duration)? When is it going to click in [CLK]? When you are at home on your own, it’s very difficult [D] to do it. It feels like you’re beating your head off a wall or something’ (P25)

‘It’s all new and you will be awestruck, you won’t have a clue what is going on. You say to yourself, I’m never going to get this at all but it comes’ (P29)

‘I’m fairly comfortable with functions at this stage (extent), but I’m not invincible when it comes to the subject but I’m steady enough’ (P41)

‘The first project was dodgy because I was still unsure about functions at the time. I was having a hard time, with a lot of errors in my code. There are always a couple of bugs that I never get fixed at the time. You run out of time. I’m not an amazing programmer by any stretch of the imagination, it takes me a fair while to work out a topic [SI] in terms of if there is a bug and I can’t wrap my head around it [GH] and my friends they don’t know what’s going on either [AOO]. The lecturer is being pestered and you are sitting there trying to work your way through. The time quickly evaporates (time factor)’. (P41)

‘Algorithms and stuff are OK, I can sit down and work my way through them. But when the lecturer is going through loops and stuff like that I try to follow but sometimes I just can’t. I say to myself “what’s going on?”’ (P42)

‘With programming I’m stuck, I just feel I’m stuck and I need help’ (P44)

‘I find it hard to relate a concept to a question you get in an assessment. You just don’t see a relation there sometimes. You have a basic layout of what you want to do but you are stuck’ (P33)

‘When I can’t understand something I just feel like “am I doing the right thing? Should I be doing programming”’ (P22)

‘When you try and put it all together and you just can’t get it working, you just feel like “should I be doing this? Why am I doing this?” In an exam when it’s actually given to you it’s grand but when you have to write it yourself it’s hard [D]’ (P23)

‘When you see other years doing projects and you say to yourself what could I do a project on’ (P23)
Memo – this is an interesting angle on self-doubt where the student not only doubts their ability in relation to first year material but also projects this doubt forward to their subsequent years on the course.
## Appendix 2 Conglomeration

**Code : [SI]**

**Long Name : Sink-In In-Vivo**

<table>
<thead>
<tr>
<th>Description</th>
<th>Refers to the accommodation of new programming material by a student. It may take a considerable amount of time between a concept being delivered (i.e. their first exposure to it in a lecture or lab) and its accommodation by the participant. Given that programming is a considerably fast moving subject with subsequent concepts often dependent on understanding of prior concepts this code is very interesting. It seems from interviewing students that often projects and tests may sometimes be given to them without adequate time being allocated for the test/project material to have being fully understood as a result of exposure to adequate practice or clarifying examples.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Dimensions</td>
</tr>
<tr>
<td>Level</td>
<td>Low – High</td>
</tr>
<tr>
<td>Time</td>
<td>Short – Long</td>
</tr>
<tr>
<td>Awareness</td>
<td>Low – High</td>
</tr>
<tr>
<td>Quantity</td>
<td>Partial – Full</td>
</tr>
<tr>
<td>P &amp; D Overview</td>
<td>Level relates to how much of a given concept(s) actually sinks in with the student, this may be higher for the earlier material than the later material like methods and classes. Quantity refers to how much material in the syllabus actually sinks in with the student. The data suggests that material covered at an earlier stage may sink in at a much later stage than envisaged. Time relates to how long it takes for topics/concepts to sink in. One participant describes this as ‘taking ages’ and another mentions ‘hours and hours’, whilst others refer to ‘weeks’. Awareness refers to how aware the student is of the material either sinking in or not. The data suggests that for some students this is low where it’s not until an assessment or test that the student realizes that the material hasn’t sank in, for others they are overtly aware of their low level of material actually sinking in.</td>
</tr>
<tr>
<td>Associated Categories</td>
<td>[D] [CLK] [GH] [PAC] [BW] [P] [PPK] [W] [CONF] [AOO] [GUESS] [MD] [PA] [SE]</td>
</tr>
</tbody>
</table>

‘Well usually, I don’t understand it at the start, it kinds of clicks [CLK] in at the end (time), I keep on doing it [P]’ (P5)

‘They were difficult to pick up for the first two weeks because of the difference between applets and applications’

‘You just can’t take it all in (level), what an applet actually is, even the “hello world” applet is still longer, you have to import stuff, albeit 8 lines’ (P3)

It’s a bit difficult [D], there’s a lot to learn like all those arrays and stuff, I don’t know I’m not catching it in lectures’ (P2)
‘I couldn’t get my head around it [GH] until the last day’ (P2)

‘It’s not until you actually start doing it that you realize what you don’t fully understand (awareness)’. (P1)

*Memo – this seems to be another angle to the sink in code, where perhaps the participant wasn’t as conscious of the material not sinking in and this realisation only hits them when they tackle a problem that utilises the programming concept.*

‘You are taking in a lot (quantity), I find it overwhelming sometimes, but it is interesting all the same, so you are not just sitting there, you’re interested in it so, but it can be overwhelming definitely’ (P14)

‘When I start new concepts like methods, constructors, classes, inheritance etc, all of like in 2 weeks or something, so much to take in and try to put them together (quantity)’ (P19)

“I don’t learn a lot when I’m sitting in a lecture room because it doesn’t go into my brain, so I’m just sitting there trying to learn it’ (P19)

‘I look at my notes, talk to someone in the class that I do programs with or something and understand as much of it as I possibly can, because I don’t think one lecture on the subject is enough to get it’. (P19)

I’m sitting looking at notes and they are not making any sense to me. I’m there following the flow and still not grasping it, then coming back to it again and still not grasping it’ (P24)

‘All of this is coming at you fast [MD] and you are being asked do you have any questions. But you don’t know because this is your very first time seeing something, so you are not going to take it all in until you go back and read the notes. A lot of the time in a lecture you cover a lot of material. You might cover a topic in the space of an hour maybe and I find I had to take it all in. You’d understand it while the lecturer is explaining what is going on but when you sit down to do it yourself you say to yourself “what was the lecturer saying there?”’. You read back over the notes and it looked easy when they were doing it. When you are doing it yourself step by step on the computer it probably sinks in a bit better rather than the lecturer just calling it out’ (P10)

‘Well in lectures, it’s hard to take it all in at once you know’ (P10)

“I’m usually lost and after a week or two new stuff is introduced and the earlier stuff starts sinking in. At the same time the term is only 13 weeks long and if it doesn’t sink in you start to panic’ (P10)

‘With almost everything at the start I didn’t have a clue what it meant, it involved a lot of work, it wasn’t just a matter of coming to class and understand what the lecturer was saying. It took hours and hours outside of college, reading up on things [P]. It took a lot of work for me to understand concepts like methods, interfaces,
constructors, things like that, just to actually understand what they do, how they work’ (P22)

‘I haven’t problems with i’s and j’s anymore, I like them now. When I first saw them I didn’t know what they represented, it was obvious when I found out’ (P22)

Memo — Alternative dimension to this category where we see when the material clicks in the student experiences a feeling of it being OK and asking themselves why didn’t they get that in the first place

‘I have looked at one or two examples of memory allocation but it still hasn’t fully sunk in with me’ (P37)

I don’t get it when it is happening, the lecturer just talks for 10 minutes about something and you still might not know what he/she means’ (P39)

‘There have been a couple of topics where I’ve sat there and realized that I didn’t get that topic at all. I just read through the notes a couple more times, thinking about it and it sort of clicks [CLK] a little bit. I’d say for the topic to fully integrate into my mind it would take ages for me (time). To get the initial sort of point behind a topic it normally only takes a day or so, not that often that would happen in lectures because the topics are so intense.’ (P41)

‘You are sitting there and wondering what is the lecturer talking about and that’s hard [D] [BW]’ (P42)

‘When I see code on the board I say “what is the lecturer talking about?” I have to go home and read the notes again and think about it in a different way, in a way that makes sense to me’ (P42)

‘Functions took me a long time to get the hang of and pointers did too. They were the two biggest ones. Everything else I got fairly OK but with those two in particular, if you miss something at the beginning, it is very hard to catch up (outcome)’ (P43)

Memo — Here the student is specific about the programming concepts that he/she is having difficulty with. It is noteworthy here that this student refers to ‘being gone’ and trying to ‘catch up’. This reminds me with other similar outcomes described by other participants where they describe being ‘lost’ as a result of the material not sinking in.

‘The passing of functions, I couldn’t get the hang of it for a long, long time and passing parameters to them and all that sort of stuff. It took me a good few months. For the whole first project I couldn’t get functions. They got explained to me a couple of times and I still couldn’t get it properly’ (P43)

‘Sometimes being told about it in lectures, it doesn’t click [CLK] in your head. Whereas if you are actually sitting down and doing it yourself you kind of understand exactly what’s going on. Not all of the time, some topics might not have a problem at all but some things I don’t think you can really learn at all from notes. You just need the experience behind you to be able to do it’ (P36)
'I understand things too late. We are done classes now and we will have a test on them soon and more than likely I'll fail that. But in a months time I'll start to understand it. That's the way a lot of things have gone for me on the programming course. I understand a bit too late but I usually get to grips with it in the end' (P44)

'In a lecture you just sit down and listen and you only retain about 15% of the information. If you did a kind of a lecture in a lab you'd get more because you are doing it' (P44)

'It's hard to grasp at the start, because there's a whole lot of stuff being thrown at you and it's a lot of information. I try to understand what the lecturer is saying on the spot because if I don't the chances are I won't get it by myself. I try to pay attention in the lectures and understand what the lecturer is saying (P33)

'I understood the chapters only because I'd seen the stuff before [PPK]. The language of programming takes a while to sink in. They (book authors) assume you know the terms [W] straight off' (P28)

'Methods, I couldn't get them until I did a lab on them. I didn't see why you'd put stuff into methods' (P35)

'Now you can see at this stage how loops are applicable, whereas first it was confusing [CONF]' (P22)

'I think more time could be spent on stuff' (P23)

'It took me about 2 extra hours to feel comfortable with it before the next lecture or tutorial' (P4)
## Appendix 2

### Conglomeration

<table>
<thead>
<tr>
<th>Code : [SO]</th>
<th>Long Name : Skipping Over</th>
<th>In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Reported by participants who skip over programming material when it doesn’t make sense to them. Given the building block nature of programming, the actual stage in the syllabus when material is skipped over and how much that material is required on the course may determine how detrimental the skipping over may be. For example if one takes selection control structures “if” and “switch” in Java, skipping over “if” may have more serious consequences than skipping over “switch”. The reason for this is that the “if” statement is used in practically all programs from the point it is covered onwards as well as being able to carry out any operation the “switch” can, however the reverse is not true.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>Small - Large</td>
</tr>
<tr>
<td>Duration</td>
<td>Short - Long</td>
</tr>
<tr>
<td>Importance</td>
<td>Low – High</td>
</tr>
</tbody>
</table>

**P & D Overview**

*Amount* relates to the amount of material and concepts on the course the student actually skips e.g. ‘a lot’. *Duration* refers to how long the material/concept is actually skipped over, this can vary from ‘a day or two’ to ‘weeks’ or to indefinitely depending on the perceived importance of the topic for future work tasks. *Importance* pertains to the perceived importance level of the skipped concept/material. This is high if it is considered that this material will come back to bite students because it will be ‘needed later’. On the other hand this property may be low if the topic is considered to be ‘minor’ or ‘never seen again’

**Associated Categories**

[D] [MD] [W]

---

'I kind of skip over the big words [W] where I know the big words actually mean something and have relation to the things I can't get’ (P2)

'If I come across something I don’t understand I skip it’ (P2)

'I skipped over a lot (amount) of big words [W] (P2)

'If I don’t understand, I knuckle down and don’t move on until that’s clarified, not to skip over stuff that I only half understand because it will come back to haunt you (outcome)’ (P8)

'It is very fast paced [MD], you take in a lot (quantity) in each class and if you don’t understand it you move on to the next class without asking questions, you fall behind (consequence) so you have to ask questions, I ask every class.'
‘Sometimes you can’t do the worksheets, if you leave them for a while, sometimes they put up the solutions or they might run through them. You leave them for a week or two (duration) and go back to it. I like lots of little problems and get all the solutions the next week’ (P10)

Memo – Here we see a more positive angle on the ‘skip-over code’ where the student has a deliberate strategy of leaving the problem for a while and tackling it again when the lecturer does up the solutions. By reading through the solutions and finding where they were stumped in the first place is a technique used by some students. This approach is in contrast to the other types of skip-over where the student just blanks out that chunk of material and doesn’t return to it.

If I find the work I’m given hard to do, I’ll leave it and do something similar. I’ll still keep practicing. Programming is something difficult enough that I need to keep practicing’ (P22)

Memo – Once again we see a variation on the skip-over code where the student doesn’t skip concepts but skips the lab question pertaining to the concept and does one of his/her own, i.e. one that appeals more to him/her or one that they can relate to more

‘I’d normally leave it there and forget about it and go back to it a day or two later and have another bite at it.’ (P25)

‘Don’t overlook something if you don’t get it first time. I can honestly say if I hadn’t kept trying with parameter passing I would have sank a long time ago. If you find something difficult get it fixed. It saved my life for parameter passing (outcome). What ever happens don’t ignore it if you are messing it up’ (P37)

Memo – Here the student is overtly aware of the ‘skipping-over’ strategy and deliberately trying to avoid it

‘The temptation to still overlook memory allocation is still annoying me. I want to get it nailed down. It’s the only thing on the course that I couldn’t get nailed down. The first few times I saw memory allocation I said I haven’t a clue what this is and I’ll come back to it at some stage and saying it will be alright, I’ll never see it again (importance level)’ (P37)

Memo – This is very interesting, just after stating that they deliberately avoided the skipping over strategy, the student subsequently states that the temptation to employ this strategy is ever present. This presents to me another example of the higher level concept of the double-edged sword nature of programming where some students oscillate between various states of mind and strategy employment throughout the programming course. The self-justification employed where they say that they’ll never see it again is interesting. This brings up another issue in relation to concept frequency. In particular, skipping over a less frequented topic like memory allocation may have less disastrous consequences than skipping over assignment statements where assignment statements are used in practically every program the student will ever write. Given the building block nature of programming, the earlier the skip-over
strategy is employed the more dangerous it is for the student's progress (consequence)

'I sometimes leave something half done and come back to it when inspiration strikes' (P39)

Memo - Perhaps the arrival of inspiration is an intervening condition

'If you didn’t catch one part and you wouldn’t be able to do the other part (outcome), because you’d be getting errors left, right and centre. I’d go on to other parts of the project unless I had access to someone in the meantime'

Memo – This is an interesting alternative version of skipping over. Here rather than skipping over and pretending the problem isn’t there the student uses it as a deliberate strategy to work on something else within their control until they are in a position to gain assistance

'If you are in a big project and there is a lot to do and that was a minor part then you’d leave it. If you had time you’d go back over it, but if it’s a big project you’d just keep moving, to try and get through it' (P43)

Memo – Here we see a slightly more strategic decision making process by this student. If the guess worked and it wasn’t a fundamental part of the project then they’d skip over trying to backtrack and work out why it worked. In this case given that it is a strategic decision perhaps in this case ‘by-pass’ would be a better term than ‘skip-over’

'Make sure you keep going over it, because if you don’t understand it then don’t skip over to the next topic because you’ll need what you just skipped. If you don’t get something don’t just go on to the next topic, make sure you go back over it. I missed a topic and skipped to the next chapter. Then I had two full chapters to learn off really fast just to catch up again’ (P43)
Appendix 2

Conglomeration

<table>
<thead>
<tr>
<th>Code : [W]</th>
<th>Long Name : Words In-Vivo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This was a surprising code to emerge in the data where participants referring to programming material as ‘words’ with which they attach a level of meaningfulness. The use of the word ‘term’ is also used in this context.</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td><strong>Dimensions</strong></td>
</tr>
<tr>
<td><strong>Meaningfulness Level</strong></td>
<td>Low - High</td>
</tr>
<tr>
<td><strong>P &amp; D Overview</strong></td>
<td>Meaningfulness Level relates to how much a new word/term means to the student when it is encountered. In the data we see that some students would like words/terms actually described in a way that enhances meaning.</td>
</tr>
<tr>
<td><strong>Associated Categories</strong></td>
<td>[LEC] [BW] [SO] [SI]</td>
</tr>
</tbody>
</table>

'I’d use words I wouldn’t write a for loop, I’d write down the words of the for loop' (P3)

'I go to the lecture notes [LEC] and everything is just big words, just jumping out at me, I don’t understand it and say “OK”, so I kind of skip over [SO] the big words where I know the big words actually mean something and have relation to the things I can’t get' (P2)

'I skipped over a lot of big words' (P2)

'I don’t really follow because it has all these words that mean nothing to me (level). I have read over them but most of the time I wouldn’t have a clue what’s going on [BW]' (P2)

'I got a programming book and it was double Dutch, but now that I know some of the terms it makes a bit of sense (level). It’s readable and I do want to do some code out of it and see if I can follow its code' (P24)

'A language dictionary explaining the terms would be useful' (P24)

'Keep reading. I’m trying to get my brain to get an understanding of it. Sometimes I don’t really know what I’m reading about' (P25)

'Well using stuff like arrays. That stuff really doesn’t do what it says, the name doesn’t help a lot' (P39)

Memo - This is a very interesting contrast between the last two codes. Here when the student sees something that they can ascribe a literal meaning to it helps in the assimilation of it but coming across stuff that doesn’t have a literal association with what they are familiar with introduces difficulty. I am reminded here of what one participant (P24) mentioned earlier that having a dictionary that explained these
words would be useful. It is very interesting to see these same concepts re-emerging across multiple sites

My lecturer just went through it bit by bit instead of a long paragraph of stuff in the notes' (P43)

Memo — here even though the student doesn’t explicitly state the term ‘words’, this description of ‘long paragraphs of stuff’ reminds me of the description of meaning less ‘words’ described by earlier participants

Programming is all new words that you have to learn. In programming it’s write a loop and put in variables and all this stuff’ (P45)

‘I think it would be better to get a full description of everything at the start and why you use it’ (P23)

‘A simple description in lay man’s terms and then a more detailed description would be ideal’ (P24)